Static Analysis and Optimization of Semantic Web Queries

ANDRÉS LETELIER, PUC Chile
JORGE PÉREZ, Universidad de Chile
REINHARD PICHLER and SEBASTIAN SKRITEK, Technische Universität Wien

Static analysis is a fundamental task in query optimization. In this article we study static analysis and optimization techniques for SPARQL, which is the standard language for querying Semantic Web data. Of particular interest for us is the *optionality* feature in SPARQL. It is crucial in Semantic Web data management, where data sources are inherently incomplete and the user is usually interested in partial answers to queries. This feature is one of the most complicated constructors in SPARQL and also the one that makes this language depart from classical query languages such as relational conjunctive queries. We focus on the class of well-designed SPARQL queries, which has been proposed in the literature as a fragment of the language with good properties regarding query evaluation. We first propose a tree representation for SPARQL queries, called pattern trees, which captures the class of well-designed SPARQL graph patterns. Among other results, we propose several rules that can be used to transform pattern trees into a simple normal form, and study equivalence and containment. We also study the evaluation and enumeration problems for this class of queries.

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

General Terms: Theory

Additional Key Words and Phrases: Optimization, query containment, RDF, Semantic Web, SPARQL

ACM Reference Format:
Letelier, A., Pérez, J., Pichler, R., and Skritek, S. 2013. Static analysis and optimization of semantic web queries. ACM Trans. Datab. Syst. 38, 4, Article 25 (November 2013), 45 pages.
DOI: http://dx.doi.org/10.1145/2500130

1. INTRODUCTION

The Semantic Web is the initiative of the World Wide Web Consortium (W3C) to make information on the Web readable not only by humans but also by machines. The *Resource Description Framework* (RDF) is the standard data model for the Semantic Web, and since its release as a W3C Recommendation in 1999 [Lassila and Swick 1999], the problem of managing RDF data has been in the focus of the Semantic Web community. As a result, the language SPARQL was proposed as a query language for RDF, and became a W3C Recommendation in 2008 [Prud'hommeaux and Seaborne 2008]. Since the appearance of these standards, the Web has witnessed a constant growth

This work was funded in part by Marie Curie action IRSES under grant no. 24761 (Net2), by the Vienna Science and Technology Fund (WWTF) through project ICT08-032 and ICT12-15, and by the Austrian Science Fund (FWF): P25207-N23. J. Pérez was supported by Fondecyt grant 11110404 and by VID grant U-Inicia 11/04 Universidad de Chile.
Authors' addresses: A. Letelier, Department of Computer Science, Pontificia Universidad Catolica de Chile, Chile; J. Pérez, Department of Computer Science, Universidad de Chile, Chile; R. Pichler and S. Skritek (corresponding author), Faculty of Informatics, Technische Universitat Wien, Austria; email: skritek@dbai.tuwien.ac.at.

DOI: http://dx.doi.org/10.1145/2500130

in the amount of RDF data published online. Moreover, the advent of huge initiatives like Open Linked Data [Berners-Lee 2006; Bizer et al. 2009] and Open Government Data [HM Government 2012; US Government 2012] that use RDF as a core technology, and the use of RDF in several diverse areas such as bio-informatics, social networks, and data integration, have increased the attention of the research community to study RDF and SPARQL from a database perspective.

Several particular issues of RDF and SPARQL pose new and interesting challenges for the database community [Arenas and Pérez 2011]. In fact, several research efforts have been pursued towards understanding their fundamental properties and developing specific techniques to efficiently deal with these technologies [Pérez et al. 2006a, 2009; Abadi et al. 2007; Polleres 2007; Weiss et al. 2008; Sidirourgos et al. 2008; Angles and Gutierrez 2008; Schmidt et al. 2008, 2010; Neumann and Weikum 2010; Arenas and Pérez 2011]. Nevertheless, and despite the importance of static query analysis, and in particular of query containment and equivalence for optimization purposes, research on the static analysis of SPARQL queries has received little attention so far (notable exceptions are Serfiotis et al. [2005] and Chekol et al. [2012]). The study of static analysis considering the peculiarities of SPARQL and, in particular, query optimization, containment, and equivalence, constitute the main focus of this article.

Let us briefly recall that the data model underlying RDF data is a (directed, arc-labeled) graph. An RDF graph is composed of RDF triples of the form (s, p, o). Regarding SPARQL, its basic constructor is the *triple pattern*, which is essentially an RDF triple that can have variables. The most basic fragment of the language are conjunctions of triple patterns, realized in SPARQL by using the AND operator (see Section 2 for a formal introduction of the language). Thus, if one thinks of RDF graphs as sets of tuples, a triple pattern is essentially a ternary relational atom, and basic SPARQL queries are essentially relational Conjunctive Queries (CQs). In view of this connection, the rich body of work on static analysis and query optimization on relational CQs including the study of equivalence and containment can be immediately carried over to the conjunctive fragment of SPARQL. Moreover, most of the research focused on statistics, indices, and storage optimization has been concentrated on this fragment [Abadi et al. 2007; Stocker et al. 2008; Weiss et al. 2008; Sidirourgos et al. 2008; Neumann and Weikum 2010].

However, when one goes beyond the SPARQL conjunctive fragment, the whole picture changes and the language becomes considerably more complicated. Of particular interest is the *optional matching* feature, which has been the focus of most of the theoretical work regarding this language [Angles and Gutierrez 2008; Pérez et al. 2009; Schmidt et al. 2010; Arenas and Pérez 2011]. The idea behind optional matching, realized in SPARQL by the OPT operator, is to allow information to be added if the information is available in the data source, leaving unbounded some variables if no matching for them exists. Thus when evaluating a SPARQL query containing the OPT operator only a subset of the variables may be bound in answers. This feature is crucial in Semantic Web data management, where data sources are inherently incomplete and have only partial knowledge about the resources that they are modeling. Recent experimental works [Gallego et al. 2011; Picalausa and Vansummeren 2011] show that the use of the OPT operator in practice is substantial. For instance, in a query log obtained from the DBPedia SPARQL endpoint [DBPedia 2012], after duplicate query elimination, more than 45% of the analyzed queries use the OPT operator [Picalausa and Vansummeren 2011].

The importance of the OPT operator has also been recognized from a database theory point of view. It has been shown that the combined complexity of SPARQL query evaluation (i.e., checking whether some set of variable bindings is a solution) raises from PTIME-membership for the conjunctive fragment to PSPACE-completeness when

OPT is considered [Pérez et al. 2009; Schmidt et al. 2010]. In Pérez et al. [2009], the class of *well-designed* SPARQL *graph patterns* was introduced as a fundamental fragment of OPT queries with good behavior for query evaluation (for a formal definition, see Section 2). In particular, it was shown that the complexity of the evaluation problem for the well-designed fragment is coNP-complete [Pérez et al. 2009].

In this article we embark on the static analysis of SPARQL queries containing the OPT operator. We focus on the class of *well-designed* SPARQL *graph patterns* mentioned before. As our first contribution we introduce a tree representation of SPARQL queries called SPARQL *pattern trees*. We also introduce two particular classes of pattern trees, the class of *well-designed* pattern trees and a relaxation that we call *quasi well-designed* pattern trees (QWDPTs, for short). We focus on QWDPTs since they are simpler to work with, enjoy several desirable properties, among others, the existence of normal forms, and more importantly, although QWPDTs are a strict syntactic relaxation they capture the complete class of well-designed SPARQL queries. We introduce a procedure to evaluate QWDPTs in a top-down way that resembles a top-down evaluation of graph patterns proposed in Pérez et al. [2006a]. We also incorporate projection as a top-level operator for QWDPTs thus covering an important fragment of SPARQL.

Notice that previous works on optimization of SPARQL have mainly focused on rewriting queries based on properties of particular operators [Pérez et al. 2009; Schmidt et al. 2010]. We propose transformation rules for QWDPTs that work at the level of the structure of the trees (and thus, the structure of queries). These rules are, for example, capable of eliminating several sources of redundancy in queries, and thus can be used for query optimization purposes. Moreover, most of the theoretical work regarding SPARQL optimization has considered only graph patterns (that is, without including projection) [Pérez et al. 2009; Arenas and Pérez 2011]. We develop some rules that are specifically applicable when projection over QWDPTs is considered.

Based on our work on the structure of pattern trees, we study the fundamental problems of checking equivalence and containment of SPARQL queries. It is known that full-SPARQL and first-order logic have the same expressive power [Angles and Gutierrez 2008]. From this result it is not difficult to prove that equivalence and containment for SPARQL in general are undecidable problems. We show that the equivalence problem for QWDPTs (and therefore, for well-designed SPARQL graph patterns) is NP-complete. The difficult part of the proof is the NP-membership. Recall from the relational world that equivalence and containment are closely related to the search for homomorphisms. The key to our NP-membership result is an appropriate extension of homomorphisms to QWDPTs—leading to the notion of *strong homomorphisms* (for details, see Section 5) and a normal form via the transformation rules for QWDPTs mentioned earlier. When projection is considered, the problem becomes harder. More specifically, we show that equivalence of QWDPTs with projection is Π_2^P-hard.

For the *containment* of queries we consider the *subsumption* relation [Arenas and Pérez 2011]. As detailed previously, solutions for queries containing the OPT operator are essentially incomplete and may possibly bind only a subset of the variables in the query [Prud′hommeaux and Seaborne 2008; Pérez et al. 2009]. This naturally leads to the notion of subsumption between solutions: a solution μ_1 subsumes another solution μ_2, if μ_1 extends μ_2 with more variable bindings. More generally, a SPARQL query T_1 subsumes another SPARQL query T_2 if, for every RDF graph G, every solution of T_2 is subsumed by some solution of T_1. It has been argued that subsumption is a meaningful way of comparing the result of SPARQL queries containing the OPT operator [Arenas and Pérez 2011]. Moreover, subsumption has also been used in the past as a meaningful way of testing containment of queries with incomplete answers over semistructured data [Kanza et al. 2002]. For QWDPTs, subsumption can also be used to test equivalence. However, it is not advisable to do so since we prove that

subsumption is presumably harder than equivalence by showing the Π_2^P-completeness of subsumption for QWDPTs. Interestingly, the complexity of subsumption remains unchanged even if we allow projection.

Finally, we study the relationship between tractable fragments of CQ answering and tractable fragments of well-designed SPARQL queries containing the OPT operator. For the classical evaluation problem mentioned before, results on tractable fragments of CQs smoothly carry over to well-designed SPARQL graph patterns. Indeed, the evaluation problem of well-designed SPARQL graph patterns which was shown to be coNP-complete in Pérez et al. [2009] becomes tractable if all its conjunctive parts (i.e., subexpressions built by the AND-operator only) are from tractable fragments of CQs. If projection is allowed, the evaluation problem of well-designed SPARQL will be shown to be Σ_2^P. The complexity drops to NP-completeness in case of a restriction to tractable fragments of CQs.

The analysis becomes more intricate when we study the enumeration problem (that is, actually computing the set of solutions) of well-designed SPARQL graph patterns. Our main result in this respect states that, for a SPARQL query in which all its conjunctive parts belong to a class of conjunctive queries that admit enumeration of all solutions with polynomial delay, also the the enumeration of solutions of the SPARQL query is feasible with *polynomial delay*. When projection comes into play, the picture changes: the enumeration of solutions is no longer feasible with polynomial delay even if the conjunctive parts belong to tractable CQ classes.

Structure of the Article. The rest of the article is organized as follows. In Section 2, we recall some basic notions and results. In particular, we formally introduce RDF and SPARQL. The crucial data structure of *quasi well-designed* pattern trees is introduced in Section 3 together with a SPARQL evaluation method based on this data structure and some equivalence-preserving transformation rules. The evaluation problem and the enumeration problem are studied in Section 4. The subsumption and equivalence problems are studied in Section 5. In Section 6, we extend the fundamental notions and results from the previous sections to well-designed SPARQL with projection. A conclusion and an outlook to future work are given in Section 7.

Several proof details in this article are relegated to the electronic appendix.

Related Work. As we have described, our work is heavily based on the formalization of SPARQL presented in Pérez et al. [2006a, 2009] and in particular on the notion of well-designed SPARQL patterns introduced in these papers. In Pérez et al. [2009] the authors also study the complexity of query evaluation. Schmidt et al. [2010] considered several aspects of SPARQL query optimization focused on rewriting queries based on properties of operators [Schmidt et al. 2010]. Neither of these works [Pérez et al. 2009; Schmidt et al. 2010] considered the complexity of equivalence and containment, nor the search for tractable fragments for query evaluation and enumeration which are the main problems touched in this article. The OPT operator in SPARQL resembles a left-outer join in SQL. Compared with the huge amount of research on static analysis of CQs, fragments of SQL containing left-outer join have almost been disregarded with respect to these problems with Larson and Zhou [2005] being one notable exception. Nevertheless, to the best of our knowledge, research on fundamental questions such as the complexity of query equivalence and tractable fragments for query evaluation has not been carried out to date for queries containing left-outer joins.

Outside the SPARQL context, Kanza et al. [2002] studied containment and equivalence for queries over a general semistructured data model. The query language considered in Kanza et al. [2002] allows for partial answers, nevertheless, as opposed to SPARQL, it does not allow one to explicitly state optional parts in a query, and partial

answers are generated by considering a different semantics for query evaluation. This makes our approach to partial answers, equivalence, and containment orthogonal to Kanza et al. [2002]. Gutierrez et al. [2011] studied similar problems for an abstract RDF query language. The difficulties in Gutierrez et al. [2011] arise from considering *blank nodes* and RDFS (features that we do not consider here), but they only consider conjunctive queries without optional parts, thus making their approach also orthogonal to ours. Cohen et al. [2006] define a polynomial delay iterator for computing full disjunctions. Full disjunctions are designed to obtain partial answers from relational sources but, in contrast to the OPT operator in SPARQL, full disjunctions are associative and commutative. Thus, the source of difficulties in devising an enumerator for full disjunctions departs from the difficulties that one encounters when enumerating SPARQL queries.

This article is a substantially extended version of Letelier et al. [2012b]. Besides containing the complete proofs of all the results stated in Letelier et al. [2012b], this version includes several new results and examples. In particular, all the results regarding projection over QWDPTs are new. In Section 6 we formalize projection over QWDPTs and provide transformation rules. These results are not presented in Letelier et al. [2012b]. Moreover, the results in Section 6.3 regarding the evaluation of QWDPTs with projection and Section 6.4 regarding the enumeration of solutions for QWDPTs with projection are all new.

2. BASIC DEFINITIONS, NOTATIONS, AND RESULTS

2.1. Basics of RDF and SPARQL

In this article we focus on ground RDF graphs, that is, RDF graphs that do not contain blank nodes.[1] Moreover, we do not make an explicit distinction between URIs (Uniform Resource Identifiers) and literals when defining RDF graphs, and thus we assume that RDF graphs are composed only of URIs. Hence, let $\mathbf{U}$ be an infinite set of URIs. An RDF triple is a tuple in $\mathbf{U} \times \mathbf{U} \times \mathbf{U}$, and an RDF graph (graph for short) is a finite set of RDF triples. The active domain of an RDF graph G, denoted by $\mathrm{dom}(G)$ with $\mathrm{dom}(G) \subseteq \mathbf{U}$, is the set of URIs actually appearing in G.

SPARQL [Prud'hommeaux and Seaborne 2008] is the standard query language for RDF. We next formalize its *graph pattern matching facility* which forms the core of the language. Assume the existence of an infinite set $\mathbf{V}$ of variables (disjoint from $\mathbf{U}$). We denote variables in $\mathbf{V}$ by using a question mark, as with $?X$. Then a SPARQL triple pattern is a tuple $t \in (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V})$. Complex graph patterns are constructed from triple patterns by using operators AND, OPT, UNION, and FILTER. In this article we focus on the SPARQL fragment composed of the operators AND and OPT. Formally, SPARQL graph patterns are recursively defined as follows:

(1) a triple pattern t is a graph pattern, and
(2) if P_1 and P_2 are graph patterns, then $(P_1 \text{ AND } P_2)$ and $(P_1 \text{ OPT } P_2)$ are graph patterns.

For a triple pattern t, we write $\mathrm{vars}(t)$ to denote the set of variables occurring in t, and for a graph pattern P we write $\mathrm{vars}(P)$ for the set of variables that occur in the triples that compose P.

[1]There is still no clear consensus on the way in which blank nodes should be treated when querying RDF [Mallea et al. 2011]. On the one hand, the SPARQL specification implies that blank nodes should be treated just as constants, and on the other hand the RDF specification defines blank nodes as existentially quantified variables. Thus, as in previous theoretical works about SPARQL [Pérez et al. 2009; Schmidt et al. 2010], we prefer to consider only ground RDF graphs.

To define the semantics of SPARQL graph patterns, we follow closely the definitions proposed in Pérez et al. [2009]. A mapping μ is a partial function $\mu : \mathbf{V} \to \mathbf{U}$. The domain of μ, denoted by $\text{dom}(\mu)$, is the set of all variables from $\mathbf{V}$ for which μ is defined. Given a triple pattern t and a mapping μ such that $\text{vars}(t) \subseteq \text{dom}(\mu)$, we denote by $\mu(t)$ the RDF triple obtained by replacing the variables in t according to μ. Given two mappings μ_1 and μ_2, we say that μ_1 and μ_2 are *compatible*, denoted by $\mu_1 \sim \mu_2$, if for every $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ it holds that $\mu_1(?X) = \mu_2(?X)$. Notice that, for compatible mappings μ_1 and μ_2, we have that $\mu_1 \cup \mu_2$ is also a mapping and is such that $(\mu_1 \cup \mu_2)(?X)$ is $\mu_1(?X)$ if $?X \in \text{dom}(\mu_1)$, or $\mu_2(?X)$ otherwise. Also notice that the mapping with empty domain, denoted by $\mu_\emptyset$, is compatible with any mapping. Before defining the semantics of SPARQL graph patterns, we define some operations between sets of mappings that resemble relational operators over sets of tuples. Let M_1 and M_2 be sets of mappings. We define the *join* and the *left-outer join* between M_1 and M_2 as follows.

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2 \text{ and } \mu_1 \sim \mu_2\}$$
$$M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup \{\mu \in M_1 \mid \forall \mu' \in M_2 : \mu \not\sim \mu'\}$$

We now have all the necessary prerequisites to formalize the evaluation of a SPARQL graph pattern over an RDF graph G as a function $[\![\cdot]\!]_G$ that, given a pattern, returns a set of mappings. Formally, $[\![P]\!]_G$ is defined recursively as follows [Pérez et al. 2009].

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

We say that two patterns P_1 and P_2 are equivalent, denoted by $P_1 \equiv P_2$, if for every RDF graph G, it holds that $[\![P_1]\!]_G = [\![P_2]\!]_G$. Notice that mappings explicitly refer to the variable names. Hence, unlike for Conjunctive Queries (CQs), the actual names of the variables matter, since two graph patterns containing different sets of variables can never be equivalent. In Pérez et al. [2009] the authors show several algebraic properties for graph patterns. In particular they show that AND is commutative and associative which allows us to drop parentheses from sequences of AND-operators.

Note that we described the set-semantics of SPARQL, while the W3C Recommendation defines a bag-semantics for query answering [Prud'hommeaux and Seaborne 2008]. Nevertheless, for the fragment described earlier and considered in the first part of this article (allowing only AND and OPT) both semantics coincide [Pérez et al. 2006b]. Only for the second part of the article where we consider the effect of allowing projection, the two semantics differ. However, in this article we only concentrate on the set-semantics.

Example 2.1 [*Pérez et al. 2009*]. Consider an RDF graph G storing information about professors in a university with the following triples, and the pattern P_1:

$(R_1, \text{name}, \text{paul}), (R_1, \text{phone}, \text{777-3426}),$
$(R_2, \text{name}, \text{john}), (R_2, \text{email}, \text{john@acd.edu}),$
$(R_3, \text{name}, \text{george}), (R_3, \text{webPage}, \text{www.george.edu}),$
$(R_4, \text{name}, \text{ringo}), (R_4, \text{email}, \text{ringo@acd.edu}),$
$(R_4, \text{webPage}, \text{www.starr.edu}), (R_4, \text{phone}, \text{888-4537})$

$$P_1 = (((?A, \text{name}, ?N) \text{ OPT } (?A, \text{email}, ?E)) \text{ OPT } (?A, \text{webPage}, ?W)).$$

If we evaluate P_1 over G, then intuitively we are retrieving the name of the resources in G and, optionally, for the resources that have an email we retrieve the email, and,

optionally, for the resources that have a Web page we retrieve the Web page. When evaluating P_1 over G we obtain the set of mappings $[\![P_1]\!]_G = \{\mu_1, \mu_2, \mu_3, \mu_4\}$ where

$$\mu_1 = \{?A \to R_1, ?N \to \text{paul}\},$$
$$\mu_2 = \{?A \to R_2, ?N \to \text{john}, ?E \to \text{john@acd.edu}\},$$
$$\mu_3 = \{?A \to R_3, ?N \to \text{george}, ?W \to \text{www.george.edu}\},$$
$$\mu_4 = \{?A \to R_4, ?N \to \text{ringo}, ?E \to \text{ringo@acd.edu}, ?W \to \text{www.starr.edu}\}.$$

Also, consider now pattern P_2 given by the following expression.

$$P_2 = ((?A, \text{name}, ?N) \text{ OPT } ((?A, \text{email}, ?E) \text{ OPT } (?A, \text{webPage}, ?W)))$$

In this case the evaluation of P_2 over G is the set of mappings $[\![P_2]\!]_G = \{\mu_1, \mu_2, \mu_3, \mu_4\}$ where

$$\mu_1 = \{?A \to R_1, ?N \to \text{paul}\},$$
$$\mu_2 = \{?A \to R_2, ?N \to \text{john}, ?E \to \text{john@acd.edu}\},$$
$$\mu_3 = \{?A \to R_3, ?N \to \text{george}\},$$
$$\mu_4 = \{?A \to R_4, ?N \to \text{ringo}, ?E \to \text{ringo@acd.edu}, ?W \to \text{www.starr.edu}\}.$$

Notice that we obtain no information for the Web page of george, since in P_2 that information is retrieved only for the resources that have an email (and george does not have an email address in G).

Well-designed graph patterns. An important class of SPARQL graph patterns identified in Pérez et al. [2009] that also plays a central role in this article is the class of *well-designed* graph patterns. A pattern P is well designed if for every subpattern $P' = (P_1 \text{ OPT } P_2)$ of P and every variable $?X$ occurring in P, it holds that

$$\text{if } ?X \text{ occurs inside } P_2 \text{ and outside } P', \text{ then } ?X \text{ also occurs inside } P_1.$$

Notice that patterns P_1 and P_2 in Example 2.1 are well designed. In Pérez et al. [2009] the authors studied several properties of well-designed patterns. Among others, they showed that the complexity of the evaluation problem is lower for well-designed patterns compared with the general language: they showed that for well-designed SPARQL, the following problem is coNP-complete, while it is complete for PSPACE when allowing arbitrary SPARQL graph patterns.

Definition 2.2. Let EVALUATION be the following problem.
INPUT: An RDF graph G, a SPARQL graph pattern P, and a mapping μ.
QUESTION: Is $\mu \in [\![P]\!]_G$?

Moreover, Perez et al. suggested that well-designed patterns are suitable for optimization procedures, proposing a set of rewriting rules. In this article we go an important step further in this direction by proposing a tree representation for well-designed SPARQL (so-called pattern trees) and equivalence-preserving transformation rules.

2.2. Graphs, Trees, and Graph Colorings

Let $G = (V, E)$ be an undirected graph, where V is the set of nodes or vertices of G and E is the set of undirected edges. We write (v_i, v_j) for the edges $e \in E$ connecting node v_i and v_j. Note that the order in (v_i, v_j) does not matter, that is, for our purposes $(v_i, v_j) = (v_j, v_i)$. For some undirected graph G we may also write $V(G)$ to denote the set of nodes and $E(G)$ to denote the set of edges. For $N \subseteq V$, we write $G[N]$ to denote the subgraph of V induced by N, that is, the graph $G' = (N, E')$ where $E' = \{e \in E \mid e = (v_i, v_j) \text{ and } v_i, v_j \in N\}$.

A tree T is a connected, acyclic graph. As usual, let a *rooted tree* be defined as a tuple $T = (V, E, r)$, where $r \in V$ is the designated root of T. We define the notions of parent, ancestor, child, and descendant of a node $t \in V$ as usual. We further assume trees to be undirected and unordered.

We assume the reader to be familiar with the well-known NP-complete problem 3COL—the 3-colorability problem of graphs.

3. PATTERN TREES

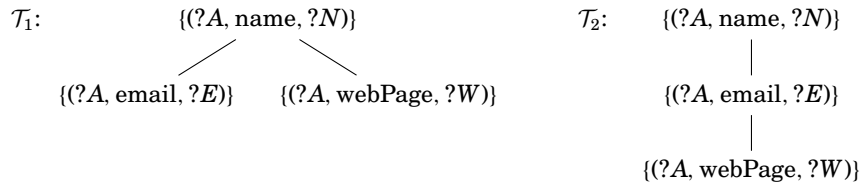
In this section we propose a novel representation of SPARQL graph patterns based on trees, together with an evaluation method for these trees. This tree representation of patterns plays a central role when we study optimization, query equivalence, and containment, and also tractable fragments of SPARQL queries. The basic concept of this representation is a *pattern tree*.

Definition 3.1 (*Pattern Tree*). A *pattern tree* $\mathcal{T}$ is a pair $\mathcal{T} = (T, \mathcal{P})$, where $T = (V, E, r)$ is a rooted tree, and $\mathcal{P} = \left(P_n\right)_{n \in V}$ is a labeling of the nodes of T such that P_n is a nonempty set of triple patterns, for every $n \in V$.

Given a pattern tree $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ and a node $n \in V$, a *subtree of* $\mathcal{T}$ *rooted at* n is a pattern tree composed of n and a connected subset of its descendants. Moreover, *the complete subtree of* $\mathcal{T}$ *rooted at* n, that we usually denote by $\mathcal{T}_n$, is the pattern tree composed of n and *all* its descendants. We further denote by $\text{vars}(P_n)$ the set of variables that occur in the triples of P_n, and by $\text{vars}(\mathcal{T})$ the set $\bigcup_{n \in V} \text{vars}(P_n)$. Also, we may use $V(\mathcal{T})$ to denote the set V of vertices of $\mathcal{T}$ in order to avoid providing an explicit name for it. In the following, for pattern trees, usually the tree structure is depicted with the corresponding labels in every node

The idea of using pattern trees to describe well-designed SPARQL patterns is that the tree structure represents the structure of the nested OPT-operators of the pattern, while the conjunctions of triple patterns are contained at the nodes.

Example 3.2. The following are pattern trees that intuitively correspond to the queries introduced in Example 2.1.



Next, we give a meaning to pattern trees by transforming them into SPARQL graph patterns. Towards this goal, we need the following definition of a transformation function $\text{Tr}(\cdot, \cdot, \cdot)$. Consider a pattern tree $\mathcal{T} = ((V, E, r), \mathcal{P})$ and a set Σ of functions $\{\sigma_n \mid n \in V\}$ such that for every $n \in V$, function σ_n defines an ordering on the children of n (that is, $\sigma_n(1)$ is the first child in the order, $\sigma_n(2)$ is the second one, and so on). As last definition before presenting the transformation, given a set $P = \{t_1, \ldots, t_\ell\}$ of triple patterns, we denote by $and(P)$ the graph pattern $(t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_\ell)$. For $\mathcal{T}$ and $n \in V$, we are now ready to define the transformation $\text{Tr}(\mathcal{T}, n, \Sigma)$ of $\mathcal{T}_n$, the complete subtree of $\mathcal{T}$ rooted at n, given the order Σ. Assume that n has k children in

$\mathcal{T}$, then $\text{TR}(\mathcal{T}, n, \Sigma)$ is defined as the graph pattern expression

$$(\cdots((and(P_n) \text{ OPT } \text{TR}(\mathcal{T}, \sigma_n(1), \Sigma)) \text{ OPT } \text{TR}(\mathcal{T}, \sigma_n(2), \Sigma)) \cdots \text{ OPT } \text{TR}(\mathcal{T}, \sigma_n(k), \Sigma)),$$

and if n has no children, then $\text{TR}(\mathcal{T}, n, \Sigma) = and(P_n)$. Finally, given a pattern tree $\mathcal{T} = ((V, E, r), \mathcal{P})$ and an ordering Σ for $\mathcal{T}$, we define $\text{TR}(\mathcal{T}, \Sigma)$ as $\text{TR}(\mathcal{T}, r, \Sigma)$.

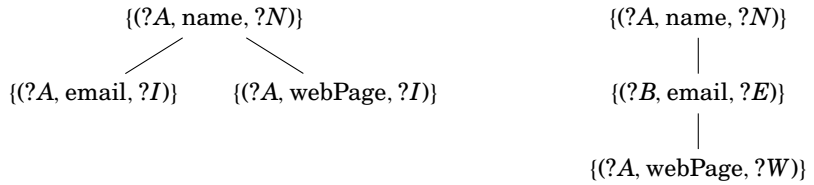
Example 3.3. Consider the tree $\mathcal{T}_1$ in Example 3.2, and let Σ be the order induced by the picture in the example. Then $\text{TR}(\mathcal{T}_1, \Sigma)$ is pattern P_1 in Example 2.1.

3.1. Semantics of Well-Designed Pattern Trees

We have established a syntactic relationship between pattern trees and SPARQL graph patterns. We now want to establish a semantic relationship between these representations. In particular, we are interested in defining the evaluation of a pattern tree over an RDF graph. Notice that several (different) SPARQL patterns can be obtained from a pattern tree depending on the ordering functions used in the transformation. Thus, we cannot directly define the evaluation of a pattern tree $\mathcal{T}$ by using the evaluation of an arbitrary transformation of $\mathcal{T}$. In this section we introduce a well-designedness condition for pattern trees that will be crucial in defining a semantics for pattern trees. In particular it will allow us to choose an arbitrary transformation of a pattern tree in order to evaluate it. We begin with the definition of the well-designedness condition for pattern trees.

Definition 3.4. A pattern tree $\mathcal{T} = ((V, E, r), \mathcal{P})$ is *well designed* if for every variable $?X$ occurring in $\mathcal{T}$, the set $\{n \in V \mid ?X \in \text{vars}(P_n)\}$ induces a connected subgraph of T.

Example 3.5. The pattern trees in Example 3.2 are well designed, while the following pattern trees are not.



Variable $?I$ in the tree on the left, and variable $?A$ in the tree on the right, induce disconnected subgraphs.

As expected, this well-designedness condition over trees is tightly connected to the well-designedness condition for graph patterns. In particular, the following holds.

PROPOSITION 3.6. *Let $\mathcal{T}$ be a well-designed pattern tree, and Σ an arbitrary set of ordering functions for $\mathcal{T}$. Then $\text{TR}(\mathcal{T}, \Sigma)$ is a well-designed graph pattern.*

A proof of this proposition is provided in the electronic appendix. Although it is possible to give a semantics directly for well-designed pattern trees, we first introduce a relaxation of this notion that we call *quasi well-designed pattern trees* that plays a fundamental role in our study, and give a semantics for this class of pattern trees. With this relaxed notion we will also be able (see Section 3.3) to get rid of some redundancy in pattern trees. In particular, we will be able to eliminate unnecessary repeated triples in graph patterns whose deletion may violate the well-designed condition.

Definition 3.7. A pattern tree $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ is a *quasi well-designed pattern tree* (*QWDPT* for short) if for every pair of nodes $u, v \in V$ and each variable $?X \in \text{vars}(P_u) \cap \text{vars}(P_v)$ there exists a node n that is a common ancestor of u and v in $\mathcal{T}$, such that $?X \in \text{vars}(P_n)$.

The pattern tree on the right in Example 3.5 is a QWDPT, while the pattern on the left is not (notice that the common ancestor of u and v in Definition 3.7 may be u or v, as in Example 3.5). Another notion that we need to introduce is that of *duplicating triples to children*. Formally, we say that a pattern tree $\mathcal{T}' = ((V', E', r'), (P'_n)_{n \in V'})$ was derived from a pattern tree $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ by duplicating a triple to a child, denoted by $\mathcal{T} \hookrightarrow \mathcal{T}'$, if $(V', E', r') = (V, E, r)$ (that is, the underlying trees are the same), and there exist a node $u \in V$, a triple $t \in P_u$, and a child v of u, such that $P'_v = P_v \cup \{t\}$, and $P_n = P'_n$ for all $n \neq v$. We denote by $\hookrightarrow^*$ the reflexive and transitive closure of $\hookrightarrow$, that is, $\mathcal{T} \hookrightarrow^* \mathcal{T}'$ if $\mathcal{T} = \mathcal{T}'$ or there exists a sequence $\mathcal{T}_1 \hookrightarrow \mathcal{T}_2 \hookrightarrow \cdots \hookrightarrow \mathcal{T}_m$ with $\mathcal{T}_1 = \mathcal{T}$ and $\mathcal{T}_m = \mathcal{T}'$. It is easy to observe that every QWDPT can be converted into a well-designed pattern tree by duplicating triples along branches. Formally, for every QWDPT $\mathcal{T}$, there exists a well-designed pattern tree $\mathcal{T}'$ such that $\mathcal{T} \hookrightarrow^* \mathcal{T}'$. It is also easy to observe that the (quasi) well-designed property is invariant under $\hookrightarrow^*$. We now have all the necessary ingredients to define a semantics of pattern trees. Towards this goal, we first identify a set of SPARQL graph patterns with a QWDPT.

Definition 3.8. The set of SPARQL graph patterns defined by a QWDPT $\mathcal{T}$ is

$$\text{SEM}(\mathcal{T}) = \{\text{TR}(\mathcal{T}', \Sigma) \mid \Sigma \text{ is an ordering for } \mathcal{T}', \mathcal{T} \hookrightarrow^* \mathcal{T}' \text{ and } \mathcal{T}' \text{ is well designed}\}.$$

We would like to define the result of evaluating a QWDPT $\mathcal{T}$ over an RDF graph G to be exactly the same as that of an arbitrarily chosen query from $\text{SEM}(\mathcal{T})$. However, in order to do so we first have to show that all queries in $\text{SEM}(\mathcal{T})$ are equivalent. The following two results are thus the basis for our definition. The first one shows that the ordering of the child nodes in a well-designed pattern tree does not influence its semantics, but that for any two orderings the results of the transformation to SPARQL graph patterns are equivalent. Similarly, the second result shows that the semantics is also independent of the concrete sequence $\mathcal{T} \hookrightarrow^* \mathcal{T}'$: all well-designed pattern trees that can be derived from $\mathcal{T}$ by duplicating triples to children are equivalent.

LEMMA 3.9. *Let $\mathcal{T}$ be a well-designed pattern tree, let Σ_1, Σ_2 be two arbitrary orderings for $\mathcal{T}$, and let $P_1 = \text{TR}(\mathcal{T}, \Sigma_1)$ and $P_2 = \text{TR}(\mathcal{T}, \Sigma_2)$ be the graph patterns obtained by transforming $\mathcal{T}$ with Σ_1 and Σ_2, respectively. Then $P_1 \equiv P_2$.*

LEMMA 3.10. *Let $\mathcal{T}$ be a QWDPT, let Σ be an ordering for $\mathcal{T}$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be well-designed pattern trees such that $\mathcal{T} \hookrightarrow^* \mathcal{T}_1$ and $\mathcal{T} \hookrightarrow^* \mathcal{T}_2$. If $P_1 = \text{TR}(\mathcal{T}_1, \Sigma)$ and $P_2 = \text{TR}(\mathcal{T}_2, \Sigma)$, then $P_1 \equiv P_2$.*

Full proofs of both lemmas are provided in the electronic appendix. The following theorem follows directly from Lemmas 3.9 and 3.10.

THEOREM 3.11. *Let $\mathcal{T}$ be a QWDPT. Then all graph patterns in $\text{SEM}(\mathcal{T})$ are equivalent, that is, for any two graph patterns $P_1, P_2 \in \text{SEM}(\mathcal{T})$, it holds that $P_1 \equiv P_2$.*

This finally allows us to define the semantics of a QWDPT via an arbitrary graph pattern from $\text{SEM}(\mathcal{T})$.

Definition 3.12. Let $\mathcal{T}$ be a QWDPT and G an RDF graph. Then the evaluation of $\mathcal{T}$ over G, denoted by $[\![\mathcal{T}]\!]_G$, is defined as the set of mappings $[\![P]\!]_G$ for an arbitrary $P \in \textsc{Sem}(\mathcal{T})$.

By Theorem 3.11, the semantics of a QWDPT according to Definition 3.12 is well defined. This means that, for a QWDPT $\mathcal{T}$, we may choose any representative from $\textsc{Sem}(\mathcal{T})$ for evaluation.

Given two QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are equivalent, denoted by $\mathcal{T}_1 \equiv \mathcal{T}_2$, if for every RDF graph G it holds that $[\![\mathcal{T}_1]\!]_G = [\![\mathcal{T}_2]\!]_G$. Similarly, a QWDPT $\mathcal{T}$ is equivalent to a SPARQL graph pattern P, denoted by $\mathcal{T} \equiv P$, if for every RDF graph G it holds that $[\![\mathcal{T}]\!]_G = [\![P]\!]_G$. Notice that Definition 3.12 plus Proposition 3.6 imply that for every QWDPT $\mathcal{T}$ there exists a well-designed graph pattern P such that $\mathcal{T} \equiv P$. The last result of this section states that the opposite also holds, and thus, QWDPTs can represent the entire class of well-designed SPARQL graph patterns.

PROPOSITION 3.13. *For every well-designed graph pattern P, there exists a QWDPT $\mathcal{T}$ such that $P \equiv \mathcal{T}$. Moreover, given a well-designed graph pattern, an equivalent QWDPT can be constructed in polynomial time.*

PROOF. In Pérez et al. [2009] it was shown that every well-designed graph pattern is equivalent to a pattern in OPT-normal form which is defined as follows.

(1) A pattern of the form $(t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_k)$ with t_i triple patterns, is in OPT-normal form.
(2) If P_1 and P_2 are in OPT-normal form then $(P_1 \text{ OPT } P_2)$ is in OPT-normal form.

Given a pattern P in OPT-normal form we describe an algorithm to construct a well-designed pattern tree. If $P = (t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_k)$ then we create a pattern tree with a single node and label $\{t_1, \ldots, t_k\}$. Now, if $P = (P_1 \text{ OPT } P_2)$ then we construct a pattern tree $\mathcal{T}_1$ from P_1, a pattern tree $\mathcal{T}_2$ from P_2, and then construct a pattern tree $\mathcal{T}$ from $\mathcal{T}_1$ and $\mathcal{T}_2$, by considering $\mathcal{T}_1$ and $\mathcal{T}_2$ together, adding the root of $\mathcal{T}_2$ as a child of $\mathcal{T}_1$, and setting the root of $\mathcal{T}_1$ as the root of the obtained tree $\mathcal{T}$. It is not difficult to show that the obtained pattern tree $\mathcal{T}$ is well designed and that there exists an ordering Σ for $\mathcal{T}$ such that $\textsc{Tr}(\mathcal{T}, \Sigma) = P$ and thus, $\mathcal{T} \equiv P$. □

3.2. Evaluating Pattern Trees

In this section we introduce an evaluation method for QWDPTs that takes advantage of our tree representation. Beside providing an intuitive meaning to QWDPTs (and therefore also to the well-designed SPARQL graph patterns represented by a pattern tree), this semantics also shows a possible way for evaluating the pattern tree. Thus QWDPTs can act as execution plans for well-designed SPARQL patterns. In Pérez et al. [2006a] the authors proposed a top-down evaluation method for SPARQL graph patterns and they showed that for well-designed patterns the top-down evaluation over a graph G is equivalent to the evaluation given by $[\![\cdot]\!]_G$. Our proposal is similar to the approach in Pérez et al. [2006a], but it is based on an alternative characterization of the evaluation of well-designed graph patterns proposed in Pérez et al. [2009]. We reformulate here this characterization for the case of pattern trees. It will later play an important role when we study transformations of pattern trees as well as containment and equivalence testing. We first introduce the necessary terminology.

We say that a mapping μ_1 is *subsumed by* μ_2, denoted by $\mu_1 \sqsubseteq \mu_2$, if $\text{dom}(\mu_1) \subseteq \text{dom}(\mu_2)$ and for every $?X \in \text{dom}(\mu_1)$ it holds that $\mu_1(?X) = \mu_2(?X)$ (implying that $\mu_1 \sim \mu_2$). We write $\mu_1 \sqsubset \mu_2$ whenever $\mu_1 \sqsubseteq \mu_2$ and $\mu_1 \neq \mu_2$. Furthermore, recall that given a set $P = \{t_1, \ldots, t_\ell\}$ of triple patterns, we denote by $and(P)$ the graph pattern $(t_1 \text{ AND } t_2 \text{ AND } \cdots \text{ AND } t_\ell)$. Now given a pattern tree $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$, we use

$and(\mathcal{T})$ to denote the SPARQL pattern constructed by taking the conjunction (AND) of all the triples that occur in $\mathcal{T}$. That is, if $V = \{n_1, \ldots, n_\ell\}$, then

$$and(\mathcal{T}) = \big(and(P_{n_1}) \text{ AND } \cdots \text{ AND } and(P_{n_\ell})\big).$$

We next characterize the evaluation of a QWDPT. It follows directly from the results in Pérez et al. [2009] for well-designed graph patterns, and the relationship with QWDPTs shown in the previous section.

LEMMA 3.14. *Let $\mathcal{T}$ be a QWDPT with root r, and let G be an RDF graph. A mapping μ is in $[\![\mathcal{T}]\!]_G$ if and only if:*

(1) $\mu \in [\![and(\mathcal{T}')]\!]_G$ for a subtree $\mathcal{T}'$ of $\mathcal{T}$ rooted at r, and
(2) for each mapping ν and subtree $\mathcal{T}''$ of $\mathcal{T}$ rooted at r, if $\nu \in [\![and(\mathcal{T}'')]\!]_G$ then $\mu \not\sqsubseteq \nu$.

PROOF. This lemma follows from the characterization of the evaluation of well-designed graph patterns proposed in Pérez et al. [2009], in particular from Proposition 4.5 in Pérez et al. [2009], and the relationship between well-designed graph patterns and quasi well-designed pattern trees. The argument is as follows. It was proved in Pérez et al. [2009] that given a well-designed graph pattern P and an RDF graph G,

$$\text{a mapping } \mu \text{ is in } [\![P]\!]_G \text{ if and only if } \mu \text{ is a } \textit{maximal (with respect to } \sqsubseteq\textit{) partial solution of } P. \tag{1}$$

The notion of *partial solution* mentioned in (1) coincides with the following notion in the case of pattern trees. Given a pattern tree $\mathcal{T}$ rooted at r, a mapping μ is a partial solution for $\mathcal{T}$ over G if there exists a subtree $\mathcal{T}'$ of $\mathcal{T}$ rooted at r, such that $\mu \in [\![and(\mathcal{T}')]\!]$. Thus, given the relationship between well-designed pattern and quasi well-designed pattern trees, and the definition of the semantics of a pattern tree $\mathcal{T}$ (which states that we can evaluate an arbitrary pattern in $\text{SEM}(\mathcal{T}, \Sigma)$), we obtain that (1) can be reformulated in our case as follows. Given a quasi well-designed pattern tree $\mathcal{T}$ and an RDF graph G,

$$\text{a mapping } \mu \text{ is in } [\![\mathcal{T}]\!]_G \text{ if and only if } \mu \text{ is a maximal (with respect to } \sqsubseteq\text{) partial solution of } \mathcal{T}. \tag{2}$$

Finally, notice that property (1) in Lemma 3.14 essentially states that mapping μ is a partial solution for $\mathcal{T}$ over G, and property (2) in Lemma 3.14 states that μ is maximal. Thus we have that the characterization in Lemma 3.14 coincides with Property (2) which shows that the lemma holds. □

Lemma 3.14 essentially states that the mappings in the evaluation of a QWDPT over some graph G are exactly those that map all triples in some subtree $\mathcal{T}'$ of $\mathcal{T}$ (hence $and(\mathcal{T}')$) into G, and that cannot be further extended by considering another subtree $\mathcal{T}''$ of $\mathcal{T}$. This characterization inspires the following procedural semantics that is obtained by evaluating the pattern tree by a top-down traversal. For simplicity, given a label P_n of node n and a graph G, we denote by $[\![P_n]\!]_G$ the set $[\![and(P_n)]\!]_G$.

Definition 3.15. Consider an RDF graph G, a QWDPT $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$, and a set M of mappings. For $n \in V$, we define the evaluation of $\mathcal{T}_n$ (the complete subtree of $\mathcal{T}$ rooted at n) given M over G, denoted by $\text{ext}(M, n, G)$, as follows. If n is a leaf, then

$$\text{ext}(M, n, G) = M \bowtie [\![P_n]\!]_G,$$

and, otherwise, if $n_1, \ldots, n_k$ are the child nodes of n, then

$$\text{ext}(M, n, G) = M_1 \bowtie M_2 \bowtie \cdots \bowtie M_k,$$

where $M_i = (M \bowtie [\![P_n]\!]_G) \rtimes \text{ext}(M \bowtie [\![P_n]\!]_G, n_i, G)$. We define the *top-down evaluation* of $\mathcal{T}$ over G, denoted by $[\![\mathcal{T}]\!]_G^{td}$, as

$$[\![\mathcal{T}]\!]_G^{td} = \text{ext}(\{\mu_\emptyset\}, r, G),$$

where $\mu_\emptyset$ is the mapping with the empty domain.

The preceding definition can be also seen in a more procedural way: Given some QWDPT with root r and some RDF graph G, first get the set M of all mappings that map P_r into G. For each mapping $\mu \in M$ property (1) of Lemma 3.14 is satisfied. Now in order to test property (2), it suffices to check for each such mapping μ if it can be extended to some child n of r, that is, to some mapping $\mu' \colon \text{vars}(P_r) \cup \text{vars}(P_n) \to \text{dom}(G)$ compatible with μ such that $\mu'(P_n) \subseteq G$. If this is possible, replace μ by μ'. Note that μ' again satisfies property (1) of Lemma 3.14. Hence one way to think of this evaluation method is to maintain a set of partial solutions together with a subtree $\mathcal{T}'$ of the input QWDPT rooted at r for each of them. In order to determine whether the mapping can be extended, it suffices to check if it can be extended to a child node of the leaf nodes of $\mathcal{T}'$.

The following theorem shows that the top-down evaluation defined previously coincides with the semantics of pattern trees introduced in the previous section.

THEOREM 3.16. *Let $\mathcal{T}$ be a QWDPT and G an RDF graph. Then $[\![\mathcal{T}]\!]_G = [\![\mathcal{T}]\!]_G^{td}$.*

PROOF. We first introduce some notation. Let $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ be a quasi well-designed pattern tree, $n \in V$. We denote by $\mathcal{T}_n$ the subtree of $\mathcal{T}$ rooted at node n. From now on in this proof, whenever we say that a pattern tree $\mathcal{T}_1$ is a subtree of a pattern tree $\mathcal{T}_2$ we assume that both trees coincide in their root node and that all the labels in $\mathcal{T}_1$ are the same as the labels of $\mathcal{T}_2$ (for the nodes that are composing the subtree $\mathcal{T}_1$).

Now let G be an RDF graph. We next show that for every node $n \in V$ the following property holds. Let M be a set of mappings all of them with the same domain, and assume that there is a set of triple patterns P_M such that $M = [\![P_M]\!]_G$ Moreover, assume that if $?X$ is a variable that occurs in two different descendants of n but not in P_n, then $?X \in \text{dom}(\mu)$ for every $\mu \in M$ (and thus $?X$ occurs in P_M). Finally, we denote by $\mathcal{T}_n^{P_M}$ the pattern tree obtained from $\mathcal{T}_n$ by adding all triples in P_M to the label of the root of $\mathcal{T}_n$ (that is, the new label of the root is $P_n \cup P_M$). We claim that

$$\text{ext}(M, n, G) \;=\; [\![\mathcal{T}_n^{P_M}]\!]_G.$$

We show this by induction on the tree $\mathcal{T}_n$. If n is a leaf node, then $\mathcal{T}_n$ is composed of a single node labeled P_n, and then $\text{ext}(M, n, G) = M \bowtie [\![P_n]\!]_G$. On the other hand $[\![\mathcal{T}_n^{P_M}]\!] = [\![P_n \cup P_M]\!]_G = [\![P_M]\!]_G \bowtie [\![P_n]\!]_G = M \bowtie [\![P_n]\!]_G$, and then the property holds. Assume now that n has $n_1, \ldots, n_k$ as children. Then in this case we have that

$$\begin{aligned}\text{ext}(M, n, G) \;=\; &\big((M \bowtie [\![P_n]\!]_G) \rtimes \text{ext}(M \bowtie [\![P_n]\!]_G), n_1, G)\big) \\ &\bowtie \cdots \bowtie \big((M \bowtie [\![P_n]\!]_G) \rtimes \text{ext}(M \bowtie [\![P_n]\!]_G), n_k, G)\big).\end{aligned}$$

Let $M' = M \bowtie [\![P_n]\!]_G$. It is not difficult to see that for every n_i, if $?X$ occurs in two different descendants u and v of n_i but not in P_{n_i}, then $?X \in \text{dom}(\mu)$ for every $\mu \in M'$. This is because u and v are also descendants of n, and thus variable $?X$ is either in P_n or in $\text{dom}(\mu)$ for every $\mu \in M$. Moreover, we have that $M' = [\![P_M \cup P_n]\!]_G$. Thus we can apply the induction hypothesis, and then we have that $\text{ext}(M \bowtie [\![P_n]\!]_G, n_i, G) = [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$, and thus, we can write $\text{ext}(M, n, G)$ as

$$\text{ext}(M, n, G) \;=\; \big([\![P_M \cup P_n]\!]_G \rtimes [\![\mathcal{T}_{n_1}^{P_M \cup P_n}]\!]_G\big) \bowtie \cdots \bowtie \big([\![P_M \cup P_n]\!]_G \rtimes [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G\big).$$

In order to prove what we need, it is therefore enough to show that

$$[\![\mathcal{T}_n^{P_M}]\!]_G \;=\; \big([\![P_M \cup P_n]\!]_G \rtimes [\![\mathcal{T}_{n_1}^{P_M \cup P_n}]\!]_G\big) \bowtie \cdots \bowtie \big([\![P_M \cup P_n]\!]_G \rtimes [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G\big). \tag{3}$$

Before proving this we observe that, although $\mathcal{T}_n$ can be a pattern which is not quasi well designed, the properties of M ensure that $\mathcal{T}_n^{P_M}$ is quasi well designed. Similarly $\mathcal{T}_{n_i}^{P_M \cup P_n}$ is quasi well designed for every $i \in \{1, \ldots, k\}$. We now prove (3). Thus assume that $\mu \in [\![\mathcal{T}_n^{P_M}]\!]_G$. Then since $\mathcal{T}_n^{P_M}$ is quasi well designed, by Lemma 3.14 we know that there exists a subtree $\mathcal{T}'$ of $\mathcal{T}_n^{P_M}$ such that $\mu \in [\![and(\mathcal{T}')]\!]_G$, and μ is maximal, that is, there is no other subtree $\mathcal{T}''$ such that μ is strictly subsumed by a mapping in $[\![and(\mathcal{T}'')]\!]_G$. Consider a maximal such subtree $\mathcal{T}'$ for μ. First notice that $\mathcal{T}'$ is composed of the root of $\mathcal{T}_n^{P_M}$ plus (possibly empty) subtrees of the $\mathcal{T}_{n_i}$'s as children of the root of $\mathcal{T}_n^{P_M}$. Thus, assume that $\mathcal{T}'$ is composed of the root of $\mathcal{T}_n^{P_M}$ plus trees $\mathcal{T}_1', \mathcal{T}_2', \ldots, \mathcal{T}_k'$ as children, where every $\mathcal{T}_i'$ is either empty (in which case nothing is added as a child to the root of $\mathcal{T}_n^{P_M}$), or $\mathcal{T}_i'$ is a subtree of $\mathcal{T}_{n_i}$. Since $\mu \in [\![and(\mathcal{T}')]\!]_G$ and $\mathcal{T}'$ contains $P_n \cup P_M$ as root, we know that there exists a mapping μ' such that $\mu' \sqsubseteq \mu$ and $\mu' \in [\![P_n \cup P_M]\!]_G$. Notice that this mapping μ' is unique (and has as domain exactly the variables mentioned in $P_n \cup P_M$). We next prove some properties of the trees $\mathcal{T}_i'$ depending on whether they are empty or not.

—Given that $\mu \in [\![and(\mathcal{T}')]\!]_G$, we have that for every $i \in \{1, \ldots, k\}$, if $\mathcal{T}_i'$ is not empty then there exists a mapping μ_i' such that $\mu_i' \sqsubseteq \mu$ and $\mu_i' \in [\![and(\mathcal{T}_i')]\!]_G$. For every nonempty $\mathcal{T}_i'$ consider the pattern $(\mathcal{T}_i')^{P_M \cup P_n}$ constructed similarly as $\mathcal{T}_{n_i}^{P_M \cup P_n}$. Then by the construction of $\mathcal{T}'$, we know that there exists a mapping $\mu' \cup \mu_i' \in [\![and((\mathcal{T}_i')^{P_M \cup P_n})]\!]$, with μ' the portion of μ such that $\mu' \in [\![P_M \cup P_n]\!]_G$. Notice that $\mu' \cup \mu_i' \sqsubseteq \mu$. We claim that $\mu' \cup \mu_i' \in [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$. On the contrary, assume that $\mu' \cup \mu_i' \notin [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$. Since $\mu' \cup \mu_i' \in [\![and((\mathcal{T}_i')^{P_M \cup P_n})]\!]$, $(\mathcal{T}_i')^{P_M \cup P_n}$ is a subtree of $\mathcal{T}_{n_i}^{P_M \cup P_n}$, and $\mathcal{T}_{n_i}^{P_M \cup P_n}$ is quasi well designed, by Lemma 3.14 we know that there exists a subtree $\mathcal{T}_i''$ of $\mathcal{T}_{n_i}^{P_M \cup P_n}$ and a mapping ν_i such that $\mu' \cup \mu_i' \sqsubset \nu_i$ and $\nu_i \in [\![and(\mathcal{T}_i'')]\!]$. Since $\mu' \cup \mu_i' \sqsubset \nu_i$ we know that there exists a variable $?Y \in \mathrm{dom}(\nu_i)$ such that $?Y \notin \mathrm{dom}(\mu' \cup \mu_i')$. Moreover for every such variable $?Y$ that is in $\mathrm{dom}(\nu_i)$ but not in $\mathrm{dom}(\mu' \cup \mu_i')$, we have that $?Y$ does not occur in any other branch of $\mathcal{T}_n^{P_M}$ since $\mathcal{T}_n^{P_M}$ is quasi well designed. In particular, $?Y$ does not occur in any $\mathcal{T}_{n_j}$ for $j \neq i$, and then we have that $?Y \notin \mathrm{dom}(\mu)$. Moreover, since $\mu' \cup \mu_i' \sqsubseteq \mu$ and all the variables that are in $\mathrm{dom}(\nu_i)$ but not in $\mathrm{dom}(\mu' \cup \mu_i')$ are not in $\mathrm{dom}(\mu)$, we have that μ and ν_i are compatibles, and then $\mu \sqsubset \mu \cup \nu_i$. Furthermore, since $\nu_i \in [\![and(\mathcal{T}_i'')]\!]$, we have that there exists a subtree $\mathcal{T}''$ of $\mathcal{T}_n^{P_M}$ such that $\mu \sqsubset \mu \cup \nu_i \in [\![and(\mathcal{T}'')]\!]$. This is a contradiction with the maximality of μ.

—Now assume that $\mathcal{T}_i'$ is empty. We show next that there is no mapping ν_i compatible with μ such that $\nu_i \in [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$. On the contrary, assume that there is a mapping ν_i compatible with μ such that $\nu_i \in [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$. Given that $\mathcal{T}_{n_i}^{P_M \cup P_n}$ is quasi well designed from Lemma 3.14 we obtain that there exists a subtree $\mathcal{T}_i'$ of $\mathcal{T}_{n_i}^{P_M \cup P_n}$ such that $\nu_i \in [\![and(\mathcal{T}_i')]\!]_G$. Recall that $\mathcal{T}_{n_i}^{P_M \cup P_n}$ is constructed from $\mathcal{T}_{n_i}$ by adding $P_M \cup P_n$ to the label of the root. Thus, we have that $\nu_i = \nu_i' \cup \nu_i''$ such that $\nu_i' \in [\![P_M \cup P_n]\!]_G$ and $\nu_i'' \in [\![and(\mathcal{T}_i'')]\!]$ where $\mathcal{T}_i''$ is the tree obtained from $\mathcal{T}_i'$ deleting $P_M \cup P_n$ from its root. Then $\mathcal{T}_i''$ is a subtree of $\mathcal{T}_{n_i}$. Consider now the tree $\mathcal{T}''$ obtained from $\mathcal{T}'$ by adding $\mathcal{T}_i''$ as a child to the root of $\mathcal{T}'$. Then we have that $\mathcal{T}''$ is a subtree of $\mathcal{T}_n^{P_M}$ and that $\mu \cup \nu_i \in [\![and(\mathcal{T}'')]\!]_G$. Thus, if $\mu \cup \nu_i \neq \mu$ we obtain a contradiction with the maximality of μ, and if $\mu \cup \nu_i = \mu$ we obtain a contradiction with the maximality of $\mathcal{T}'$. Thus, we have shown that there is no mapping ν_i compatible with μ such that $\nu_i \in [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$.

Let μ' the portion of μ such that $\mu' \in [\![P_M \cup P_n]\!]_G$. Summarizing we have shown that:

—for every $\mathcal{T}_i'$ which is not empty, there exists a portion of μ, say μ_i' such that $\mu' \cup \mu_i' \in [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$, and thus $\mu' \cup \mu_i' \in \left([\![P_M \cup P_n]\!]_G \bowtie [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G\right)$, and

—for every $\mathcal{T}_i'$ which is empty, we have that μ is not compatible with any mapping in $[\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$, implying that μ' is not compatible with any mapping in $[\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$, and thus $\mu' \in [\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G$.

From this we obtain that μ can be written as $\mu = \mu_1 \cup \mu_2 \cup \cdots \cup \mu_k$ such that $\mu_i \in [\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G$, which implies that μ is in $\big([\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_1}^{P_M \cup P_n}]\!]_G\big) \bowtie \cdots \bowtie ([\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G)$.

For the opposite direction, if we assume that μ is in $([\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_1}^{P_M \cup P_n}]\!]_G) \bowtie \cdots \bowtie \big([\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_k}^{P_M \cup P_n}]\!]_G\big)$, then $\mu = \mu_1 \cup \cdots \mu_k$ with $\mu_i \in [\![P_M \cup P_n]\!]_G \not\bowtie [\![\mathcal{T}_{n_i}^{P_M \cup P_n}]\!]_G$. By using an argument similar to the one used in the previous case, and using the fact that every $\mathcal{T}_{n_i}^{P_M \cup P_n}$ is quasi well designed and Lemma 3.14, it is not difficult to conclude that μ is in $[\![\mathcal{T}_n^{P_M}]\!]_G$.

To conclude the proof of the theorem, just observe that for the pattern tree $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ the set $[\![\mathcal{T}]\!]_G^{\text{td}}$ is defined as $\text{ext}(\{\mu_\emptyset\}, r, G)$, which by the property shown before (and since $\mathcal{T}$ is quasi well designed) is equal to $[\![\mathcal{T}_r^{\emptyset}]\!]_G = [\![\mathcal{T}]\!]_G$. This completes the proof of the theorem. □

Recall that in Definition 3.12 we defined the semantics of QWDPTs by their extensions to well-designed SPARQL patterns. Theorem 3.16 now allows us to define the semantics of QWDPTs directly via their tree representation.

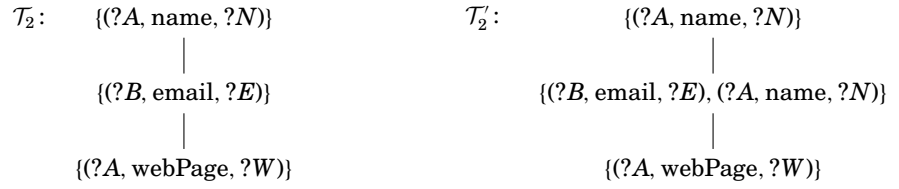
As already noted at the beginning of this subsection, this alternative, procedural semantics immediately gives rise to an evaluation procedure for QWDPTs. Recall that for well-designed graph patterns, the problem EVALUATION is coNP-complete [Pérez et al. 2009]. Since well-designed graph patterns can be transformed in polynomial time into equivalent QWDPTs and vice versa, the same complexity result also applies to QWDPTs. In fact, the complexity of evaluating QWDPTs greatly depends on the complexity of evaluating the BGP at each node of the tree: beside the evaluation problem, by a clever implementation of the top-down semantics, even the enumeration problem can be solved efficiently for a QWDPT if the corresponding problem can be solved efficiently for each BGP at each node of the tree (note that this would not be the case by a straightforward implementation of the top-down semantics). Before discussing the evaluation and enumeration problem for QWDPTs in more detail in Section 4, we first introduce in Section 3.3 several transformations on QWDPTs that may simplify the structure of a QWDPT. Providing a useful tool for the evaluation of QWDPTs, these transformations will play an important role in Section 4 and are thus presented first. For example, they will allow us to provide a much simpler proof of the coNP-completeness of EVALUATION than the one given in Pérez et al. [2009].

Furthermore, the top-down evaluation supports the idea of considering QWDPTs as a first step towards high-level query execution plans for well-designed SPARQL query patterns: they provide a syntactical representation of a query together with an evaluation method working on this representation. In these terms, the relaxation from well-designed pattern trees to QWDPTs provides additional potential for optimization and redundancy elimination for those query plans.

3.3. Transformation of QWDPTs

The results in Section 3.2 already suggest that there may exist several syntactically different, yet equivalent QWDPTs, as illustrated by the following example.

Example 3.17. Consider the QWDPT $\mathcal{T}_2$ from Example 3.5 together with an equivalent QWDPT $\mathcal{T}_2'$ that was retrieved from $\mathcal{T}_2$ by duplicating the triple (?A, webPage, ?W) from the root to its child.



The goal of this section is to identify preferable representations of equivalent QWDPTs. We do so by identifying nonoptimal structures within a pattern tree and providing a procedure for how to resolve them. This shows one advantage of QWDPTs, namely that they allow us to easily talk about the structure of queries and to define several equivalence-preserving transformations on the structure of the pattern trees. Previous works [Pérez et al. 2009; Schmidt et al. 2010] on transformation rules for SPARQL patterns have been based on the properties of the SPARQL operators. In contrast, the transformations that we introduce in this section are based on the tree structure of QWDPTs (i.e., the *operator structure*) and the structure of the sets of triple patterns composing the pattern tree.

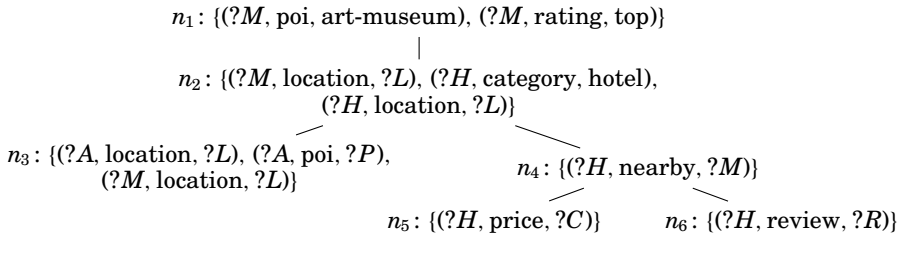
Before presenting our transformation rules, we need to introduce some additional notation. Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ be a pattern tree, and n a node in V. We define the *branch of n in $\mathcal{T}$*, denoted by $\text{branch}(n, \mathcal{T})$, as the unique path from r to n, given as the sequence of nodes $n^1, \ldots, n^k$ with $n^1 = r$ and $n^k = n$. If it is clear from the context, we may drop the name of the pattern tree and simply write $\text{branch}(n)$. We denote by $P_{\text{branch}(n,\mathcal{T})}$ the set of triple patterns $\bigcup_{i=1}^{k} P_{n^i}$. Given two sets P_1 and P_2 of triple patterns, a *homomorphism* h from P_1 into P_2, written $h\colon P_1 \to P_2$, is a mapping $h\colon \text{vars}(P_1) \to \mathbf{U} \times \mathbf{V}$ such that for all triple patterns $t \in P_1$ it holds that $h(t) \in P_2$, where $h(t)$ denotes the triple obtained from t by replacing all variables $?X \in \text{vars}(t)$ by $h(?X)$ and leaving URIs unchanged. It is further convenient to introduce the following notation to speak about variables occurring in some P_n.

Definition 3.18. Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ be a pattern tree and let $n, \hat{n} \in V$ be vertices such that $\hat{n}$ is the parent node of n. Then the *new variables at n* are defined as $\text{newvars}(n) = \text{vars}(P_n) \setminus \text{vars}(P_{\text{branch}(\hat{n})})$. For the case of the root r, we define $\text{newvars}(r)$ as $\text{vars}(P_r)$.

We are now ready to formally state a set of transformation rules for QWDPTs. In the formulation of the rules we assume that, whenever we remove a node n from a pattern tree, then all edges incident to n are removed as well. We further assume a fixed QWDPT $\mathcal{T} = ((V, E, r), (P_n)_{n \in V})$ to be the pattern tree before the application, and we consider $\mathcal{T}' = ((V', E', r'), (P'_n)_{n \in V'})$ as the resulting QWDPT after applying the rule. If P'_n is not defined explicitly for some $n \in V'$, we always consider $P'_n = P_n$ by "default".

We will use the following running example in order to explain the intuition behind these rules.

Example 3.19. Consider the following QWDPT $\mathcal{T}$.



Basically the query asks for the following information: first of all we are looking for a list of top-rated art museums (variable $?M$ in node n_1). In node n_2, for each such museum, we are interested in a list of hotels ($?H$) in the same area as the museum, if such information is available (thus this part of the query is attached as child node to the root). In case the information on hotels is available, on the one hand (node n_3) the query tries to retrieve information on further points-of-interest in the same area as the museum ($?A$) if such information can be found. On the other hand (node n_4), in case that the hotel is additionally located close to the museum, it also tries to derive the price for a room ($?C$ in node n_5) and possible reviews ($?R$ in node n_6) for the hotel. Note that by Definition 3.15, the semantics of the query is to start the evaluation at the root node, and then in a top-down traversal to extend every partial solution as long as possible to the new variables in the child nodes. This captures the intuition that the information queried further down in the tree would be "nice to have", but even if it is not present the query parts further up the tree may still return an answer.

The transformation rules presented next are all motivated by this top-down evaluation. The first rule is just the inverse of the concept of duplicating triples to children. Recall that we showed earlier that duplicating triples to children does not change the semantics of a QWDPT. From this it follows immediately that the inverse operation, that is, deleting triples from a node that occur in some ancestor of that node, preserves the semantics as well. Consider for example the triple $t = (?M, \text{location}, ?L)$ in node n_3 in Example 3.19. Since the same triple already occurs in P_{n_2}, we can safely remove it from P_{n_3}. Intuitively, the correctness of this rule can be seen in terms of the top-down evaluation as follows. For every mapping μ that we try to extend to n_3, we know that $\mu(P_{\text{branch}(n_2)}) \subseteq G$ (for some RDF graph G). Hence especially $\mu'(t) \subseteq G$ holds for every extension μ' of μ. This gives the first transformation rule, formally defined next.

Rule R1 (*deletion of redundant triples*): Let $n \in V$. If there exists a triple $t \in P_n$ such that, $t \in P_{n'}$ for some ancestor n' of n, then delete t from P_n, that is, $P'_n = P_n \setminus \{t\}$. If $P'_n = \emptyset$, delete n and turn its child nodes into children of the parent of n.

By a similar thought, also node n_4 in Example 3.19 shows a strange behavior: it does not introduce any "new" variable (i.e., all variables in P_{n_4} already occur somewhere in $P_{\text{branch}(n_2)}$). Hence "extending" a partial solution μ on branch(n_2) to n_4 is no extension in the sense of increasing the domain of μ, but it only enforces additional constraints on μ. However, independent of μ also mapping P_{n_4} into some given RDF graph G or not (i.e., independent of μ satisfying those additional constraints or not), the partial solution μ remains unchanged. We therefore refer to nodes like n_4 as *unproductive* nodes. We cannot always just delete such unproductive nodes though, since whether $\mu(P_{n_4}) \subseteq G$ or not makes a difference when going to the children of n_4. Extending μ to $?C$ and $?R$ should only be possible if indeed $\mu(P_{n_4}) \subseteq G$ holds. Hence the idea of the second rule is to defer this test $\mu(P_{n_4}) \subseteq G$ until it really makes a difference, and to remove it if it has no influence at all.

Rule R2 (*deletion of unproductive nodes*): Let $n, \hat{n} \in V$ such that $\hat{n}$ is the parent of n, and let $n_1, \ldots, n_k \in V$ be the children of n. If newvars(n) $= \emptyset$, then merge n into each of its children and make each n_i a child of $\hat{n}$. That is, let $P'_{n_i} = P_{n_i} \cup P_n$ for $i = \{1, \ldots, k\}$, $V' = V \setminus \{n\}$, and $E' = (E \setminus \{(\hat{n}, n), (n, n_1), \ldots, (n, n_k)\}) \cup \{(\hat{n}, n_1), \ldots, (\hat{n}, n_k)\}$. If n has no child node, then applying this rule is equivalent to deleting n.

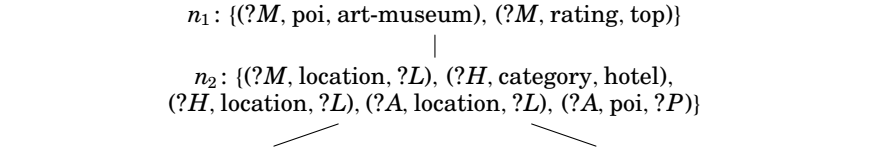
Hence applying Rule R2 to the QWDPT in Example 3.19 results in copying P_{n_4} into P_{n_5} and P_{n_6}, making n_5 and n_6 direct children of n_2, and deleting n_4.

So each application of Rule R2 reduces the number of nodes in a QWDPT (hence the number of OPT-operators in a corresponding SPARQL graph pattern). The next rule

aims at the same goal. Recall from the definition of the semantics of the left-outer join $M_1 ⟕ M_2$ that it behaves differently from the join operator $M_1 ⋈ M_2$ only if there exist mappings in M_1 for which there does not exist a compatible mapping in M_2. Hence if we can indeed guarantee that M_2 contains at least one compatible mapping for each mapping M_1, we could replace the left-outer join by a join, thus replace an OPT-operator by an AND-operator. This is for example the case in node n_3 in Example 3.19. Consider the mapping h defined as $h = \{?A \to ?M, ?L \to ?L, ?P \to \text{art-museum}, ?M \to ?M\}$ and that is the identity on all constants. Then it is easy to check that $\bigcup_{t \in P_{n_3}} h(t) \subseteq P_{\text{branch}(n_2)}$. Hence h is a homomorphism from P_{n_3} into $P_{\text{branch}(n_2)}$. Since in addition it is also the identity on all variables in P_{n_3} that already occurred in $P_{\text{branch}(n_2)}$, it is easy to see that given a partial solution μ on branch(n_2), the mapping $\mu' = \mu(h(?X))$ for all variables $?X \in \text{vars}(P_{n_3})$ is compatible with μ and thus a possible extension of μ to n_3. Hence we can safely merge the node n_3 into n_2, which corresponds to replacing the corresponding OPT-operator by an AND.

Rule R3 (*homomorphism upwards*): Let $n, \hat{n} \in V$ be nodes such that $\hat{n}$ is the parent of n, and let $n_1, \ldots, n_k \in V$ be the children of n. If there exists a homomorphism $h\colon P_n \to P_{\text{branch}(\hat{n})}$ with $h(?X) = ?X$ for all variables $?X \in \text{vars}(P_n) \cap \text{vars}(P_{\text{branch}(\hat{n})})$, then *merge* n into $\hat{n}$. That is, let $P'_{\hat{n}} = P_{\hat{n}} \cup P_n$, $V' = V \setminus \{n\}$ (remove n) and $E' = (E \setminus \{(\hat{n}, n), (n, n_1), \ldots, (n, n_k)\}) \cup \{(\hat{n}, n_1), \ldots, (\hat{n}, n_k)\}$ (turn n's child nodes into children of $\hat{n}$).

Example 3.20. We have seen that all Rules R1, R2, and R3 can be applied to the QWDPT shown in Example 3.19. The resulting pattern tree is shown next. Note that by applying those rules we were able to heavily simplify the structure of the QWDPT.



Note that the Rules R1, R2, and R3 will play an essential role for the further results in this article, especially for the complexity of the equivalence problem studied in Section 5.

Since the top-down evaluation allows one to work on different branches of a QWDPT independent of each other (which opens the door to parallel query processing), it is a natural goal to replace long branches of a tree by several shorter (and parallel) ones. An observation similar to the one that allowed us to develop Rule R3 provides a first step towards this goal. Instead of having a homomorphism from some node n only "upwards" into branch$(\hat{n})$ (with $\hat{n}$ being the parent of n), assume that this homomorphism in addition has the child n' of n as target. In this case we know that whenever some partial solution to branch$(\hat{n})$ can be extended directly to n' (i.e., without extending it to n before), then it can also be extended to n. Hence we can shift n' from being a child of n to being a child of $\hat{n}$.

Rule R4 (*parallelization*): Consider nodes $\hat{n}, n, n' \in V$ such that $\hat{n}$ is the parent of n, and n is the parent of n'. If there exists a homomorphism $h\colon P_n \to P_{n'} \cup P_{\text{branch}(\hat{n})}$ with $h(?X) = ?X$ for all variables $?X \in \text{vars}(P_n) \cap \text{vars}(P_{\text{branch}(\hat{n})})$, then turn n' from a child of n into a child of $\hat{n}$, if the resulting pattern tree is quasi well designed. That is, $V' = V$, $E = (E \setminus \{(n, n')\}) \cup \{(\hat{n}, n')\}$, if $\mathcal{T}'$ is still quasi well designed.

The next result formally shows that all these rules are indeed correct, that is, that they preserve the semantics of the QWDPT they are applied to.

THEOREM 3.21. *Let $\mathcal{T}$ be a QWDPT and $\mathcal{T}'$ the pattern tree that results from applying either Rule R1, or R2, or R3, or R4, to $\mathcal{T}$. Then $\mathcal{T}'$ is a QWDPT such that $\mathcal{T} \equiv \mathcal{T}'$.*

Recall that an intuitive description of why those rules are correct was already given before using the top-down semantics of QWDPTs. The formal proof is very technical and does not provide any more insights than the description given earlier. The complete proof of the theorem can be found in the electronic appendix.

Having this set of rules, our next goal is to identify QWDPTs that have some "nice" properties. Towards this goal, we say that a QWDPT $\mathcal{T}$ is *reduced* with respect to some rule R, if R cannot be applied to $\mathcal{T}$. While checking whether some QWDPT $\mathcal{T}$ is reduced with respect to R3 or R4 is an expensive task (it requires to decide the existence of some homomorphisms), it is rather easy to determine whether $\mathcal{T}$ is reduced with respect to R1 or R2. Moreover, already if $\mathcal{T}$ is reduced only with respect to R1 and R2, it possesses some useful properties that make it easier to work with—and reason about—$\mathcal{T}$. We thus introduce a first normal form for QWDPTs based on these two rules.

Definition 3.22. We say that a QWDPT $\mathcal{T}$ is in *nonredundant normal form (NR normal form)* if $\mathcal{T}$ is reduced with respect to Rules R1 and R2.

In the following, we discuss several properties of the NR normal form. The first result shows that for a given QWDPT this normal form is actually unique and can indeed be computed independently from possible rule applications of R3 and R4.

PROPOSITION 3.23. *Let $\mathcal{T}$ be a QWDPT. Then the following hold.*

(1) *Iteratively applying Rules R1 and R2 (in arbitrary order) to $\mathcal{T}$ leads to a unique pattern tree $\mathcal{T}^*$ in NR normal form.*
(2) *If $\mathcal{T}$ is in NR normal form then it remains in NR normal form when applying Rules R3 or R4 to $\mathcal{T}$.*

PROOF SKETCH. In order to prove (1), we show in the electronic appendix that for a QWDPT $\mathcal{T} = ((V, E, r), \mathcal{P})$ the unique NR normal form $\mathcal{T}^* = ((V^*, E^*, r), \mathcal{P}^*)$ consists of $V^* = V \setminus \{n \in V \mid \text{newvars}(n) = \emptyset\}$ and $E^* = \{(n, p(n)) \mid n \in V^* \setminus \{r\}\}$ where for every $n \in V^* \setminus \{r\}$ the node $p(n) \in V$ is the first ancestor of n in $\mathcal{T}$ (i.e., the ancestor closest to n) such that $p(n) \in V^*$. Finally, $P_n^* = P_{\text{branch}(n,\mathcal{T})} \setminus P_{\text{branch}(\hat{n},\mathcal{T})}$ for every $n \in V^*$ where $\hat{n}$ is the parent node of n in V^*.

Property (2) follows from the fact that the pattern tree is quasi well designed and the requirement that the homomorphism that allows to apply R3 and R4 is the identity on the shared variables. Using these facts, we show in the electronic appendix that for every node n in a QWDPT in NR normal form, the application of R3 or R4 cannot introduce any triple patterns in branch(n) that allows to apply R1 or R2 to n. □

The next result makes use of the crucial property of the NR normal form, which is the following. Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ be a QWDPT in NR normal form. Then for every $n \in V$ such that $n \neq r$, it holds that newvars(n) $\neq \emptyset$. This simple property, which follows directly from the definition of Rule R2, allows us to define an alternative characterization of the solutions of QWDPTs in terms of maximal subtrees. In fact, this property allows one to uniquely identify the subtree that may witness that some mapping μ is a solution to the QWDPT. In the characterization we use the following notation. Given a mapping μ and a set of mappings M, we say that M subsumes μ, denoted by $\mu \sqsubseteq M$, if there exists a mapping $\nu \in M$ such that $\mu \sqsubseteq \nu$.

LEMMA 3.24. *Let $\mathcal{T}$ be a QWDPT in NR normal form with root r, and G an RDF graph. Then $\mu \in [\![\mathcal{T}]\!]_G$ if and only if there exists a subtree $\mathcal{T}'$ of $\mathcal{T}$ rooted at r such that:*

(1) $\mathrm{dom}(\mu) = \mathrm{vars}(\mathcal{T}')$*, and*
(2) $\mathcal{T}'$ is the maximal subtree of $\mathcal{T}$ such that $\mu \sqsubseteq [\![and(\mathcal{T}')]\!]_G$.

PROOF. To prove the lemma we use the characterization of the evaluation of a QWDPT provided in Lemma 3.14. Thus assume first that $\mu \in [\![\mathcal{T}]\!]_G$ and let $\mathcal{T}'$ be the subtree of $\mathcal{T}$ mentioned in Lemma 3.14. That is, $\mu \in [\![and(\mathcal{T}')]\!]_G$ and there does not exist another subtree $\mathcal{T}''$ and a mapping $\nu \in [\![and(\mathcal{T}'')]\!]_G$ such that $\mu \sqsubset \nu$. First notice that $\mathrm{dom}(\mu) = \mathrm{vars}(\mathcal{T}')$ since $\mu \in [\![and(\mathcal{T}')]\!]_G$. Thus we only need to show that $\mathcal{T}'$ is the maximal subtree of $\mathcal{T}$ such that there exists $\nu \in [\![and(\mathcal{T}')]\!]_G$ with $\mu \sqsubseteq \nu$. To obtain a contradiction, assume that $\mathcal{T}'$ is not maximal. Thus, there exists another subtree $\mathcal{T}''$ that strictly contains $\mathcal{T}'$ as subtree such that $\mu \sqsubseteq \nu$ for some $\nu \in [\![and(\mathcal{T}'')]\!]_G$. We consider two cases.

—If $\mathrm{dom}(\mu) = \mathrm{dom}(\nu)$ then $\mathrm{vars}(\mathcal{T}'') = \mathrm{vars}(\mathcal{T}')$ which, since $\mathcal{T}''$ strictly contains $\mathcal{T}'$, contradicts the fact that $\mathcal{T}$ is reduced with respect to Rule R2.
—If $\mathrm{dom}(\mu) \neq \mathrm{dom}(\nu)$ then $\mu \sqsubset \nu$ which contradicts the characterization in Lemma 3.14 (since $\mathcal{T}''$ is a subtree with $\nu \in [\![and(\mathcal{T}'')]\!]_G$ and $\mu \sqsubset \nu$).

In any case we obtain a contradiction and thus, $\mathcal{T}'$ should be the maximal subtree satisfying that there exists $\nu \in [\![and(\mathcal{T}')]\!]_G$ with $\mu \sqsubseteq \nu$.

To prove the opposite direction, assume that $\mathrm{dom}(\mu) = \mathrm{vars}(\mathcal{T}')$ and $\mathcal{T}'$ is the maximal subtree of $\mathcal{T}$ for which there exists a mapping $\nu \in [\![and(\mathcal{T}')]\!]_G$ with $\mu \sqsubseteq \nu$. From these two properties it is straightforward to conclude that $\mu \in [\![and(\mathcal{T}')]\!]_G$. Thus, we only need to prove that there does not exist another subtree $\mathcal{T}''$ and a mapping $\nu \in [\![and(\mathcal{T}'')]\!]_G$ such that $\mu \sqsubset \nu$. To obtain a contradiction, assume that there exists such a subtree $\mathcal{T}''$. Notice that since $\mu \sqsubset \nu$ then $\mathrm{dom}(\mu) \subsetneq \mathrm{dom}(\nu)$ which implies that $\mathrm{vars}(\mathcal{T}') \subsetneq \mathrm{vars}(\mathcal{T}'')$. We claim that since $\mathcal{T}$ is reduced with respect to Rule R2, then $\mathrm{vars}(\mathcal{T}') \subsetneq \mathrm{vars}(\mathcal{T}'')$ implies that $\mathcal{T}'$ is a subtree of $\mathcal{T}''$. On the contrary, assume that $\mathcal{T}'$ is not a subtree of $\mathcal{T}''$. Then there exists a node n in $\mathcal{T}'$ which is not in $\mathcal{T}''$. Notice that since n is not in $\mathcal{T}''$ then no descendant of n is in $\mathcal{T}''$. Moreover, since $\mathcal{T}$ is reduced with respect to Rule R2, we know that $\mathrm{newvars}(n) \neq \emptyset$, and thus we have that there exists a variable $?X$ such that: (1) $?X$ occurs in n, (2) $?X$ does not occur in any ancestor of n in $\mathcal{T}$, and (3) $?X$ occurs in a node n' in $\mathcal{T}''$ which is not a descendant of n. Properties (1), (2), and (3) contradict the fact that $\mathcal{T}$ is a QWDPT. Thus we have that necessarily $\mathcal{T}'$ is a subtree of $\mathcal{T}''$. Moreover, since $\mathrm{vars}(\mathcal{T}') \subsetneq \mathrm{vars}(\mathcal{T}'')$ we have that $\mathcal{T}'$ is a proper subtree of $\mathcal{T}''$. Finally since there exists a mapping $\nu \in [\![and(\mathcal{T}'')]\!]_G$ and $\mu \sqsubseteq \nu$ we obtain that $\mathcal{T}'$ cannot be a maximal subtree satisfying the condition in Lemma 3.24. This is our desired contradiction. □

Notice that as opposed to Lemma 3.14 that characterizes the mappings in the evaluation of a QWDPT as the maximal (with respect to $\sqsubseteq$) mappings satisfying some property, Lemma 3.24 takes advantage of the NR normal form to characterize mappings in terms of the structure of a QWDPT, in particular, in terms of maximal subtrees.

The NR normal form provides a "cheap" elimination of some redundancies. As such it will be an integral part of the equivalence test for QWDPTs in Section 5.2. While the NR normal form proves helpful in many situations, especially in Section 5.2, it will not always be sufficient to assume NR normal form. Instead, it will turn out to be useful if the QWDPT is further reduced with respect to R3 which can reduce some more complex sources of redundancy in the structure of trees. As a result, the following normal form will play an important role in the equivalence test.

Definition 3.25. Let $\mathcal{T}$ be a QWDPT. We say that $\mathcal{T}$ is in *R3 normal form* if $\mathcal{T}$ is reduced with respect to Rules R1, R2, and R3.

One intuition of the R3 normal form is that given some QWDPT $((V, E, r), \mathcal{P})$ in this normal form, for every $n \in V$ with parent $\hat{n}$, there exists at least one RDF graph G and mapping μ with $\mu(P_{\text{branch}(\hat{n})}) \subseteq G$ that cannot be extended to a mapping μ' such that $\mu'(P_n) \subseteq G$. That is, from the fact that some variable assignment maps some $P_{\text{branch}(\hat{n})}$ into G, we cannot derive any statement about P_n. This intuitively implies that every node in the tree carries some information which is nonredundant with respect to its ancestors.

The next proposition shows that the R3 normal form exists and can be reached in polynomially many steps. It also identifies a simple condition under which this normal form is unique. Its proof is provided in the electronic appendix.

PROPOSITION 3.26. *Let $\mathcal{T}$ be a QWDPT. Then the following hold.*

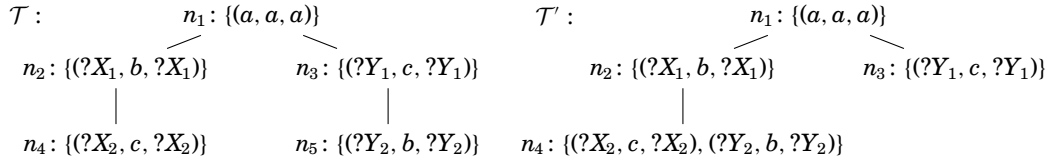
(*1*) *Iteratively applying R1, R2, and R3 to $\mathcal{T}$ eventually leads to a (not necessarily unique) pattern $\mathcal{T}^*$ that is in R3 normal form. Moreover, if $\mathcal{T}$ is in NR normal form, then iteratively applying R3 leads to a unique pattern $\mathcal{T}^*$ in R3 normal form.*
(*2*) *The number of rule applications of R1, R2, and R3 needed to arrive at a pattern in R3 normal form is linear in the size of $\mathcal{T}$.*

As pointed out, one goal of applying the transformation rules is to remove redundancies from QWDPTs, and thus to reduce the size of the QWDPT. A natural question is therefore whether the R3 normal form is able to guarantee some kind of minimality of QWDPTs. As usual, different minimization criteria for QWDPTs could be considered, like the number of variables, the number of triple patterns, or the number of OPT-operators. We shortly discuss these properties next, showing that QWDPTs in R3 normal form are not minimal with respect to these measures. Identifying transformations that guarantee to lead to a minimal pattern with respect to these measures remains as part of our future work.

First of all, we observe that the number of variables in a QWDPT cannot be reduced. In fact, we will show in Theorem 5.10 that equivalent QWDPTs always contain the same set of variables. The same theorem shows that for a QWDPT in NR normal form, the number of *different* triple patterns is already minimal, since equivalent QWDPTs in NR normal form must contain the same set of triple patterns. Another aspect is the overall number of triple patterns in a QWDPT, that is, the number considering repeated occurrences of the same pattern. Obviously, Rule R1 reduces their number. However, it can be easily seen that the resulting QWDPT not necessarily contains a minimal number of triple patterns. Even more, the application of Rule R2 may even increase the number of triple patterns: When applied to a node containing k triple patterns and having ℓ children, applying R2 increases the number of triple patterns by $k \cdot (\ell - 1)$. Thus, there may exist a trade-off between the number of triple patterns and OPT-operators in a QWDPT. As mentioned before, the presented transformation rules are rather aimed towards the minimization of the OPT-operators. As a result, the normal forms do not guarantee the QWDPT to contain a minimal number of triple patterns.

Although our transformation rules are aimed towards reducing the number of nodes (thus, the number of OPT-operators), even a QWDPT in R3 normal form need not contain a minimal number of OPT-operators, as shown by the following example.

Example 3.27. Consider the QWDPTs $\mathcal{T}$ and $\mathcal{T}'$ as depicted next.



Obviously, both, $\mathcal{T}$ and $\mathcal{T}'$ are in R3 normal form, and $\mathcal{T} \equiv \mathcal{T}'$. Thus, $\mathcal{T}$ is not minimal.

Identifying transformations that guarantee to lead to a QWDPT with an actually minimal number of OPT-operators remains as future work.

In this section, we have proposed a tree representation of SPARQL queries and a set of rules that can be used to restructure these trees. The results presented in this section therefore describe a starting point for the study of an algebra of query plans, which forms the basis of query optimization for this language. QWDPTs together with Rules R1–R3 will also be crucial for studying classical static analysis problems for SPARQL in the next sections. Rule R4 has been mainly presented so as to give a flavor of what further transformation rules in this algebra could look like. It may be beneficial in particular in an environment where parallel processing is supported.

4. EVALUATION AND ENUMERATION OF WELL-DESIGNED SPARQL

As promised in Section 3.2, we now take a closer look on the evaluation and enumeration problem for QWDPTs, and thus for well-designed SPARQL graph patterns. First of all, note that BGPs are essentially CQs over a relational schema with a single ternary predicate. Hence one way to see QWDPTs is as a collection of CQs whose results are combined according to the tree structure. Looking for tractable fragments of the evaluation (refer to Definition 2.2) or enumeration problem of QWDPTs, it is therefore natural to try to establish a relationship between tractable fragments of CQs and QWDPTs.

Conjunctive Query (CQ) evaluation[2] is a classical NP-complete problem [Chandra and Merlin 1977]. A lot of effort has thus been invested into the search for tractable fragments of CQs [Yannakakis 1981; Chekuri and Rajaraman 2000; Flum et al. 2002; Gottlob et al. 2002, 2000; Greco and Scarcello 2010a]. Typical tractable fragments are Acyclic CQs (ACQs) [Yannakakis 1981], CQs with bounded treewidth [Flum et al. 2002] or bounded hypertree width [Gottlob et al. 2002]. This search for tractable fragments of CQs has also been extended to the enumeration problem (i.e., given a CQ Q and a database D, output all tuples in the result of Q over D) [Flum et al. 2002; Bagan et al. 2007; Greco and Scarcello 2010b].

We now want to extend the study of tractable fragments of CQ evaluation to tractable fragments of evaluating well-designed SPARQL graph patterns. For the decision problem (i.e., EVALUATION; refer to Definition 2.2), tractable fragments of CQ evaluation immediately carry over to tractable fragments of SPARQL evaluation. For the enumeration problem (i.e., given an RDF graph G and a well-designed SPARQL graph pattern P, compute all solutions μ) a much more detailed analysis is required. In both cases, we first discuss the general relationship between the problem for CQs and QWDPTs, and then show how this relationship can be exploited to extend tractable fragments from CQs to QWDPTs.

In the following, we say that a set P of triple patterns *is from a tractable fragment of CQ evaluation* if, given an RDF graph G, the existence of a mapping $\mu\colon \text{vars}(P) \to \text{dom}(G)$ with $\mu(P) \subseteq G$ can be decided in polynomial time. Also, analogously to CQs, a join tree for a set P of triple patterns is a pair (T, λ) of a tree T and a function $\lambda\colon V(T) \to P$ that satisfies the following properties: (1) $\bigcup_{n \in V(T)} \{\lambda(n)\} = P$ and (2) for every pair t_1, t_2 of triple patterns in P, it holds for every node $n \in V(T)$ on the unique path between n_1 and n_2 (where $n_1, n_2 \in V(T)$ such that $\lambda(n_1) = t_1$ and $\lambda(n_2) = t_2$) that $\text{vars}(t_1) \cap \text{vars}(t_2) \subseteq \text{vars}(\lambda(n))$. We call a set P of triple patterns *acyclic* if there exists a join tree for P.

[2]There are several strongly related problems such as asking whether a given tuple is contained in the result of a given CQ over a given database, or asking whether a given Boolean CQ evaluates to true over a given database, or query containment, etc. All these problems have straightforward reductions between each other. By slight abuse of notation we thus simply speak of "CQ evaluation" to refer to any of these problems.

4.1. Evaluation of QWDPTs

We start with a look at the decision problem EVALUATION (refer to Definition 2.2), which was shown coNP-complete in Pérez et al. [2009]. For our representation of SPARQL graph patterns as QWDPTs, a coNP test can work as follows. Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ be a QWDPT and assume that it is in NR normal form (which can be computed in polynomial time). By using the characterization of the evaluation of QWDPTs provided in Lemma 3.24, in order to check whether μ is a solution of $\mathcal{T}$ over G, the coNP algorithm can first find a subtree $\mathcal{T}'$ of $\mathcal{T}$ rooted at r such that $\mathrm{dom}(\mu) = \mathrm{vars}(\mathcal{T}')$. Notice that if this subtree exists, then it is unique (since $\mathcal{T}$ is in NR normal form), and thus, this step can be done in polynomial time. Then the algorithm checks that $\mathcal{T}'$ is a maximal subtree such that $\mu \sqsubseteq [\![and(\mathcal{T}')]\!]_G$. The latter test requires coNP power since we have to check that μ cannot be extended to match any of the sets of triple patterns at nodes "below" the leaf nodes of $\mathcal{T}'$. However, it is sufficient to check this for every child node of $\mathcal{T}'$ (i.e., for every child of a leaf node of $\mathcal{T}'$) individually: if μ can be extended to any child node, this immediately proves that it is not maximal. The single source of the coNP-hardness is thus the test that μ cannot be extended to some BGP, while the selection of possible nodes to attach to $\mathcal{T}'$ (hence the selection of the BGP to test) is not responsible for this hardness. We will make use of this property shortly.

Finally, note that this simple coNP algorithm heavily relies on the NR normal form from Section 3.3 (the coNP algorithm provided in Pérez et al. [2009] is considerably more involved).

Turning towards the relationship with CQs, clearly, if all sets of triple patterns are from tractable fragments of CQ evaluation, the problem of checking whether μ is a solution of $\mathcal{T}$ over G becomes tractable.

COROLLARY 4.1. *Suppose that we only consider QWDPTs (and thus well-designed SPARQL graph patterns), where for each node t the set P_t of triple patterns is from a tractable fragment of CQ evaluation. Then EVALUATION is also tractable for those QWDPTs.*

This follows immediately from the algorithm sketched before; instead of coNP power to test whether μ can be extended to some "child" node of $\mathcal{T}'$, this is now feasible in polynomial time. Note that tractability is required for each set P_t individually, hence for different nodes t and t', the sets P_t and $P_{t'}$ may belong to different tractable fragments. This observation can be strengthened even more. Note that the $\mathcal{T}'$ can be always identified in polynomial time, independent of the sets P_t being from tractable fragments of CQ evaluation or not. Since this property is only needed in order to test whether μ can be extended to some "child" node of $\mathcal{T}'$, it suffices if this property is satisfied by all those "child" nodes n_i. Furthermore, it is not necessary that each P_{n_i} is from a tractable fragment of CQ, but it suffices if $\mu(P_{n_i})$ is from such a fragment.

4.2. Enumeration of Well-Designed SPARQL

Next we take a look onto the enumeration problem for QWDPTs, that can be defined as follows.

Definition 4.2. Let ENUMERATION be the following problem.
INPUT: An RDF graph G and a QWDPT $\mathcal{T}$.
OUTPUT: All mappings $\mu \in [\![\mathcal{T}]\!]_G$.

First of all, we observe that already the enumeration problem for CQs is intractable. Recall that an appropriate notion of tractable enumeration has to take the size of the output into account. Indeed, even for a single CQ, the set of solutions can be exponentially big. Hence polynomial-time algorithms with respect to the input make

no sense in this case. Recall further that there exist several different notions of tractable enumeration [Johnson et al. 1988]. The weakest form are *output polynomial* algorithms. The only guarantee they provide is that they will output all results of a query in time polynomial in the size of the output. However, the output may start only after an exponential delay, or there may be exponential gaps between two outputs (exponential in the input size). A stronger concept is that of *polynomial delay* algorithms. These algorithms guarantee that the time to either compute the next solution or to detect that no further solution exists is polynomially bounded in the input size.

Thus, the intractability of CQ enumeration means that in general solutions cannot be enumerated in output polynomial time (unless $\mathsf{P} = \mathsf{NP}$; this follows from the observation that the evaluation of Boolean CQs is NP-complete). Thus, also the enumeration problem for CQs was often studied in the literature. See Bagan et al. [2007] and Greco and Scarcello [2010b] for recent results. This intractability also holds for CQs corresponding to BGPs, and thus enumerating all solutions to a QWDPT is intractable as well.

Another aspect of enumeration algorithms, that is also reflected in the notions of tractability given earlier, is that one usually prefers to retrieve the results incrementally over the running time of the algorithm, instead of having to wait until the algorithm finishes and only then to get all results at once.

However, note that even if we are given such an algorithm for CQs, a straightforward implementation of the semantics of QWDPTs as presented in Section 3.2 would still return all results to the QWDPT only at the end of the algorithm. Thus, before studying how tractable fragments for the enumeration problem of CQs carry over to QWDPTs, by describing a more clever way of combining the results for the different nodes of the tree, we will first establish a general relationship between the enumeration of solutions to CQs and QWDPTs (hence, SPARQL graph patterns).

A usual way to implement enumeration algorithms is in form of *iterators*, that is, they are implemented in terms of an object providing functions **next**() and **hasNext**(), where **next**() returns the next solution, while **hasNext**() returns *true* if there exists yet another solution. We will make use of iterators as well: on the one hand we assume to be provided with an iterator for CQ enumeration (that we will use to evaluate the sets of triple patterns at each node of a QWDPT), and on the other hand we provide an iterator for QWDPTs.

Following the presentation in Cohen et al. [2006], in order to increase readability we do not define the functions **next**() and **hasNext**() explicitly. Instead, the enumeration algorithm is described as an ordinary algorithm, and we consider iterators as constructs that take an enumeration algorithm as argument and provide the **next**() and **hasNext**() functions. That is, consider an iterator $I :=$ **new** Iterator($E(x)$) for an enumeration algorithm E with input x. In response to I.**next**() being called, the iterator executes $E(x)$ until it encounters **output**(A) for the first time. Then the execution of E is interrupted, and A is returned as the result of I.**next**(). At the next call of I.**next**(), the execution of E is continued at the position where it was last interrupted, that is, right after the last **output**(.) command executed (and the last state of E is restored). Once E terminates (instead of being interrupted), no further answer exists. The function **hasNext**() can be either implemented by continuing the execution of E and checking whether another result is generated or not, or (like in our case) it is implemented by checking the current state of E.

In the following, assume a QWDPT $\mathcal{T} = ((V, E, r), \mathcal{P})$ to be evaluated over some RDF graph G. The algorithm in Figure 1 assumes the existence of some enumeration algorithm EnumerateCQ(P_t,μ) that, given a set P_t of triple patterns and a partial variable assignment μ returns all extensions of μ to P_t over G. Since this problem is equivalent to enumerating all solutions to a CQ, we therefore consider EnumerateCQ(P_t,μ) as a

```
Enumerate(t,μ)
 1: cqit := new Iterator(EnumerateCQ(P_t, μ));
 2: while( cqit.hasNext() ){
    // let t_1, ..., t_k be the children of t
 3:    maxi := 0;
 4:    μ_curr := cqit.next();
 5:    for( i = 1 to k ){
 6:       it_i := new Iterator(Enumerate(t_i, μ_curr));
 7:       flag_i := it_i.hasNext();
 8:       if(flag_i){
 9:          μ_i := it_i.next();
10:          maxi := i;
11:       }
12:    }
13:    if( ⋀_{i=1}^{k} ¬flag_i ){
14:       output(μ_curr);
15:       continue;
16:    }
17:    repeat{
18:       output( μ_curr ∪ ⋃_{1≤i≤k∧flag_i=true} μ_i );
19:       continueflag := false;
20:       for( i = maxi downto 1 ){
21:          if(it_i.hasNext()){
22:             μ_i := it_i.next();
23:             continueflag := true;
24:             for( j = i+1 to maxi ){
25:                if( flag_j ) {
26:                   it_j := new Iterator(Enumerate(t_j, μ_curr));
27:                   μ_j := it_j.next();
28:                }
29:             }
30:             i := 0; // leave the for-loop
31:          }
32:       }
33:    } until( ¬continueflag)
34: }
```

Fig. 1. Iterator for SPARQL tree patterns.

black box. The idea of our (recursive) enumeration algorithm Enumerate(t,μ) is as follows. For $t \in V$ and a partial assignment μ, the algorithm first checks whether μ can be extended to P_t (lines 1–2). For each such extension μ_{curr}, it checks recursively for each child node t_i whether there exists an extension of μ_{curr} to P_{t_i} (lines 5–7). Next, for each t_i that has such an extension the first solution is stored, together with the biggest index i such that t_i provides a solution (lines 8–10). If μ_{curr} cannot be extended to any child node, then the algorithm just returns μ_{curr} as one extension of μ to the complete subtree rooted at t (line 14; recall that the execution of the **output**(.) statement ends the execution of the call to **next**(), and the control flow is returned to the caller), and then considers the next extension of μ on t (line 15 jumps to the next iteration of the while loop in line 2). If on the other hand μ_{curr} can be extended to some children of t (lines 17–33), the algorithm enumerates all these extensions as follows. (For the sake of simplicity, in the following we only consider the ℓ children to which μ_{curr} can be extended.) In lines 17–33, all possible solutions are created that can be built from combining the

extensions of μ_{curr} to $t_1, \ldots, t_\ell$. Note that the first possible extension for each t_i was saved in μ_i in line 9. After returning this solution (line 18), the solutions are enumerated by iterating over the solutions for $t_1, \ldots, t_\ell$ as follows. First, the child node t_i with the highest index i is identified that has yet another solution (lines 20–21). This extension is saved in μ_i (line 22), and for all children t_j with $i < j \leq \ell$ the iterators are reset to the first extension of μ_{curr}, which is stored in μ_j (lines 24–27). If such an index i exists, the new solution is returned in the next iteration of the repeat loop (line 18), otherwise all extensions of μ_{curr} have already been returned, and the algorithm terminates.

THEOREM 4.3. *The problem of enumerating all solutions of a QWDPT (and hence, of a well-designed SPARQL graph pattern) can be reduced in polynomial time (by a Turing reduction) to the problem of enumerating all solutions of CQs.*

PROOF IDEA. The iterator described in Figure 1 reduces the problem of enumerating all solutions of a QWDPT to the problem of enumerating all solutions of CQs. Moreover, neglecting the cost of the calls to the iterator for CQs, the algorithm in Figure 1 clearly works in polynomial time. □

With this result at hand, we now turn towards *tractable* enumeration. In fact, our goal is to identify conditions under which the enumeration of the solutions of a QWDPT is feasible with *polynomial delay*. For instance, acyclic CQs and CQs of bounded treewidth or hypertree width [Yannakakis 1981; Chekuri and Rajaraman 2000; Flum et al. 2002; Gottlob et al. 2002] have this property. From Theorem 4.3 we can now conclude that any tractability results for CQs immediately carry over to well-designed SPARQL graph patterns.

THEOREM 4.4. *Suppose that we only consider QWDPTs where the sets of triple patterns at each node are from fragments of CQ enumeration for which polynomial-time delay algorithms exist. Then the problem ENUMERATION can be solved with polynomial-time delay for such QWDPTs.*

An inspection of our iterator for QWDPTs reveals that Theorem 4.4 could be further strengthened: for the tractability of the enumeration problem, it is sufficient that the sets of triple patterns are from tractable fragments of CQ evaluation *after considering all "old variables" at each node as constants*, that is, after replacing for node n all variables $?X \in \text{vars}(P_n) \setminus \text{newvars}(n)$ by $\mu(?X)$. In general, such an elimination of variables from a CQ may yield a significantly bigger tractable class.

5. CONTAINMENT AND EQUIVALENCE

In this section we study the fundamental problems of containment and equivalence of well-designed SPARQL queries. Similarly to query languages on relational databases, these problems are crucial for query optimization. For containment we consider the *subsumption relation* ($\sqsubseteq$) introduced in Section 3.2 rather than the classical subset relation ($\subseteq$). Clearly, for CQs, the two notions coincide. However, in the presence of partial query answers, subsumption is the more natural notion of containment [Kanza et al. 2002; Arenas and Pérez 2011], and has also been considered in recent work to compare the evaluation of two patterns containing OPT-operators [Pérez et al. 2009; Arenas and Pérez 2011]. This is illustrated in the following example (taken from Arenas and Pérez [2011]).

Example 5.1 [*Arenas and Pérez 2011*]. Consider two SPARQL graph patterns $P_1 = (?X, n, ?Y)$ and $P_2 = (?X, n, ?Y)$ OPT $(?X, e, ?Z)$, and an RDF graph $G = \{(a, n, b), (a, e, c)\}$. Then $[\![P_1]\!]_G = \{\mu = \{?X \to a, ?Y \to b\}\}$, while $[\![P_2]\!]_G = \{\mu' = \{?X \to a, ?Y \to b, ?Z \to c\}\}$. Hence $P_1 \not\sqsubseteq P_2$. This is, however, unintuitive, since the answer to P_2

contains strictly more information than that to P_1, and it is easy to see that for no graph G, pattern P_2 returns fewer bindings than P_1.

For CQs without existentially quantified variables, deciding both the equivalence and containment of two queries are tractable problems. In the presence of existential quantifiers, they are classical NP-complete problems [Chandra and Merlin 1977]. In this article, we study equivalence and containment for well-designed SPARQL queries, or, equivalently, for our representation by quasi well-designed pattern trees (QWDPTs). We start in this section by considering the case without projection, and will extend our study to projection in Section 6. In contrast to CQs, the problems of testing for containment (in the form of subsumption) and equivalence differ in their computational complexity. Indeed, in this section we prove that subsumption between QWDPTs is Π_2^P-complete while the equivalence problem is NP-complete. The NP-membership will be the most difficult part to prove. The key to this NP-membership result is the R3 normal form introduced in the previous section and an appropriate extension of homomorphisms, which we shall refer to as "strong homomorphisms".

5.1. Complexity of Subsumption

We extend the definition of subsumption of mappings introduced in Section 3.2, to subsumption of sets of mappings. Given sets of mappings M_1 and M_2 we say that M_1 is subsumed by M_2, denoted by $M_1 \sqsubseteq M_2$, if for every $\mu_1 \in M_1$ there exists a $\mu_2 \in M_2$ such that $\mu_1 \sqsubseteq \mu_2$. For two QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$, we further say that $\mathcal{T}_1$ is subsumed by $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if $[\![\mathcal{T}_1]\!]_G \sqsubseteq [\![\mathcal{T}_2]\!]_G$ holds for every RDF graph G.

Finally, we introduce the notion of a *frozen RDF graph* that is not only useful for proving the next lemma, but will be used in several proofs in this section. Let P be a set of triple patterns and let fr be a bijective function that assigns to each $?X \in \text{vars}(P)$ a unique, new URI $fr(?X) = x$, and that maps constants onto themselves. Then the frozen RDF graph G for P is the set of triples $G = \{fr(t) | t \in P\}$. Furthermore, let fr^{-1} denote the inverse of fr, that is, $fr^{-1}(fr(?X)) = ?X$.

We are now ready to state our main characterization of subsumption between QWDPTs by providing a necessary and sufficient condition to test whether $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$.

LEMMA 5.2. *Consider QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ with roots r_1 and r_2, respectively. Then $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ if and only if for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that:*

(*1*) $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$*, and*
(*2*) *there exists a homomorphism from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over* $\text{vars}(\mathcal{T}_1')$.

PROOF. We first prove direction ($\Leftarrow$). Thus let $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ and $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$ be QWDPTs, and assume that for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that: (1) $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$, and (2) there exists a homomorphism from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over $\text{vars}(\mathcal{T}_1')$. Now, let G be an RDF graph and $\mu \in [\![\mathcal{T}_1]\!]_G$. We need to prove that there exists a mapping $\mu' \in [\![\mathcal{T}_2]\!]_G$ such that $\mu \sqsubseteq \mu'$.

Given that $\mu \in [\![\mathcal{T}_1]\!]_G$, from Lemma 3.14 we know that there exists a subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, such that $\mu \in [\![and(\mathcal{T}_1')]\!]_G$. Moreover, by hypothesis, we know that there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ such that $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$, and a homomorphism h from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ which is the identity over $\text{vars}(\mathcal{T}_1')$. Let $triples(\mathcal{T}_1')$ denote the set of triples in $\mathcal{T}_1'$, and similarly for $triples(\mathcal{T}_2')$. Then we know that $h(triples(\mathcal{T}_2')) \subseteq triples(\mathcal{T}_1')$. Moreover, given that $\mu \in [\![and(\mathcal{T}_1')]\!]_G$ we know that $\text{dom}(\mu) = \text{vars}(\mathcal{T}_1')$ and $\mu(triples(\mathcal{T}_1')) \subseteq G$. Thus, we have that $\mu(h(triples(\mathcal{T}_2'))) \subseteq G$,

which means that the mapping $\mu(h(\cdot))$ (i.e., the composition of μ and h) is in $[\![and(\mathcal{T}_2')]\!]_G$. Now, notice that $\text{dom}(\mu(h(\cdot))) = \text{vars}(\mathcal{T}_2')$. Thus, since $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$ and h is the identity over $\text{vars}(\mathcal{T}_1')$ we have that for every variable $?X \in \text{dom}(\mu)$ it holds that $?X \in \text{dom}(\mu(h(\cdot)))$ and $\mu(?X) = \mu(h(?X))$, which implies that $\mu \sqsubseteq \mu(h(\cdot))$. Finally, since $\mu(h(\cdot)) \in [\![and(\mathcal{T}_2')]\!]_G$, applying Lemma 3.14 we obtain either $\mu(h(\cdot)) \in [\![\mathcal{T}_2]\!]_G$ or there exists a mapping μ' such that $\mu(h(\cdot)) \sqsubseteq \mu'$ and $\mu' \in [\![\mathcal{T}_2]\!]_G$. In either case we have that there exists a mapping $\mu' \in [\![\mathcal{T}_2]\!]_G$ such that $\mu \sqsubseteq \mu'$, which proves the case.

To prove direction ($\Rightarrow$), assume that for every RDF graph G it holds that $[\![\mathcal{T}_1]\!]_G \sqsubseteq [\![\mathcal{T}_2]\!]_G$. We need to prove that for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that: (1) $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$, and (2) there exists a homomorphism from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over $\text{vars}(\mathcal{T}_1')$. Let $\mathcal{T}_1'$ be a subtree of $\mathcal{T}_1$ rooted at r_1, and call G_1 the RDF graph $fr(triples(\mathcal{T}_1'))$. Notice that by construction we have that $fr \in [\![and(\mathcal{T}_1')]\!]_{G_1}$. Thus, by Lemma 3.14 we know that there exists a mapping, say f', such that $fr \sqsubseteq f'$ and $f' \in [\![\mathcal{T}_1]\!]_{G_1}$. We are assuming that $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, thus, we know that there exists a mapping, say f'', such that $f' \sqsubseteq f''$ and $f'' \in [\![\mathcal{T}_2]\!]_{G_1}$. Applying Lemma 3.14 again, we know that there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ such that $f'' \in [\![and(\mathcal{T}_2')]\!]_{G_1}$. This implies that $\text{dom}(f'') = \text{vars}(\mathcal{T}_2')$ and $f''(triples(\mathcal{T}_2')) \subseteq G_1 = fr(triples(\mathcal{T}_1'))$. First, recall that $fr \sqsubseteq f''$ and thus, since $\text{dom}(fr) = \text{vars}(\mathcal{T}_1')$ and $\text{dom}(f'') = \text{vars}(\mathcal{T}_2')$, we obtain that $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$, thus proving property (1) given before. Now, recall that $fr(?X)$ is a fresh URI for every $?X \in \text{vars}(\mathcal{T}_1')$. Thus, let g be a function such that $g(fr(?X)) = ?X$ for every $?X \in \text{vars}(\mathcal{T}_1')$, that is, g is the *inverse* of fr (which is well defined since $fr(?X)$ is fresh for every $?X$). Consider now the mappings $g(f''(\cdot))$ and $g(fr(\cdot))$. Given that $f''(triples(\mathcal{T}_2')) \subseteq fr(triples(\mathcal{T}_1'))$, we have that $g(f''(triples(\mathcal{T}_2'))) \subseteq g(fr(triples(\mathcal{T}_1'))) = triples(\mathcal{T}_1')$, which implies that $g(f''(\cdot))$ is a homomorphism from $triples(\mathcal{T}_2')$ to $triples(\mathcal{T}_1')$. Moreover, since $fr \sqsubseteq f''$ we have that for every $?X \in \text{vars}(\mathcal{T}_1')$ it holds that $f''(?X) = fr(?X)$ which implies that for every $?X \in \text{vars}(\mathcal{T}_1')$ it holds that $g(f''(?X)) = g(fr(?X)) = ?X$. Thus we have that $g(f''(\cdot))$ is a homomorphism from $triples(\mathcal{T}_2')$ to $triples(\mathcal{T}_1')$ which is the identity over $\text{vars}(\mathcal{T}_1')$, thus proving property (2) given before. This completes the proof. □

Note that Lemma 5.2 yields a straightforward Π_2^P procedure to test whether $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ holds: for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, check whether there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 and a homomorphism satisfying properties (1) and (2). In what follows we also show the matching lower bound.

THEOREM 5.3. *The subsumption problem of QWDPTs (and, therefore, of well-designed SPARQL graph patterns) is Π_2^P-complete.*

PROOF SKETCH. We first prove the Π_2^P-membership. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be QWDPTs with roots r_1 and r_2, respectively. Consider first the following problem: given a fixed subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, check whether there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that: (1) $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$ and (2) there exists a homomorphism h from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over $\text{vars}(\mathcal{T}_1')$. This problem is clearly in NP, since we can guess the subtree $\mathcal{T}_2'$ and the homomorphism h, and then check in polynomial time that (1) and (2) are satisfied.

From the preceding observation and Lemma 5.2, it follows easily that checking whether $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ is in Π_2^P. From Lemma 5.2 we know that in order to check $\mathcal{T}_1 \not\sqsubseteq \mathcal{T}_2$, it is enough to guess a subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1 and then check that there is no subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that: (1) $\text{vars}(\mathcal{T}_1') \subseteq \text{vars}(\mathcal{T}_2')$ and (2) there exists a homomorphism from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over $\text{vars}(\mathcal{T}_1')$. This last property can be checked using an NP-oracle, and therefore the whole process of checking $\mathcal{T}_1 \not\sqsubseteq \mathcal{T}_2$ can be done in NP^{NP}. This implies that checking $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, is in Π_2^P.

We show the Π_2^P*-hardness* by reduction from the well-known Π_2^P-hard problem 3-QSAT$_{\forall,2}$. Let an arbitrary instance of this problem be given by a quantified Boolean formula $\Psi = \forall \vec{x}\, \exists \vec{y}\, (C_1 \wedge \cdots \wedge C_m)$ where each $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ (for $i \in \{1, \ldots, m\}$) is a disjunction of three literals over the variables $\vec{x} = (x_1, \ldots, x_k)$ and $\vec{y} = (y_1, \ldots, y_\ell)$.

We define two QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ with $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ and $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$ as follows. In order to keep the proof readable, we do not use triple syntax in this reduction, but use atoms with binary and ternary predicate symbols instead. We shall give a translation of these atoms into triple syntax at the end of the proof. Now let $V_1 = \{r_1, n_1, \ldots, n_k\}$ and $E_1 = \{(r_1, n_i) \mid 1 \leq i \leq k\}$, that is, r_1 is the root and all other nodes are children of r_1. Let further

$$P_{r_1} = \{s(?U, ?U), r(0,1), r(1,0)\} \cup \{q_i(d,0,1) \mid 1 \leq i \leq k\} \\ \cup \{c(0,0,1), c(0,1,0), c(1,0,0), c(1,0,1), c(1,1,0), c(0,1,1), c(1,1,1)\} \text{ and}$$
$$P_{n_i} = \{q_i(?Z_i, 1, 0)\} \text{ for } i \in \{1, \ldots, k\},$$

where we introduce one new variable $?Z_i$ for each variable $x_i \in \vec{x}$.
Finally, let $V_2 = \{r_2\}$ and $E_2 = \emptyset$, with

$$P_{r_2} = \{s(?U, ?U)\} \cup \{r(?Y_j, ?\bar{Y}_j) \mid 1 \leq j \leq \ell\} \cup \{q_i(?Z_i, ?X_i, ?\bar{X}_i) \mid 1 \leq i \leq k\} \\ \cup \{c(l^*_{i,1}, l^*_{i,2}, l^*_{i,3}) \mid 1 \leq i \leq n\}$$

where $?Y_j, ?\bar{Y}_j$ are new variables for every $y_j \in \vec{y}$ and $?X_i, ?\bar{X}_i$ are new variables for every $x_i \in \vec{x}$. The $?Z_i$ are the same as in $\mathcal{T}_1$, and

$l^*_{i,j} = ?X_\alpha$ (respectively, $?Y_\beta$) if $l_{i,j} = x_\alpha$ (respectively, y_β) and
$l^*_{i,j} = ?\bar{X}_\alpha$ (respectively, $?\bar{Y}_\beta$) if $l_{i,j} = \neg x_\alpha$ (respectively, $\neg y_\beta$).

The intuition behind the reduction is best explained in terms of Lemma 5.2. Consider an arbitrary subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1. The crucial idea is that $\mathcal{T}_1'$ encodes a truth assignment I on $\vec{x}$ as follows: if $n_i \in \mathcal{T}_1'$, then $I(x_i) = \mathit{true}$, otherwise $I(x_i) = \mathit{false}$. Now recall that according to Lemma 5.2, $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ iff there exists a subtree $\mathcal{T}_2' \subseteq \mathcal{T}_2$ together with a homomorphism that is the identity on vars$(\mathcal{T}_1')$ that maps the triple patterns in $\mathcal{T}_2'$ into those of $\mathcal{T}_1'$. The encoding of the truth assignment is now enforced as follows: The only possible subtree of $\mathcal{T}_2$ is $\mathcal{T}_2$ itself. Hence for all $i \in \{1, \ldots, k\}$ s.t. $n_i \in \mathcal{T}_1'$, the required homomorphism h must map $?Z_i$ onto $?Z_i$, and therefore $?X_i$ onto 1 and $?\bar{X}_i$ onto 0, which encodes $I(x_i) = \mathit{true}$ and $I(\neg x_i) = \mathit{false}$. On the other hand, if $n_i \notin \mathcal{T}_1'$, then h must map $?Z_i$ onto d, which enforces $h(?X_i) = 0$ and $h(?\bar{X}_i) = 1$, which encodes $I(x_i) = \mathit{false}$ and $I(\neg x_i) = \mathit{true}$. Finally the idea is that for every possible subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1 the homomorphism h defined the way described before can be extended to the variables $?Y_1, ?\bar{Y}_1, \ldots, Y_\ell, ?\bar{Y}_\ell$ in a way that it maps the set of triple patterns in $\mathcal{T}_2$ into those of $\mathcal{T}_1'$ iff the corresponding truth assignment I on $\vec{x}$ can be extended to $\vec{y}$ in a way that $C_1, \wedge \cdots \wedge C_m$ evaluates to *true*.

The proof of the correctness of the reduction is given in the electronic appendix to this article. We close the proof by showing how the ternary symbols q_i and c can be replaced by a collection of triples. This can be done as follows: Each atom $q_i(\alpha, \beta, \gamma)$ is simply replaced by three triples $(1, q_i, \alpha)$, $(2, q_i, \beta)$, and $(3, q_i, \gamma)$. For the c-atoms, some more care is required. Let $c(\alpha, \beta, \gamma)$ be the j-th atom (with $j \in \{1, \ldots, 7\}$) with leading symbol c in P_{r_1}. Then we replace it by 3 triples (j, c_1, α), (j, c_2, β), and (j, c_3, γ). Moreover, we replace each atom $c(l^*_{i,1}, l^*_{i,2}, l^*_{i,3})$ in P_n by 3 triples $(?A_i, c_1, l^*_{i,1})$, $(?A_i, c_2, l^*_{i,2})$, and $(?A_i, c_3, l^*_{i,3})$ for a new variable $?A_i$. □

For equivalence and containment of CQs it is well known that $Q_1 \equiv Q_2$ iff $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. The next result establishes the same close connection between subsumption and equivalence of well-designed SPARQL queries.

LEMMA 5.4. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two QWDPTs. Then $\mathcal{T}_1 \equiv \mathcal{T}_2$ if and only if $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ and $\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$.*

PROOF. The "only if" part is straightforward. To prove the "if" part, let $P_1 \in \text{SEM}(\mathcal{T}_1)$ and $P_2 \in \text{SEM}(\mathcal{T}_2)$, and let G be an arbitrary graph. We need to prove that $[\![P_1]\!]_G = [\![P_2]\!]_G$. Let μ be a mapping in $[\![P_1]\!]_G$. We show next that $\mu \in [\![P_2]\!]_G$. Intuitively, if this is not the case, then because of the mutual subsumption, there must exist an extension of μ in $[\![P_1]\!]_G$, which contradicts the assumption that $\mu \in [\![P_1]\!]_G$. Formally, given that $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, we know that $[\![P_1]\!]_G \sqsubseteq [\![P_2]\!]_G$, and thus, there exists a mapping μ' in $[\![P_2]\!]_G$ such that $\mu \sqsubseteq \mu'$. Similarly, given that $\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$ we have that $[\![P_2]\!]_G \sqsubseteq [\![P_1]\!]_G$, and thus, there exists $\mu'' \in [\![P_1]\!]_G$ such that $\mu' \sqsubseteq \mu''$. Therefore we have that $\mu \sqsubseteq \mu''$ and thus μ and μ'' are compatible mappings. In Pérez et al. [2009] (see Claim 3.9) the authors proved that for a SPARQL pattern P constructed using only AND- and OPT-operators, and an arbitrary graph G, if $\nu_1, \nu_2 \in [\![P]\!]_G$ are compatible mappings, then $\nu_1 = \nu_2$. In our case, given that $\mu, \mu'' \in [\![P_1]\!]_G$ are compatible mappings, we obtain that $\mu = \mu''$. Finally, since $\mu \sqsubseteq \mu' \sqsubseteq \mu''$ we obtain that $\mu = \mu'$ and then $\mu \in [\![P_2]\!]_G$. This proves that $[\![P_1]\!]_G \subseteq [\![P_2]\!]_G$. Similarly, it can be proved that $[\![P_2]\!]_G \subseteq [\![P_1]\!]_G$, and thus $[\![P_1]\!]_G = [\![P_2]\!]_G$. □

From Theorem 5.3 and Lemma 5.4 we obtain that equivalence of well-designed SPARQL queries can be tested in Π_2^P. However, in the next section we provide a better upper bound, namely NP.

5.2. Complexity of Equivalence

We now turn towards testing the equivalence of two QWDPTs (and, thus of two well-designed SPARQL graph patterns). The NP-membership proof consists of three major steps. First, we introduce as a key concept the notion of a *strong homomorphism* between two branches of one or two pattern trees and use it to define the notion of *strongly homomorphically equivalent* branches. In order to describe the intuition of these concepts, we also show some of their basic properties. This first step is provided in Section 5.2.1. As the second step, we provide a strong homomorphism-based characterization of equivalent QWDPTs in R3 normal form (we observe that testing for the existence of a strong homomorphism is NP-hard). Since computing the R3 normal form of a QWDPT is not in NP, in the last step we show that actually computing the R3 normal form is not necessary, but guessing a polynomial number of R3 rule applications is sufficient. The second and third step are presented in Section 5.2.2. Most of the proof details of this section can be found in the electronic appendix of this article.

5.2.1. Strong Homomorphisms.

Definition 5.5 (*Strong Homomorphism*). Consider two QWDPTs $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ and $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$. Moreover, let $n_1 \in V_1$, $n_2 \in V_2$, and let $\text{branch}(n_1, \mathcal{T}_1)$ be the sequence of nodes $r_1 = n^1, \ldots, n^k = n_1$.

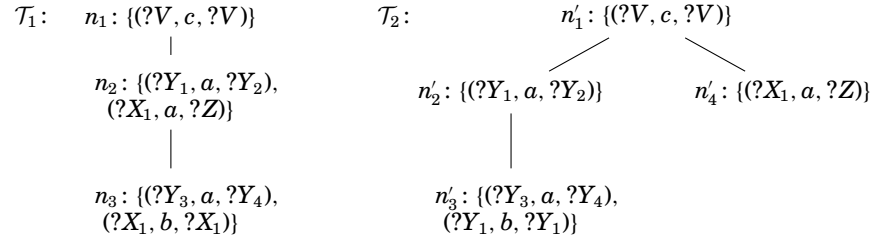
We say that there exists a *strong homomorphism* $\mathcal{H}\colon \text{branch}(n_1, \mathcal{T}_1) \to \text{branch}(n_2, \mathcal{T}_2)$ if $\mathcal{H}$ is a set $\mathcal{H} = \{h_i \mid 1 \leq i \leq k\}$ of homomorphisms $h_i\colon P_{n^i} \to P_{\text{branch}(n_2,\mathcal{T}_2)} \cup P_{\text{branch}(n^{i-1},\mathcal{T}_1)}$ such that $h_i(?X) = ?X$ for all $?X \in \text{vars}(P_{n^i}) \cap \text{vars}(P_{\text{branch}(n_2,\mathcal{T}_2)} \cup P_{\text{branch}(n^{i-1},\mathcal{T}_1)})$ (where for $i = 1$ let $P_{\text{branch}(n^{i-1},\mathcal{T}_1)} = \emptyset$).

We further say that $\text{branch}(n_1, \mathcal{T}_1)$ and $\text{branch}(n_2, \mathcal{T}_2)$ are *strongly homomorphically equivalent* if there exist strong homomorphisms $\mathcal{H}_1\colon \text{branch}(n_1, \mathcal{T}_1) \to \text{branch}(n_2, \mathcal{T}_2)$ and $\mathcal{H}_2\colon \text{branch}(n_2, \mathcal{T}_2) \to \text{branch}(n_1, \mathcal{T}_1)$.

The basic intuition of a strong homomorphism $\mathcal{H}\colon \text{branch}(n_1) \to \text{branch}(n_2)$ is that every variable assignment μ that maps $P_{\text{branch}(n_2)}$ into some RDF graph G can be extended to a variable assignment μ' that also maps $P_{\text{branch}(n_1)}$ into G. Note that a simple homomorphism $h\colon P_{\text{branch}(n_1)} \to P_{\text{branch}(n_2)}$ is not enough to guarantee this property.

Intuitively, the reason for this is that for two nodes n_1 and n_2 as in Definition 5.5, there may be RDF graphs G and variable assignments μ that not only map all triple patterns in branch(n_2) into G, but also the triple patterns contained in some "prefix" of branch(n_1). In order to extend such variable assignments to the complete branch branch(n_1), the existing assignments on the variables in this "prefix" of branch(n_1) must not be altered. This is illustrated in the following example.

Example 5.6. Consider the QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ defined as follows.



It is easy to see that there exists a homomorphism $h\colon P_{\text{branch}(n_3,\mathcal{T}_1)} \to P_{\text{branch}(n'_3,\mathcal{T}_2)}$ that is the identity on all shared variables. However, a strong homomorphism $\mathcal{H}\colon P_{\text{branch}(n_3,\mathcal{T}_1)} \to P_{\text{branch}(n'_3,\mathcal{T}_2)}$ does not exist, because a homomorphism $h_3\colon P_{n_3} \to P_{\text{branch}(n'_3,\mathcal{T}_2)} \cup P_{\text{branch}(n_2,\mathcal{T}_1)}$ with the properties required by Definition 5.5 is missing.

Now consider some variable assignment τ that is defined only on variables in branch($n'_3, \mathcal{T}_2$). If τ maps branch($n'_3, \mathcal{T}_2$) into an RDF graph G, then because of the homomorphism h there exists an extension τ' of τ that also maps branch($n_3, \mathcal{T}_1$) into G.

However, this is no longer the case for mappings that contain variables not occurring in branch($n'_3, \mathcal{T}_2$), but somewhere in the branch of n_3. As a concrete example, consider the mapping $\mu\colon \{?V \to v, ?Z \to z\} \cup \{?Y_i \to y_i \mid 1 \le i \le 4\} \cup \{?X_1 \to x_1\}$ over the RDF graph $G = \{(v, c, v), (y_1, a, y_2), (x_1, a, z), (y_3, a, y_4), (y_1, b, y_1)\}$. It can now be easily checked that $\mu \in [\![\mathcal{T}_2]\!]_G$ and that it maps branch($n'_3, \mathcal{T}_2$) into G. The homomorphism h suggests that we can conclude that μ also maps branch($n_3, \mathcal{T}_1$) into G (or can be extended to a mapping that does so). But $\mu(P_{\text{branch}(n_3,\mathcal{T}_1)}) \not\subseteq G$, because of $\mu((?X_1, b, ?X_1)) = (x_1, b, x_1) \notin G$. Hence, despite the existence of h and $\mu(\text{branch}(n'_3, \mathcal{T}_2)) \subseteq G$, μ is not a solution to $\mathcal{T}_1$, thus $\mathcal{T}_1 \not\equiv \mathcal{T}_2$.

Note that if we replace the pattern $(?X_1, b, ?X_1)$ in n_3 by $(?Y_1, b, ?Y_1)$, then there exists a strong homomorphism $\mathcal{H}\colon P_{\text{branch}(n_3,\mathcal{T}_1)} \to P_{\text{branch}(n'_3,\mathcal{T}_2)}$. Actually, $\mathcal{T}_1$ and $\mathcal{T}_2$ can then be shown to be equivalent.

This idea, which will be crucial for our NP equivalence test, is formalized in the following lemma.

LEMMA 5.7. *Consider two QWDPTs* $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ *and* $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$*, and let* $n_1 \in V_1$ *and* $n_2 \in V_2$ *with* $\text{branch}(n_1, \mathcal{T}_1) = n^1, \ldots, n^k$*. Then the following statements are equivalent.*

(1) *There exists a strong homomorphism* $\mathcal{H}\colon \text{branch}(n_1, \mathcal{T}_1) \to \text{branch}(n_2, \mathcal{T}_2)$.
(2) *For every* $i \in \{1, \ldots, k\}$*, for every RDF graph* G *and every mapping* μ: $\text{vars}(P_{\text{branch}(n_2,\mathcal{T}_2)} \cup P_{\text{branch}(n^{i-1},\mathcal{T}_1)}) \to \text{dom}(G)$ *the following holds:*
If $\mu(P_{\text{branch}(n_2,\mathcal{T}_2)} \cup P_{\text{branch}(n^{i-1},\mathcal{T}_1)}) \subseteq G$*, then there exists a mapping* μ': $\text{vars}(P_{\text{branch}(n_1,\mathcal{T}_1)}) \to \text{dom}(G)$ *such that* $\mu'(P_{\text{branch}(n_1,\mathcal{T}_1)}) \subseteq G$ *and* $\mu(?X) = \mu'(?X)$ *for all* $?X \in \text{dom}(\mu) \cap \text{dom}(\mu')$ *(where for* $i = 1$*, let* $P_{\text{branch}(n^{i-1})} = \emptyset$*).*

PROOF IDEA. The basic idea of showing that property (1) implies property (2) is to show the existence of μ' via induction along the path from n^{i-1} to n_1 (say $n^i, \ldots, n^k$): assuming that there exists a mapping μ_{j-1} mapping P_{n^j} into G (for $j \in \{i, \ldots, k\}$), the

mapping μ_j can be constructed by combining μ_{j-1} and $h_j \in \mathcal{H}$. Omitting some technical details, μ_j can be basically defined as $\mu_{j-1}(h_j(P_{n^j})) \cup \mu$. The idea is that h_j maps P_{n^j} into $P_{\text{branch}(n^{j-1},\mathcal{T}_1)} \cup P_{\text{branch}(n_2,\mathcal{T}_2)}$, from where either μ_{j-1} or μ (which agree on the shared variables) map it into G. For the other direction (i.e., (2) $\Rightarrow$ (1)), the existence of the required homomorphisms can be shown making use of the frozen RDF graph of parts of the query: consider the path $n^1, \ldots, n^k$ with $n^1 = r_1$ and $n^k = n_1$. Again, by omitting some technical details, each homomorphism $h_i \in \mathcal{H}$ (i.e., $i \in \{1, \ldots, k\}$) can be basically retrieved by considering $P = P_{\text{branch}(n^i,\mathcal{T}_1)} \cup P_{\text{branch}(n_2,\mathcal{T}_2)}$. Now for $G = fr(P)$, obviously $\mu(P) \subseteq G$ holds for the mapping μ defined on all variables $?X \in \text{vars}(P)$ as $\mu(?X) = fr(?X)$. The homomorphism h_i with all required properties can then be constructed from the extension μ' of μ (that exists by assumption) as $h_i(?X) = fr^{-1}(\mu'(?X))$ for all $?X \in \text{vars}(P_{n^i})$. The formal proof of the lemma is again provided in the electronic appendix of this article. □

We next discuss two further properties of strong homomorphisms on QWDPTs.

Recall that Example 5.6 shows that the existence of homomorphisms between two branches does not imply the existence of strong homomorphisms. An immediate corollary of the previous result, that will be also needed later, shows that the converse holds. Its proof, based on the frozen RDF graph of branch$(n_2, \mathcal{T}_2)$ and Lemma 5.7, is provided in the electronic appendix.

COROLLARY 5.8. *Consider two QWDPTs* $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ *and* $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$ *in R3 normal form, and let* $n_1 \in V_1$ *and* $n_2 \in V_2$.

If there exists a strong homomorphism $\mathcal{H}\colon \text{branch}(n_1, \mathcal{T}_1) \to \text{branch}(n_2, \mathcal{T}_2)$ *then there also exists a homomorphism* $h\colon P_{\text{branch}(n_1,\mathcal{T}_1)} \to P_{\text{branch}(n_2,\mathcal{T}_2)}$ *that is the identity on all variables* $?X \in \text{vars}(P_{\text{branch}(n_1,\mathcal{T}_1)}) \cap \text{vars}(P_{\text{branch}(n_2,\mathcal{T}_2)})$.

Lemma 5.7 holds in the general case that n_1 and n_2 belong to different QWDPTs. If n_1 and n_2 are from the same pattern tree $\mathcal{T}$, we can even show a slightly stronger result: Instead of talking about extensions of μ as in Lemma 5.7, we know that the property is already satisfied by μ itself. The formal proof of the proposition can again be found in the electronic appendix.

PROPOSITION 5.9. *Consider a QWDPT* $\mathcal{T} = ((V, E, r), \mathcal{P})$ *and nodes* $n_1, n_2 \in V$. *Then the following statements are equivalent.*

(1) *There exists a strong homomorphism* $\mathcal{H}\colon \text{branch}(n_1, \mathcal{T}) \to \text{branch}(n_2, \mathcal{T})$.
(2) *For every RDF graph* G *and* $\mu \in [\![\mathcal{T}]\!]_G$ *it holds that* $\mu(P_{\text{branch}(n_1)}) \subseteq G$ *whenever* $\mu(P_{\text{branch}(n_2)}) \subseteq G$.

5.2.2. Testing Equivalence of QWDPTs is in NP. As outlined earlier, the next step towards an NP test for equivalence of QWDPTs is a strong homomorphism-based characterization of equivalent QWDPTs in R3 normal form.

Recall that in QWDPTs, every variable occurs in some node that is an ancestor to all other nodes containing this variable. Given two QWDPTs $\mathcal{T}_1, \mathcal{T}_2$ in R3 normal form, the main aspect of the characterization of equivalence presented shortly is to require, for every variable, this node in $\mathcal{T}_1$ (say n_1) to be strongly homomorphically equivalent with the corresponding node in $\mathcal{T}_2$ (say n_2). Intuitively, this ensures that every variable mapping that maps branch$(n_1, \mathcal{T}_2)$ into some RDF graph G can be also extended to map branch$(n_2, \mathcal{T}_2)$ into G. By symmetry and the maximality of solutions, this implies that in fact every solution on one QWDPT is also a solution on the other QWDPT.

THEOREM 5.10. *Let* $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ *and* $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$ *be two QWDPTs in R3 normal form. Then* $\mathcal{T}_1 \equiv \mathcal{T}_2$ *if and only if:*

(1) $\bigcup_{n \in V_1} P_n = \bigcup_{n \in V_2} P_n$,
(2) $P_{r_1} = P_{r_2}$, *and*
(3) *for all pairs* (n_1, n_2) *of nodes* $n_1 \in V_1$ *and* $n_2 \in V_2$ *with* $\text{newvars}(n_1) \cap \text{newvars}(n_2) \neq \emptyset$ *it holds that* $\text{branch}(n_1, \mathcal{T}_1)$ *and* $\text{branch}(n_2, \mathcal{T}_2)$ *are strongly homomorphically equivalent.*

PROOF IDEA. In the following, we only sketch the main ideas of the proof, which can be found in the electronic appendix. For a better orientation, we will use the same names here as in the full proof, even if this may look sometimes a little unintuitive.

The general structure of the proof showing that $\mathcal{T}_1 \equiv \mathcal{T}_2$ implies the properties (1), (2), and (3) is as follows: Properties (1) and (2) are shown by providing a way how to create a counterexample in case the property does not hold. In both cases, this counterexample consists of the "frozen RDF graph" of some part of the "smaller" query, that is, that query that misses either a triple or a variable (in case that both pattern trees contain an element the other does not, one of them can be chosen arbitrarily). It is then shown that the "identity" mapping $\mu = fr$ is a solution to the "smaller" query, but not to the other one.

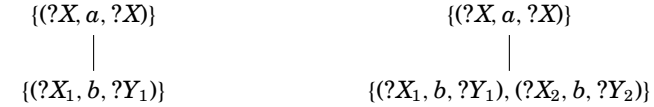
Instead of property (3), an obvious equivalent property is shown, namely: for every variable $?X \in \text{vars}(\mathcal{T}_1)$, the branches $\text{branch}(n_1, \mathcal{T}_1)$ and $\text{branch}(n_2, \mathcal{T}_2)$ are strongly homomorphically equivalent for $n_1 \in V_1$ such that $?X \in \text{newvars}(n_1)$ and $n_2 \in V_2$ such that $?X \in \text{newvars}(n_2)$. The proof is done by induction along some total order $<_v$ on the variables that satisfies the condition that if n is an ancestor of n' in either $\mathcal{T}_1$ or $\mathcal{T}_2$, then $n <_v n'$ (such an order can be shown to exist).

For the other direction, the main idea is to show that $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ ($\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$ can be shown by symmetric arguments, which implies $\mathcal{T}_1 \equiv \mathcal{T}_2$ by Lemma 5.4). Given $\mu \in [\![\mathcal{T}_1]\!]_G$ (for arbitrary G) the goal is therefore to show that there exists some subtree $\mathcal{R}_2$ of $\mathcal{T}_2$ that contains the same variables as $\mathcal{R}_1$ (the subtree of $\mathcal{T}_1$ representing the solution according to Lemma 3.24) and such that $\mu(\mathcal{R}_2) \subseteq G$. Towards this goal, $\mathcal{R}_2$ is claimed to be the minimal subtree S of $\mathcal{T}_2$ rooted at r_2 such that $\text{vars}(\mathcal{R}_1) \subseteq \text{vars}(S)$. The claim is proven by induction along a sequence $S_1, \ldots, S_m$ of subtrees where S_1 consists of r_2 only, $S_m = S$, and every S_{i+1} is S_i extended by one node. The proof must show that in every step the node added to S_i contains only variables from $\mathcal{R}_1$ and $\mu(S_{i+1}) \subseteq G$. Since $P_{r_1} = P_{r_2}$, the induction start is trivial. For the induction step, it is convenient to distinguish two cases.

The first case assumes that μ is defined on all variables in $\text{vars}(P_{n_{i+1}})$ (where n_{i+1} is the node added to S_i). In this case the idea is to show that the identity on all variables in $\text{vars}(P_{n+1})$ is a homomorphism from P_{n+1} into $P(\mathcal{R}_1) \cup P(S_i)$. This proves $P_{n+1} \subseteq P(\mathcal{R}_1) \cup P(S_i)$. Since $\mu(P(\mathcal{R}_1) \cup P(S_i)) \subseteq G$ (for S_i by the induction hypothesis and for $\mathcal{R}_1$ by assumption), this proves the case. The second case, that is, if μ is not defined on all variables in $\text{vars}(P_{n_{i+1}})$ can be shown to lead to a contradiction to μ being a solution. □

We want to point out that the requirement that both queries contain the same set of atoms is necessary, and does not follow from the strong homomorphical equivalence of all nodes that share "new" variables, as can be seen in the following example.

Example 5.11. Consider the following two QWDPTs.



Obviously these two QWDPTs are not equivalent, as they do not even contain the same set of variables. However, it can be easily checked that all required strong

homomorphisms exist: the homomorphism building the strong homomorphism is just the identity on all variables except for $?X_2$ and $?Y_2$ which are mapped to $?X_1$ and $?Y_1$, respectively.

Since testing the existence of strong homomorphisms is NP-hard, while computing the R3 normal form of a QWDPT requires a coNP test, Theorem 5.10 does not yet provide an NP algorithm for deciding equivalence. However, the following result shows that the coNP test can be avoided: instead of computing the R3 normal form, it suffices to apply rule R3 "often enough" so that the conditions of Theorem 5.10 are satisfied. Theorem 5.12 then guarantees that these properties still hold in R3 normal form, thus excluding the possibility of "false positives".

THEOREM 5.12. *Consider two QWDPTs* $\mathcal{T}_1 = ((V_1, E_1, r_1), \mathcal{P}_1)$ *and* $\mathcal{T}_2 = ((V_2, E_2, r_2), \mathcal{P}_2)$ *in NR normal form such that (1)* $\bigcup_{n \in V_1} P_n = \bigcup_{n \in V_2} P_n$*, (2)* $P_{r_1} = P_{r_2}$*, and (3) for all pairs* (n_1, n_2) *of nodes* $n_1 \in V_1$ *and* $n_2 \in V_2$ *with* $\text{newvars}(n_1) \cap \text{newvars}(n_2) \neq \emptyset$ *it holds that* $\text{branch}(n_1, \mathcal{T}_1)$ *and* $\text{branch}(n_2, \mathcal{T}_2)$ *are strongly homomorphically equivalent. Moreover, let* $\mathcal{T}_1^* = ((V_1^*, E_1^*, r_1^*), \mathcal{P}_1^*)$ *and* $\mathcal{T}_2^* = ((V_2^*, E_2^*, r_2^*), \mathcal{P}_2^*)$ *be R3 normal forms of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *respectively.*

Then the following conditions still hold for $\mathcal{T}_1^*$ *and* $\mathcal{T}_2^*$*: (1)* $\bigcup_{n \in V_1^*} P_n = \bigcup_{n \in V_2^*} P_n$*, (2)* $P_{r_1^*} = P_{r_2^*}$*, and (3) for all pairs* (n_1^*, n_2^*) *of nodes* $n_1^* \in V_1^*$ *and* $n_2^* \in V_2^*$ *with* $\text{newvars}(n_1^*) \cap \text{newvars}(n_2^*) \neq \emptyset$*, it holds that* $\text{branch}(n_1^*, \mathcal{T}_1^*)$ *and* $\text{branch}(n_2^*, \mathcal{T}_2^*)$ *are strongly homomorphically equivalent.*

PROOF IDEA. First of all, observe that the application of rule R3 leaves the set of triple patterns in a QWDPT unchanged. Thus property (1) still holds for $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$.

For the remaining two properties, we only give an intuition of the proof idea, and provide the complete proof in the electronic appendix to this article.

Since property (3) is useful for proving (2), it is convenient to discuss the proof of property (3) first. In order to do so, it suffices to show that for pairs (n_1, n_2) of nodes $n_1 \in V_1$ and $n_2 \in V_2$ that introduce the same variable, there still exist strong homomorphisms between the corresponding branches after applying any of Rule R1, R2, or R3. The existence of the different homomorphisms is shown by applying the following ideas. The original (strong) homomorphism may remain valid, the required homomorphism can be constructed by extending some homomorphism by the identity mapping, or the composition of two existing homomorphisms gives the required homomorphism. In case that none of these cases applies, the existence of further strong homomorphisms can be concluded that can then be used to construct the required mapping. The concrete proof is a very technical description of how these ideas can be applied in different cases to get the desired result. Therefore even a more detailed proof sketch is omitted.

Finally, in order to show that property (2) holds, first observe that obviously Rules R1 and R2 never change the root. Thus the only rule to consider is Rule R3. We distinguish three cases, based on the roots r_1^* and r_2^* (i.e., the roots in the assumed R3 normal forms). If both roots are ground, then it can be shown that either Rule R3 was never applicable to any child of the roots, or the child node contained a subset of the triple patterns in the root. In both cases, the roots remained unchanged, and the result follows from the assumption.

The second case is that one of r_1^* and r_2^* is ground, but the other is not. This can be shown to contradict the assumption that $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ are in R3 normal form. Both QWDPTs share the same set of variables. Hence the variable that occurs in the non-ground root (say in r_2^*) must occur somewhere in $\mathcal{T}_1^*$, say in some node n. Because of property (3), there exists a strong homomorphism $\text{branch}(n, \mathcal{T}_1^*) \to r_2^*$. It can be further shown that there exists a strong homomorphism from r_2^* to r_1^* (basically following from $P_{r_1} = P_{r_2}$ and the homomorphisms that allowed us to transform r_2 into r_2^* by merging

children of r_2 into r_2). However, the existence of these two strong homomorphisms can be shown to contradict the assumption that $\mathcal{T}_1^*$ is in R3 normal form, since their combination gives rise to a homomorphism that allows one to apply Rule R3 to n (in the following we refer to this property as property (i)).

The third case applies if both roots are assumed to be nonground. In this case, one can show that if $\text{newvars}(r_1^*) \not\subseteq \text{newvars}(r_2^*)$ this again gives a contradiction to $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ being in R3 normal form (the idea is similar to before: first identify appropriate strong homomorphisms, and then show the contradiction using property (i)). For $\text{newvars}(r_1^*) \subseteq \text{newvars}(r_2^*)$, because $\text{newvars}(r_1^*) = \text{vars}(r_1^*)$ it must be the case that $\text{vars}(r_1^*) \subseteq \text{vars}(r_2^*)$. Because there further exists a strong homomorphism $\mathcal{H}: r_1^* \to r_2^*$, it must be the identity on all variables in $\text{vars}(r_1^*)$, and therefore $P_{r_1^*} \subseteq P_{r_2^*}$. By symmetric arguments also $P_{r_2^*} \subseteq P_{r_1^*}$ follows. □

Given two QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$, the combination of Theorem 5.10 and Theorem 5.12 suggests the following algorithm for deciding whether $\mathcal{T}_1 \equiv \mathcal{T}_2$.

(1) Transform $\mathcal{T}_1$ and $\mathcal{T}_2$ into NR normal forms $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$, respectively.
(2) Guess two application sequences S_1 and S_2 of Rule R3 (i.e., nodes where to apply Rule R3 and the corresponding homomorphisms), one starting on $\mathcal{T}_1^*$ and one on $\mathcal{T}_2^*$, of length at most $|V_1^*|$, respectively, $|V_2^*|$.
(3) Check that the homomorphisms from step 2 fulfill the conditions of Rule R3. Denote the QWDPTs resulting from these R3 applications with $\mathcal{T}_1'$ and $\mathcal{T}_2'$.
(4) For any two nodes n_1 in $\mathcal{T}_1'$ and n_2 in $\mathcal{T}_2'$ with $\text{newvars}(n_1) \cap \text{newvars}(n_2) \neq \emptyset$, guess strong homomorphisms (i.e., collections of homomorphisms according to Definition 5.5) in both directions between $\text{branch}(n_1, \mathcal{T}_1')$ and $\text{branch}(n_2, \mathcal{T}_2')$.
(5) Check that $\mathcal{T}_1'$ and $\mathcal{T}_2'$ fulfill conditions (1)–(3) of Theorem 5.12.

By showing the correctness of this algorithm and providing a matching lower bound, this gives the main result of this section.

THEOREM 5.13. *The equivalence problem of QWDPTs (and, therefore, of well-designed SPARQL graph patterns) is* NP*-complete.*

PROOF. The NP-*membership* follows from the preceding algorithm. The proof of its correctness and runtime is given in the electronic appendix to this article.

The NP-hardness is shown by a straightforward reduction from the well-known NP-complete problem 3COL on graphs. Let $G = (V, E)$ be an arbitrary instance of 3COL. We define the following two sets of triple patterns.

—G_1: $\{(r, e, g), (g, e, r), (b, e, r), (r, e, b), (b, e, g), (g, e, b), (?V_1, ?V_2, ?V_3)\}$
—G_2: $\{(?X_i, e, ?X_j) \mid (v_i, v_j) \in E\}$

Using these patterns, we define the QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ as follows: Let $\mathcal{T}_1$ contain a single node r_1 and let $\mathcal{T}_2$ contain two nodes, that is, the root r_2 and a child node n_2 of the root r_2. We define $P_{r_1} = G_1 \cup G_2$, $P_{r_2} = G_1$, and $P_{n_2} = G_2$.

First of all it is easy to see that the reduction is feasible in polynomial time. The proof of its correctness is provided in the electronic appendix. □

In this section, we have thus proved that testing equivalence of QWDPTs, and therefore of well-designed SPARQL graph patterns, is in NP. We notice that our proof required a nontrivial use of the tree representation of SPARQL queries and the normal forms introduced in the previous section. One of the main contributions of this section is the introduction of the notion of strong homomorphisms which is the core notion in the NP equivalence test. We expect that just as homomorphisms are the fundamental tool for checking equivalence of CQs and extensions of CQs [Chandra and Merlin 1977;

Ullman 1997], strong homomorphisms can be used to design equivalence tests for more expressive fragments of SPARQL.

6. WELL-DESIGNED SPARQL GRAPH PATTERNS WITH PROJECTION

So far we only considered the case where solutions to QWDPTs (or, equivalently, well-designed SPARQL graph patterns) contain all variables for which a mapping could be found. In terms of CQs, this corresponds to the case where all variables in a CQ are free, that is, of CQs without projection. However, since projection is an interesting and important feature in every query language, in this section we reconsider all results achieved so far in the presence of projection. Towards this goal, after settling the semantics of projection we start by reconsidering the transformation rules and normal forms explored in Section 3.3. We then take a look on the evaluation and enumeration problem of QWDPTs in the presence of projection. Subsequently, we will study subsumption and equivalence for QWDPTs in the new setting.

Unlike in the previous sections, in the presence of projection there is a difference between bag- and set-semantics. Following the majority of research on CQs and their extensions, we also consider set-semantics and leave bag-semantics as future work.

6.1. Semantics and Notation

In SPARQL 1.0, projection is not part of the graph patterns, but is applied as a result modifier on top of the set of mappings returned by the SPARQL graph pattern. We take the same approach for our studies of projection on QWDPTs.

Definition 6.1 (*pQWDPT*). A *projected QWDPT (pQWDPT)* is a pair $(\mathcal{T}, \mathbf{X})$ where $\mathcal{T}$ is a QWDPT and $\mathbf{X} \subseteq \text{vars}(\mathcal{T})$ is a set of variables.

That is, in order to express projection we explicitly annotate a QWDPT with the set of variables onto which the result shall be projected.

The semantics of a pQWDPT is easily defined in terms of QWDPTs. For a mapping μ and a set of variables $\mathbf{X}$, let $\mu_{|\mathbf{X}}$ denote the mapping μ' defined as $\text{dom}(\mu') = \mathbf{X} \cap \text{dom}(\mu)$ and $\mu'(?X) = \mu(?X)$ for all $?X \in \text{dom}(\mu')$. That is, $\mu_{|\mathbf{X}}$ denotes the projection of μ to the variables in $\mathbf{X}$. Given a pQWDPT $(\mathcal{T}, \mathbf{X})$ and an RDF graph G, its semantics $[\![(\mathcal{T}, \mathbf{X})]\!]_G$ is defined as the projection of $[\![\mathcal{T}]\!]_G$ onto $\mathbf{X}$, formally defined as $[\![(\mathcal{T}, \mathbf{X})]\!]_G = \{\mu_{|\mathbf{X}} \mid \mu \in [\![\mathcal{T}]\!]_G\}$ [Pérez et al. 2006b]. In analogy to CQs, we call the variables in $\mathbf{X}$ the *free variables* ($\text{fvars}(\mathcal{T})$) and the variables in $\text{vars}(\mathcal{T}) \backslash \mathbf{X}$ the *existential variables* ($\text{evars}(\mathcal{T})$). Also, for a set P_n of triple patterns at some node n of $\mathcal{T}$, let $\text{fvars}(P_n)$ be the set of free variables occurring in P_n, that is, $\text{fvars}(P_n) = \text{vars}(P_n) \cap \text{fvars}(\mathcal{T})$. In analogy to the notion of $\text{newvars}(n)$ for some node n of $\mathcal{T}$, let $\text{newfvars}(n)$ denote the set $\text{newfvars}(n) = \text{fvars}(P_n) \backslash \text{vars}(P_{\text{branch}(\hat{n})})$ where $\hat{n}$ is the parent node of n. Analogously, $\text{newevars}(n) = \text{evars}(P_n) \backslash \text{vars}(P_{\text{branch}(\hat{n})})$.

For two QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$, we always assume that $\text{evars}(\mathcal{T}_1) \cap \text{evars}(\mathcal{T}_2) = \emptyset$. Note that this can be easily achieved by variable renaming.

6.2. Rules and Normal Forms

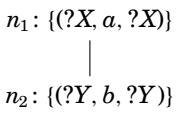
Concerning the evaluation of pQWDPTs, there is not much difference between allowing for projection or not. Especially the top-down evaluation presented in Section 3.2 is still applicable (the results retrieved from this evaluation must only be projected onto the free variables).

However, in the presence of projection there is some difference with respect to transformations as discussed in Section 3.3, and the properties we can show for the corresponding normal forms. Recall that in Section 3.3, we introduced a set of equivalence-preserving transformation rules for QWDPTs together with two normal forms of QWDPTs: in NR normal form, neither Rule R1 nor Rule R2 can be applied to

a QWDPT $\mathcal{T}$, while in R3 normal form in addition Rule R3 is not applicable as well. We now examine in how far those results carry over or need to be adapted when allowing in addition for projection.

First of all, Rules *R1*, *R3*, and *R4* are obviously still correct (i.e., equivalence preserving) also in the presence of projection. It is, however, easy to see that on several pQWDPTs the transformations described by Rule R2 could be performed safely although the formulation in Section 3.3 does not allow Rule R2 to be applied. Such a case is demonstrated by the following example.

Example 6.2. Consider the following pQWDPT $(\mathcal{T}, \mathbf{X})$ with $\mathbf{X} = \{?X\}$ and $\mathcal{T}$ as shown next.



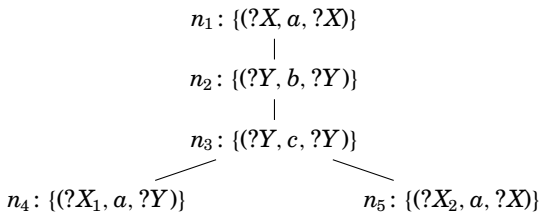
For the subtree $\mathcal{T}'$ of $\mathcal{T}$ consisting only of n_1, it obviously holds for every RDF graph G that $[\![(\mathcal{T}, \mathbf{X})]\!]_G = [\![(\mathcal{T}', \mathbf{X})]\!]_G$. Note that while n_2 introduces a new variable, this variable is neither part of the solution, nor has it any influence on the solutions.

This example shows that considering only nodes n with newvars$(n) = \emptyset$ is unnecessarily restrictive in the presence of projection. Instead, Rule R2 can be strengthened to Rule R2' as follows:

Rule R2' (*deletion of unproductive nodes in the presence of projection*): Let $n, \hat{n} \in V$ such that $\hat{n}$ is the parent of n, and let $n_1, \ldots, n_k \in V$ be the children of n. If newfvars$(n) = \emptyset$ and the resulting pattern tree is still quasi well designed, then merge n into each of its children and make each n_i a child of $\hat{n}$. That is, let $P'_{n_i} = P_{n_i} \cup P_n$ for $i = \{1, \ldots, k\}$, $V' = V \setminus \{n\}$, and $E' = (E \setminus \{(\hat{n}, n), (n, n_1), \ldots, (n, n_k)\}) \cup \{(\hat{n}, n_1), \ldots, (\hat{n}, n_k)\}$. If n has no child node, then applying this rule is equivalent to deleting n.

Note that while in the preceding rule it is sufficient if a node does not introduce any "new" free variables, unlike Rule R2 it has the additional restriction that it must not be applied if the result is not quasi well designed. This is because it might now be the case that R2' is applied to a node that introduces some existential variable. Merging this node into more than one child would lead to a pattern tree that is no longer quasi well designed, as depicted in the following example.

Example 6.3. Consider the following pQWDPT $(\mathcal{T}, \{?X, ?X_1, ?X_2\})$.



Then n_2 could be merged into n_3, and the resulting node could be merged into both n_4 and n_5 which would become direct children of n_1. However, then both of them would contain the variable $?Y$, which does not occur in n_1. Hence the result would not be quasi well designed.

Note that Example 6.3 also shows that a similar result as Proposition 3.23 does not hold for Rule R2'. Especially, applying R1 and R2' in arbitrary order does not lead to a unique result. In the previous example, there are two possibilities to apply R2': either

merge n_2 into n_3, or merge n_3 into both n_4 and n_5. In both cases, R2' cannot be applied subsequently, since as discussed the result would not be quasi well designed. Hence we get two different pQWDPTs to which neither R1 nor R2' can be applied.

We nevertheless apply the definitions of NR normal form and R3 normal form to pQWDPTs as well. That is, a pQWDPT $(\mathcal{T}, \mathbf{X})$ is in NR normal form if $\mathcal{T}$ is reduced with respect to Rules R1 and R2'. Furthermore, $(\mathcal{T}, \mathbf{X})$ is in R3 normal form if it is reduced with respect to Rules R1, R2', and R3. Note that just as in Section 3.3, both NR and R3 normal form of a pQWDPT can be computed by a linear number of rule applications of R1, R2', and R3.

A more detailed study of the properties of NR and R3 normal form of pQWDPTs in the line of the one presented in Section 3.3 for QWDPTs is left as future work. However, already these very basic considerations and definitions allow us to come up with a characterization of solutions to pQWDPTs similar to Lemma 3.24. Furthermore, they are also sufficient to extend our results on testing for subsumption and equivalence between QWDPTs to pQWDPTs. We start with providing the important characterizations of solutions to pQWDPTs.

LEMMA 6.4. *Let $(\mathcal{T}, \mathbf{X})$ be a pQWDPT in NR normal form, G an RDF graph, and μ a variable mapping for some subset of $\mathbf{X}$. Then $\mu \in [\![(\mathcal{T}, \mathbf{X})]\!]_G$ if and only if there exists a subtree $\mathcal{T}'$ of $\mathcal{T} = ((V, E, r), \mathcal{P})$ rooted at r such that:*

(1) $\mathrm{dom}(\mu) = \mathrm{fvars}(\mathcal{T}')$ *and*
(2) there exists a mapping λ on $\mathrm{evars}(\mathcal{T}')$ *such that* $\mu \cup \lambda \in [\![\mathcal{T}]\!]_G$.

PROOF. Follows immediately from the definition of $[\![(\mathcal{T}, \mathbf{X})]\!]_G$ and Lemma 3.24. □

This characterization provides a useful tool that will be used in the proofs of various complexity results presented next.

6.3. Evaluation in the Presence of Projection

For CQs without existential quantified variables, the decision problem corresponding to EVALUATION is tractable. However, it becomes NP-complete for CQs with existentially quantified variables. The next result shows that a similar behavior can be observed for well-designed SPARQL graph patterns as well. That is, the complexity increases by one level in the polynomial hierarchy if projection is added. Formally, we consider the following variant of EVALUATION.

Definition 6.5. Let EVALUATION^{proj} be the following decision problem.
INPUT: A pQWDPT $(\mathcal{T}, \mathbf{X})$, an RDF graph G, and a mapping μ on $\mathbf{X}$.
QUESTION: Is $\mu \in [\![(\mathcal{T}, \mathbf{X})]\!]_G$?

We next show that this problem is Σ_2^P-complete.

THEOREM 6.6. *The problem EVALUATION^{proj} is Σ_2^P-complete.*

PROOF SKETCH. *Membership* is shown by devising a simple "guess and check" algorithm that tests whether the solution candidate μ satisfies Lemma 6.4. Given a QWDPT $\mathcal{T}$, a mapping μ, a set $\mathbf{X}$ of free variables, and an RDF graph G, the witness (or certificate) that must be guessed by the algorithm consists of:

(1) the subtree $\mathcal{T}'$ of $\mathcal{T}$ rooted at r and
(2) the mapping λ on $\mathrm{evars}(\mathcal{T}')$.

For the "check" part, it remains to test whether $\mathrm{fvars}(\mathcal{T}') = \mathrm{dom}(\mu)$ and whether $\mu \cup \lambda \in [\![\mathcal{T}]\!]_G$. The first test can be obviously done in polynomial time, while the second test is in coNP [Pérez et al. 2009].

Hardness is shown by reduction from the problem Q-3COL$_{\exists,2}$, which is the following decision problem: given a graph $G = (V, E)$ together with a partition (V_1, V_2) of V, the question is whether there exists a 3-coloring ϕ_1 of V_1, such that for all possible 3-colorings ϕ_2 of V_2 the 3-coloring $\phi = \phi_1 \cup \phi_2$ is *no valid* 3-coloring of G. Please observe that in order to provide short names for the different concepts, with a 3-coloring we refer to an arbitrary assignment of the nodes to three colors (not necessarily assigning different colors to nodes connected by an edge). If in addition all adjacent nodes are assigned different colors, we call this a *valid* 3-coloring. The Σ_2^P-completeness of this problem follows immediately from Ajtai et al. [2000].

Hence let an arbitrary instance of this problem be given by a graph $C = (V_C, E_C)$ and a partition (V_1, V_2) of V_C. We construct an instance $((\mathcal{T}, \mathbf{X}), G, \mu)$ of EVALUATION^{proj}. Let $\mathcal{T} = ((V, E, r), \mathcal{P})$ with $V = \{r, n\}$ and $E = \{(r, n)\}$, and $\mathcal{P}, G, \mathbf{X}, \mu$ be defined as follows.

$$
\begin{aligned}
G &= \{(a,a,a),(1,e,2),(2,e,1),(1,e,3),(3,e,1),(2,e,3),(3,e,2),(1,c,1),(2,c,2),(3,c,3)\},\\
P_r &= \{(?Y_i, c, ?Y_i) \mid v_i \in V_1\},\\
P_n &= \{(?Y_i, c, ?Y_i) \mid v_i \in V_2\} \cup \{(?Y_i, e, ?Y_j) \mid (v_i, v_j) \in E\} \cup \{(?A, a, ?A)\},\\
\mathbf{X} &= \{?A\} \quad \text{and} \quad \mu = \mu_\emptyset \text{ (i.e., the empty mapping).}
\end{aligned}
$$

This reduction is obviously feasible in polynomial time. A formal proof of its correctness is provided in the electronic appendix. Next, only a short intuitive description of the idea of the proof is provided.

Towards the idea of the proof, first note that $\mu_\emptyset \in [\![(\mathcal{T}, \mathbf{X})]\!]_G$ iff there exists some mapping on vars(P_r) that cannot be extended to P_n. Now there is an obvious one-to-one correspondence between mappings μ on vars(P_r) and 3-colorings τ of V_1. There is further a one-to-one correspondence between mappings μ' on vars$(P_n) \setminus \{?A\}$ and colorings τ' of V_2. However, the combined mapping $\mu \cup \mu'$ only maps P_n into G if the corresponding coloring $\tau \cup \tau'$ is a valid 3-coloring of C: obviously, $\mu \cup \mu'$ must map variables $?Y_i$ and $?Y_j$ representing nodes $v_i, v_j \in V_C$ that are connected by an edge in E_C to different values in order to map the triple $(?Y_i, e, ?Y_j)$ into G. Hence, $\mu_\emptyset$ is only a solution if there exists some mapping μ on P_r such that it cannot be extended to a mapping μ' with $\mu'(P_n) \subseteq G$. By the relationship sketched before, this corresponds to the fact that there exists some 3-coloring on V_1 that cannot be extended to a coloring on V_2 that is a valid 3-coloring of C (since every such coloring would give rise to a corresponding extension μ' of μ). □

While for CQs with projection, deciding if a given tuple is part of the solution over some instance is NP-complete, the problem becomes tractable if the query is *acyclic*. In fact, ACQs are a rather restricted class of CQs for which this problem is tractable, since many other tractable classes aim at generalizing the set of ACQs. Hence in our setting the natural next question is how the complexity is changed if, given some QWDPT $\mathcal{T}$, we assume the sets of triple patterns at each node in $\mathcal{T}$ to be acyclic. The next result shows that while the problem becomes easier, it still remains intractable.

THEOREM 6.7. *Suppose that we only consider pQWDPTs built from QWDPTs where the sets of triple patterns at each node are from tractable fragments of CQ evaluation. Then the problem EVALUATION^{proj} is in* NP. *The problem remains* NP*-hard even if the sets of triple patterns at each node of the QWDPT are acyclic.*

PROOF. *Membership* is shown using the "guess and check" algorithm presented in the proof of Theorem 6.6. Just note that in the current setting also the second test is feasible in polynomial time, according to Corollary 4.1.

The *hardness* will follow immediately from the proof of Theorem 6.8 given shortly. Its discussion is therefore postponed. □

Note that the reason for EVALUATION^{proj} to become easier in this setting is the same as in Corollary 4.1. That is, testing whether the mapping $\mu \cup \lambda \in [\![\mathcal{T}]\!]_G$ (where μ is the solution, $\mathcal{T}'$ the guessed subtree, and λ the guessed mapping on evars($\mathcal{T}'$)) is in polynomial time. Hence the formulation "the sets of triple patterns at each node of the QWDPT are from tractable fragments of CQ evaluation" might be unnecessarily restrictive. In fact, it is only important that the second step (the "check" part) of the preceding algorithm can be decided in polynomial time. Possible relaxations of the formulation of the theorem that still guarantee that the check can be done in polynomial time have been already discussed in the paragraph following Corollary 4.1. The same discussion also applies in this case.

6.4. Enumeration in the Presence of Projection

We next extend our results on the enumeration problem to settings that also allow for projection. Similar as in the case of the decision problem, also the enumeration problem becomes harder in the presence of projection. While for the case without projection we showed that the existence of polynomial delay algorithms for the sets of triple patterns at each node of a QWDPT implies the existence of a polynomial delay algorithm for the QWDPT, the following result shows that for pQWDPTs, such an algorithm does not exist. In fact, it shows an even stronger result, namely that already in case that the sets of triple patterns at each node are acyclic, there does not even exist an output polynomial algorithm.

THEOREM 6.8. *Under the assumption that $P \neq$ NP, the problem ENUMERATION^{proj} cannot be solved in output polynomial time, even if the sets of triple patterns at each node of the input pQWDPT are acyclic.*

PROOF. The proof is by showing that if an output polynomial algorithm exists, then the problem 3COL could be solved in polynomial time. Towards this goal, let an arbitrary instance of 3COL be given by a graph $C = (V_C, E_C)$ where $V_C = \{v_1, \ldots, v_n\}$ and $E_C = \{e_1, \ldots, e_m\}$. Given C, construct an instance $((\mathcal{T}, \mathbf{X}), G)$ of ENUMERATION^{proj} with $\mathcal{T} = ((V, E, r), \mathcal{P})$ where $V = \{r, n_1, \ldots, n_m\}$ and $E = \{(r, n_1)\} \cup \{(n_{i-1}, n_i) \mid 2 \leq i \leq m\}$. Let further G, $\mathcal{P}$, and $\mathbf{X}$ be defined as

$$
\begin{aligned}
G &= \{(u, a, u), (1, c, 1), (2, c, 2), (3, c, 3), (1, e, 2), (2, e, 1), (1, e, 3), (3, e, 1), (2, e, 3), (3, e, 2)\},\\
P_r &= \{(?U, a, ?U)\},\\
P_{n_i} &= \{(?V_j, c, ?V_j) \mid v_j \in V_C\} \cup \{(?V_k, e, ?V_\ell)\}, \text{ for } 1 \leq i \leq m-1 \text{ and } e_i = (v_k, v_\ell),\\
P_{n_m} &= \{(?V_k, e, ?V_\ell), (?W, a, ?W)\} \text{ where } e_m = (v_k, v_\ell) \text{ and}\\
\mathbf{X} &= \{?U, ?W\}.
\end{aligned}
$$

Now $\mathcal{T}$ is obviously quasi well designed, and also the BGP at each node is obviously acyclic: for the nodes $n_1, \ldots, n_{m-1}$, just consider the join tree where all triple patterns $\{(?V_k, c, ?V_k) \mid v_k \in V_C\}$ are connected to the triple pattern $(?V_k, e, V_\ell)$. The BGPs P_r and P_{n_m} are trivially acyclic.

The idea of the proof is very similar to that of the proof of Theorem 6.6. Therefore only a short proof sketch is given. It is obvious that $[\![(\mathcal{T}, \mathbf{X})]\!]_G$ contains at most two mappings, namely $\mu_1 = \{?U \to a\}$ and $\mu_2 = \{?U \to a, ?W \to a\}$. Furthermore, the following relationships between C and $[\![(\mathcal{T}, \mathbf{X})]\!]_G$ hold.

—If there exists no valid 3-coloring for C, then $[\![(\mathcal{T}, \mathbf{X})]\!]_G = \{\mu_1\}$.
—If there exist valid 3-colorings for C but not all possible 3-colorings are valid, then $[\![(\mathcal{T}, \mathbf{X})]\!]_G = \{\mu_1, \mu_2\}$.
—If all possible 3-colorings of C are valid 3-colorings, then $[\![(\mathcal{T}, \mathbf{X})]\!]_G = \{\mu_2\}$.

The key observation towards those three properties is that every mapping μ with $\mu(P_{n_1}) \subseteq G$ encodes a 3-coloring on V_C, and every extension μ' of μ to the variable $?W$ with $\mu'(P_{\text{branch}(n_m)}) \subseteq G$ encodes a *valid* 3-coloring on C: note that $\mu'(P_{n_i}) \subseteq G$ iff the coloring encoded by μ' assigns different colors to the two nodes connected by edge $e_i \in E_C$.

In order to see that this indeed shows the nonexistence of an output polynomial enumeration algorithm for ENUMERATION^{proj} even in the case that the BGP at each node of the input pQWDPT is acyclic, assume to the contrary that such an algorithm exists. Since we have just shown that $|[\![(\mathcal{T}, \mathbf{X})]\!]_G| \leq 2$, this would imply that we can enumerate the solutions of $\mathcal{T}$ in (input) polynomial time. Hence we construct a PTIME decision procedure for 3COL by first enumerating the solutions of $\mathcal{T}$ over G and then checking whether $\mu_2 \in [\![(\mathcal{T}, \mathbf{X})]\!]_G$. Since this is obviously feasible in polynomial time, this gives the desired contradiction to $\mathsf{P} \neq \mathsf{NP}$. □

Note that from the previous proof the hardness of Theorem 6.7 follows immediately: consider the same reduction as shown in the preceding proof. Then we have just shown that $\mu_2 \in [\![(\mathcal{T}, \mathbf{X})]\!]_G$ iff C is a positive instance of 3COL.

6.5. The Subsumption and Equivalence Problem

Next, extending the results of Section 5 to projection, we study the problems of deciding subsumption and equivalence of pQWDPTs. Recall that all results in this section are stated for set-semantics and that the study of bag-semantics is left for future work. Note that in case of projection two pQWDPTs may return the same variable assignment on the free variables due to completely different mappings on the existential variables (and thus structures of the queries). Hence, intuitively, testing for subsumption and equivalence in the presence of projection requires one to consider more possibilities than without projection. Indeed, we show in this section that while the complexity of subsumption remains unchanged, the complexity of testing equivalence increases.

Before showing these results, we first clarify the notions of subsumption and equivalence for pQWDPTs. Given two pQWDPTs $(\mathcal{T}_1, \mathbf{X}_1)$ and $(\mathcal{T}_2, \mathbf{X}_2)$, $(\mathcal{T}_1, \mathbf{X}_1)$ is subsumed by $(\mathcal{T}_2, \mathbf{X}_2)$, denoted as usual by $(\mathcal{T}_1, \mathbf{X}_1) \sqsubseteq (\mathcal{T}_2, \mathbf{X}_2)$, if $[\![(\mathcal{T}_1, \mathbf{X}_1)]\!]_G \sqsubseteq [\![(\mathcal{T}_2, \mathbf{X}_2)]\!]_G$ holds for every RDF graph G. Similarly, the two pQWDPTs are equivalent, denoted by $(\mathcal{T}_1, \mathbf{X}_1) \equiv (\mathcal{T}_2, \mathbf{X}_2)$, if $[\![(\mathcal{T}_1, \mathbf{X}_1)]\!]_G = [\![(\mathcal{T}_2, \mathbf{X}_2)]\!]_G$ holds for every RDF graph G.

Our first result shows that deciding subsumption for pQWDPTs is similar to deciding subsumption for QWDPTs. In fact, the following lemma—characterizing subsumption between pQWDPTs—is a generalization of Lemma 5.2, since basically property (2) requires the homomorphism to be the identity on the shared variables.

LEMMA 6.9. *Consider pQWDPTs $(\mathcal{T}_1, \mathbf{X}_1)$ and $(\mathcal{T}_2, \mathbf{X}_2)$ with roots r_1 and r_2, respectively. Then $(\mathcal{T}_1, \mathbf{X}_1) \sqsubseteq (\mathcal{T}_2, \mathbf{X}_2)$ if and only if $\mathbf{X}_1 \subseteq \mathbf{X}_2$ and for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ rooted at r_1, there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$ rooted at r_2 such that:*

(*1*) $\text{vars}(\mathcal{T}_1') \cap \mathbf{X}_1 \subseteq \text{vars}(\mathcal{T}_2')$, *and*
(*2*) *there exists a homomorphism from the triples in $\mathcal{T}_2'$ to the triples in $\mathcal{T}_1'$ that is the identity over* $\text{vars}(\mathcal{T}_1') \cap \mathbf{X}_1$.

The proof of this lemma is similar to that of Lemma 5.2, providing the characterization for the case without projection. The proof can therefore be found in the electronic appendix of this article. Note that this result, just as Lemma 5.2 for QWDPTs, gives an immediate Π_2^P algorithm for testing subsumption between two pQWDPTs.

Hence, for the subsumption problem, the complexity does not change when going from QWDPTs to pQWDPTs. The next result shows that testing for equivalence becomes harder in the presence of projection than it is without.

LEMMA 6.10. *Let* $(\mathcal{T}_1, \mathbf{X}_1)$ *and* $(\mathcal{T}_2, \mathbf{X}_2)$ *be two pQWDPTs. Deciding whether* $(\mathcal{T}_1, \mathbf{X}_1) \equiv (\mathcal{T}_2, \mathbf{X}_2)$ *is* Π_2^P*-hard.*

The proof of this result is provided completely in the electronic appendix to this work.

Recall that for the case without projection, Lemma 5.4 established a close connection between subsumption and equivalence (by showing $\mathcal{T}_1 \equiv \mathcal{T}_2$ iff $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ and $\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$ holds for QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$), thus giving an immediate Π_2^P algorithm for testing equivalence. As the following example shows, this relationship does no longer hold in the presence of projection.

Example 6.11. Consider the following two pQWDPTs $(\mathcal{T}_1, \mathbf{X})$ and $(\mathcal{T}_2, \mathbf{X})$ with

$\mathcal{T}_1$: $n_1 : \{(?X_1, a, ?Y_1), (?Y_2, a, ?Y_3)\}$ with child $n_2' : \{(?X_2, a, ?Y_3), (?X_2, a, ?X_2), (?Y_2, a, ?Y_2)\}$

$\mathcal{T}_2$: $n_1' : \{(?X_1, a, ?Y_1')\}$ with child $n_2' : \{(?X_2, a, ?Y_3'), (?X_2, a, ?X_2)\}$

and let $\mathbf{X} = \{?X_1, ?X_2\}$. Then $(\mathcal{T}_1, \mathbf{X}) \sqsubseteq (\mathcal{T}_2, \mathbf{X})$ and $(\mathcal{T}_2, \mathbf{X}) \sqsubseteq (\mathcal{T}_1, \mathbf{X})$, as can be checked using Lemma 6.9: the checks for $(\mathcal{T}_1, \mathbf{X}) \sqsubseteq (\mathcal{T}_2, \mathbf{X})$ as well as the check for the subtree of $\mathcal{T}_2$ consisting of n_1' only for testing $(\mathcal{T}_2, \mathbf{X}) \sqsubseteq (\mathcal{T}_1, \mathbf{X})$ are rather easy to verify. For the remaining case of testing the complete tree $\mathcal{T}_2$, consider the following mapping from vars($\mathcal{T}_1$) to vars($\mathcal{T}_2$): first of all, $?X_1 \to ?X_1$ and $?X_2 \to ?X_2$ is required. Furthermore, take $?Y_1 \to ?Y_1'$, $?Y_2 \to ?X_2$, and $?Y_3 \to ?Y_3'$. It can be checked that this indeed maps all triples in $\mathcal{T}_1$ into $\mathcal{T}_2$, and therefore proves $(\mathcal{T}_2, \mathbf{X}) \sqsubseteq (\mathcal{T}_1, \mathbf{X})$. However, for the RDF graph $G = \{(1, a, 1), (1, a, 0), (2, a, 3)\}$, the mapping μ defined as $\mu := \{?X_1 \to 1, ?Y_1 \to 0, ?Y_2 \to 2, ?Y_3 \to 3\}$ gives rise to the solution $\mu_{|\mathbf{X}} = \{?X_1 \to 1\}$ of $(\mathcal{T}_1, \mathbf{X})$: just note that μ cannot be extended to n_2 since $(\mu(?Y_2), a, \mu(?Y_2)) = (2, a, 2) \notin G$. However, $\lambda = \{?X_1 \to 1\} \notin [\![(\mathcal{T}_2, \mathbf{X})]\!]_G$ since every extension λ' of λ to $?Y_1'$ with $(\lambda'(?X_1), a, \lambda'(?Y_1')) \subseteq G$ can be extended to a mapping λ'' such that $\lambda''(P_{n_2'}) \subseteq G$.

As a result, for pQWDPTs the Π_2^P-membership of subsumption does not give any upper bound on the complexity of equivalence, and we have to leave the exact complexity of deciding equivalence as an open problem. An upper bound for this problem is not yet known.

We conclude by observing that the "theme" of problems becoming more difficult in the presence of projection compared to the setting without projection showed up for almost every problem studied in this section. The subsumption problem is a notable exception in this respect.

7. CONCLUDING REMARKS

Static analysis is a fundamental task in query optimization. In this article we have initiated the study of this problem for SPARQL queries by concentrating on the fragment of *well-designed* SPARQL queries. One of our main contributions is the introduction of an abstract representation of well-designed queries as trees. This representation allowed us to provide transformation rules and normal forms that proved useful when studying equivalence, containment, and tractable query enumeration. Table I provides an overview on the corresponding complexity results presented in this article. It can be observed that allowing for projection on top of well-designed SPARQL graph patterns increases the complexity of almost all of the problems studied. Our results further show that the huge amount of results on efficient CQ evaluation can be fruitfully applied to the evaluation of the conjunctive parts of well-designed SPARQL graph patterns to reduce the complexity of several of the problems studied. Beside settling the exact complexity of the equivalence problem in presence of projection, an interesting line for future work is the inclusion of further SPARQL operators to our study, in particular,

Table I. Overview on Complexity Results

	$\sqsubseteq$	$\equiv$	eval. (tract. CQ)	enum. for tract. CQ
no projection	Π_2^P	NP	coNP* (in P)	polynomial delay
with projection	Π_2^P	Π_2^P-hard	Σ_2^P (NP)	not output polynomial

Complexity of the problems on well-designed SPARQL graph patterns studied in this article: subsumption ($\sqsubseteq$), equivalence ($\equiv$), evaluation (eval.), and enumeration (enum.). "tract. CQ" denotes the settings where all sets of triple patterns of the input (p)QWDPT are assumed to be from tractable classes of CQ evaluation. If not stated otherwise, the problems are complete for the given complexity classes.
* [Pérez et al. 2009].

to consider filtering, union, and the new language features of SPARQL 1.1, such as difference, nested projection, and aggregate functions.

On the practical side, in Letelier et al. [2012a], we combined the techniques and algorithms presented in this article (both those for manipulating pattern trees and those for testing equivalence and subsumption) in a tool that helps a user to analyze (thus to better understand), modify, and manually optimize her SPARQL queries.

The next step will be to create a competitive SPARQL engine based on pattern trees, transformations on them, and the top-down evaluation.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

Abadi, D. J., Marcus, A., Madden, S., and Hollenbach, K. J. 2007. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*. ACM Press, New York, 411–422.

Ajtai, M., Fagin, R., and Stockmeyer, L. J. 2000. The closure of monadic np. *J. Comput. Syst. Sci. 60*, 3, 660–716.

Angles, R. and Gutierrez, C. 2008. The expressive power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference (ISWC'08)*. Lecture Notes in Computer Science, vol. 5318, Springer, 114–129.

Arenas, M. and Pérez, J. 2011. Querying semantic web data with SPARQL. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'11)*. ACM Press, New York, 305–316.

Bagan, G., Durand, A., and Grandjean, E. 2007. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of the 21st International Workshop on Computer Science Logic (CSL'07)*. Lecture Notes in Computer Science, vol. 4646, Springer, 208–222.

Berners-Lee, T. 2006. Linked data – Design issues. http://www.w3.org/DesignIssues/LinkedData.html.

Bizer, C., Heath, T., and Berners-Lee, T. 2009. Linked data - The story so far. *Int. J. Semantic Web Inf. Syst. 5*, 3, 1–22.

Chandra, A. K. and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77)*. ACM Press, New York, 77–90.

Chekol, M. W., Euzenat, J., Geneves, P., and Layaida, N. 2012. SPARQL query containment under shi axioms. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*. AAAI Press.

Chekuri, C. and Rajaraman, A. 2000. Conjunctive query containment revisited. *Theor. Comput. Sci. 239*, 2, 211–229.

Cohen, S., Fadida, I., Kanza, Y., Kimelfeld, B., and Sagiv, Y. 2006. Full disjunctions: Polynomial-delay iterators in action. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06)*. ACM Press, New York, 739–750.

DBPedia. 2012. DBPedia.org. http://DBpedia.org/sparql.

Flum, J., Frick, M., and Grohe, M. 2002. Query evaluation via tree-decompositions. *J. ACM 49,* 6, 716–752.

Gallego, M. A., Fernandez, J. D., Martinez-Prieto, M. A., and de la Fuente, P. 2011. An empirical study of real-world SPARQL queries. In *Proceedings of the 1st International Workshop on Usage Analysis and the Web of Data (USEWOD'11)*.

GOTTLOB, G., LEONE, N., AND SCARCELLO, F. 2000. A comparison of structural CSP decomposition methods. *Artif. Intell. 124*, 2, 243–282.

GOTTLOB, G., LEONE, N., AND SCARCELLO, F. 2002. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci. 64*, 3, 579–627.

GRECO, G. AND SCARCELLO, F. 2010a. The power of tree projections: Local consistency, greedy algorithms, and larger islands of tractability. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'10)*. ACM Press, New York, 327–338.

GRECO, G. AND SCARCELLO, F. 2010b. Structural tractability of enumerating CSP solutions. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP'10)*. Lecture Notes in Computer Science, vol. 6308, Springer, 236–251.

GUTIERREZ, C., HURTADO, C. A., MENDELZON, A. O., AND PÉREZ, J. 2011. Foundations of semantic web databases. *J. Comput. Syst. Sci. 77,* 3, 520–541.

HM GOVERNMENT. 2012. data.gov.uk. http://data.gov.uk.

JOHNSON, D. S., PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. 1988. On generating all maximal independent sets. *Inf. Process. Lett. 27,* 3, 119–123.

KANZA, Y., NUTT, W., AND SAGIV, Y. 2002. Querying incomplete information in semi-structured data. *J. Comput. Syst. Sci. 64,* 3, 655–693.

LARSON, P.-A. AND ZHOU, J. 2005. View matching for outer-join views. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*. ACM Press, New York, 445–456.

LASSILA, O. AND SWICK, R. R. 1999. Resource description framework (RDF) model and syntax. W3C Recommendation. http://www.w3.org/TR/PR-rdf-syntax.

LETELIER, A., PÉREZ, J., PICHLER, R., AND SKRITEK, S. 2012a. SPAM: A SPARQL analysis and manipulation tool. *Proc. VLDB Endow. 5,* 12, 1958–1961.

LETELIER, A., PÉREZ, J., PICHLER, R., AND SKRITEK, S. 2012b. Static analysis and optimization of semantic web queries. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'12)*. ACM Press, New York, 89–100.

MALLEA, A., ARENAS, M., HOGAN, A., AND POLLERES, A. 2011. On blank nodes. In *Proceedings of the International Semantic Web Conference.* 421–437.

NEUMANN, T. AND WEIKUM, G. 2010. The RDF-3x engine for scalable management of RDF data. *Int. J. VLDB 19,* 1, 91–113.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2006a. Semantics and complexity of SPARQL. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06).* Lecture Notes in Computer Science, vol. 4273, Springer, 30–43.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2006b. Semantics of SPARQL. Tech. rep. TR/DCC-2006-17, Universidad de Chile. http://users.dcc.uchile.cl/~jperez/papers/sparql.semantics.pdf.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2009. Semantics and complexity of SPARQL. *ACM Trans. Datab. Syst. 34,* 3.

PICALAUSA, F. AND VANSUMMEREN, S. 2011. What are real sparql queries like? In *Proceedings of the International Workshop on Semantic Web Information Management (SWIM'11)*. ACM Press, New York, 7.

POLLERES, A. 2007. From sparql to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. ACM Press, New York, 787–796.

PRUDHOMMEAUX, E. AND SEABORNE, A. 2008. SPARQL query language for RDF. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/.

SCHMIDT, M., HORNUNG, T., KUCHLIN, N., LAUSEN, G., AND PINKEL, C. 2008. An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario. In *Proceedings of the 7th International Semantic Web Conference (ISWC'08).* Lecture Notes in Computer Science, vol. 5318, Springer, 82–97.

SCHMIDT, M., MEIER, M., AND LAUSEN, G. 2010. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory (ICDT'10).* ACM Press, New York, 4–33.

SERFIOTIS, G., KOFFINA, I., CHRISTOPHIDES, V., AND TANNEN, V. 2005. Containment and minimization of RDF/s query patterns. In *Proceedings of the 4th International Semantic Web Conference (ISWC'05).* Lecture Notes in Computer Science, vol. 3729, Springer, 607–623.

SIDIROURGOS, L., GONCALVES, R., KERSTEN, M. L., NES, N., AND MANEGOLD, S. 2008. Columnstore support for RDF data management: Not all swans are white. *Proc. VLDB Endow. 1,* 2, 1553–1563.

STOCKER, M., SEABORNE, A., BERNSTEIN, A., KIEFER, C., AND REYNOLDS, D. 2008. SPARQL basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. ACM Press, New York, 595–604.

ULLMAN, J. D. 1997. Information integration using logical views. In *Proceedings of the 6th International Conference on Database Theory (ICDT'97).* Lecture Notes in Computer Science, vol. 1186, Springer, 19–40.

US GOVERNMENT. 2012. data.gov. http://www.data.gov.

WEISS, C., KARRAS, P., AND BERNSTEIN, A. 2008. Hexastore: Sextuple indexing for semantic web data management. *Proc. VLDB Endow. 1,* 1, 1008–1019.

YANNAKAKIS, M. 1981. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB'81).* 82–94.

Received October 2012; revised April 2013; accepted June 2013