FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Complexity of well-designed SPARQL

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Alexander Svozil, Bsc.
Matrikelnummer 1026213

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Sebastian Skritek
Univ.Ass. Dipl.-Ing. Markus Kröll

Wien, 13. Juni 2016

Alexander Svozil

Reinhard Pichler

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Complexity of well-designed SPARQL

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Alexander Svozil, Bsc.
Registration Number 1026213

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Assistance: Univ.Ass. Dipl.-Ing. Dr.techn. Sebastian Skritek
Univ.Ass. Dipl.-Ing. Markus Kröll

Vienna, 13th June, 2016

Alexander Svozil | Reinhard Pichler

Erklärung zur Verfassung der Arbeit

Alexander Svozil, Bsc.
Wasnergasse 13/20, Wien 1200

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. Juni 2016

Alexander Svozil

Acknowledgements

First of all I want to thank my advisor, Prof. Dr. Reinhard Pichler not only for his guidance, support and patience but also for opening up this topic for me. I would also like to thank him for introducing me to his team at the database institute; in particular, Dr. Sebastian Skritek and Dipl.-Ing. Markus Kröll. I would like to thank them all for letting me participate in their discourse and their valuable input. In particular I would like to thank Dr. Sebastian Skritek as he spent a great amount of time discussing and improving the proofs in the thesis with me.

Last I would like to thank my family and my friends. My parents Karl and Karin for not only making my academic studies possible but also for encouraging me to pursue my goals. I would like to especially thank my dad for always sharing a cup of coffee with me when I needed it the most. I would also like to thank my friends Mihail and Chris for making studying at the TU Wien one of the best experiences in my life.

Kurzfassung

SPARQL Protocol and RDF Query Language (SPARQL) ist eine standardisierte Möglichkeit um Resource Description Framework (RDF) Daten aus dem Internet abzufragen. Ein Feature von SPARQL 1.0 ist der GRAPH Operator, der es erlaubt, mehrere lokale RDF Graphen in einer einzigen SPARQL Query zu verwenden. Mit der Zeit tauchten immer mehr SPARQL Schnittstellen im Internet auf und um jene auch ansprechen zu können, wurde der SERVICE Operator in der SPARQL 1.1 Federated Query extension eingeführt. Mit dem SERVICE Operator kann man, ähnlich dem GRAPH Operator, mehrere SPARQL Endpoints in einer Query benützen. Sowohl der GRAPH Operator als auch der SERVICE Operator werfen neue Probleme in der Komplexitätsanalyse von SPARQL auf. Die beiden Operatoren wurden bis jetzt noch nicht analysiert. Weil die Evaluierung von allgemeinen SPARQL Queries PSPACE-complete ist, beschränken wir unsere Beobachtungen auf well-designed SPARQL, wo das Evaluierungsproblem coNP-complete ist. Wenn man das well-designed SPARQL Fragment mit SERVICE und GRAPH erweitert, erhält man ein neues SPARQL Fragment welches eine gute Basis für unsere Komplexitätsanalyse darstellt. Wenn man allgemeines SPARQL um die beiden Operatoren erweitern würde, könnte es sein, dass die Komplexität der beiden Operatoren von diesem Fragment überschattet würden. Wir werden zeigen, dass die Komplexität des Evaluierungsproblems im well-designed Fragment, welches mit den Operatoren GRAPH und SERVICE erweitert wurde, coNP-complete ist. Wenn man den SERVICE operator in der Praxis benutzt, löst dieser neue Schwierigkeiten aus, die mit der Einführung mehrerer Notationen gelöst wurden. Der Unterschied dieser Notationen wurde bis jetzt noch nicht wirklich erforscht. Nachdem wir die Notationen eingeführt haben, werden wir sie diskutieren. Wir werden auch das Fragment weakly well-designed SPARQL behandeln. Es ist ein auf well-designed SPARQL basierendes Fragment, welches allerdings mächtiger ist, weil es einige Bedingungen des well-designed SPARQL Fragments schwächt.

Abstract

The SPARQL Protocol and RDF Query Language (SPARQL) presents a standardized way to query the growing amount of Resource Description Framework (RDF) data on the internet. A feature of SPARQL 1.0 is the GRAPH operator which is able to query multiple local RDF graphs in only a single query. To access the growing amounts of SPARQL endpoints available on the internet, the SERVICE operator was introduced in the SPARQL 1.1 Federated Query extension. The SERVICE operator is used to query several SPARQL endpoints in only a single query, similar to the GRAPH operator. Both the SERVICE and the GRAPH operator pose new problems in the complexity analysis of SPARQL, as the two operators have not yet been analyzed. Because the evaluation of general SPARQL patterns is PSPACE-complete we restrict our considerations to well-designed SPARQL, where the evaluation problem is coNP-complete. Extending the well-designed fragment of SPARQL with the SERVICE and GRAPH operators yields a new SPARQL fragment which is a good basis for a complexity analysis, as general SPARQL would cloud the complexity of the SERVICE and GRAPH operators. It is shown that the evaluation problem in this fragment is coNP-complete. Using the SERVICE operator in practice elicits difficulties which were resolved by introducing various notions. The subtle difference between those notions has not yet been fully explored. After defining the notions the differences between them will be discussed. Building upon well-designed SPARQL we will also cover weakly well-designed SPARQL which renders a powerful fragment by relaxing constraints of well-designed SPARQL.

Contents

CHAPTER 1

Introduction

Within the last years an increasing number of Resource Description Framework (RDF) [LS98] data has been published on the web. RDF is an integral part of the semantic web concept [BLHL+01] because one can express relationships between resources similar to class diagrams. Like basic grammar, RDF allows to make statements in the form of subject-predicate-object expressions. These expressions are called "triples" in RDF terminology and are stored in so called RDF graphs, the RDF databases. One of the most cited and important publisher of RDF data is DBPedia, a large-scale, multilingual knowledge base extracted from Wikipedia [LIJ+15]. Other important initiatives which use RDF as a core technology are Open Linked Data [CTL, BL] and Open Government Data [data, datb]. Several research efforts have been pursued towards understanding RDF and developing specific techniques to efficiently use it in practice [AMMH07, WKB08, SGK+08, SHK+08, NW10, GHM04, HAMP14].

The standardized way to query RDF data on the internet is the SPARQL Protocol And RDF Query Language (SPARQL). SPARQL is standardized by the RDF Data Access Working Group of the World Wide Web Consortium (W3C) [SHa] and is a key technology for the semantic web which can be witnessed by the active development of SPARQL query engines. Some examples for these systems are AllegroGraph [all], Apache Jena [jen], Sesame[ses] and OpenLink Virtuoso [vir].

A peculiarity of SPARQL is the optional matching feature which allows to construct queries where can specify some part of the queried data which is mandatory to be returned and some optional data which is optionally returned if it is available. An example would be a phone book where some people decide to only to put their name and telephone number but not all of them put their address. If we want to query for maximum information in this database, we could ask for the name, the telephone number and put the address into the optional part. When only the name and telephone number of a person are specified our query will not fail to give an answer but just return the provided (incomplete) information. This feature is sought-after when querying data in the

semantic web because everybody has their own policy on how to publish their data. A lot of work has already been done studying SPARQL and especially regarding the complexity of the optional matching feature [AG08, PAG06, PAG09, SML10, AP11, KK16, KG14]. Two of the main results of this analysis are, that evaluating general SPARQL is PSPACE-complete [PAG06] and equally expressive as relational calculus [AG08, Pol07]. Thus a new fragment where the evaluation problem is only coNP-complete called well-designed SPARQL [PAG09] was established. It was also found out that the SPARQL query containment and query equivalence problems are undecidable in the general SPARQL fragment. Static query analysis of SPARQL, i.e., query containment and query equivalence is used to optimize queries. Optimizing a query can be done by replacing a given query by an equivalent one which possesses superior computational properties. Obviously this requires a procedure to decide whether two queries are equivalent. Several papers have been published regarding query containment and query equivalence for intrinsic fragments of SPARQL and optimization [SKCT05, WEGL12, LPPS12, LPPS13, SML10, CEGL11, SSB+08, BKPR14]. The most thorough analysis of the containment and equivalence problems over well-designed SPARQL was done in [PS14].

The GRAPH operator, which is a feature of SPARQL 1.0, can be used to query multiple local RDF graphs in a single query. Lately the number of SPARQL endpoints on the internet grew and with this growth the need for federated SPARQL queries also rose. This led to the introduction of the SPARQL 1.1 Federated Query extension [AS]. The Federated Query Extension extension brought up the SERVICE operator. Similar to the GRAPH operator, the SERVICE operator is used to query multiple SPARQL endpoints on the internet in a single query. A big effort has already been put into analyzing the operators, benchmarking implementations of federated query processing and querying distributed RDF data sources with SPARQL in general [BAACP13, SHH+11, QL08, BAP14, MSMMV15, SNP+13, AVL+11, GS11, HS12, BAPU14] but the complexity analysis of the GRAPH and SERVICE operators remains an open question.

Using the SERVICE operator in practice elicits a lot of new difficulties. As described the SERVICE operator allows to evaluate a query over multiple endpoints in the web in just a single query. An example for a SPARQL graph pattern using the SERVICE operator would be $Q = (\text{SERVICE } u\ P_1)$. The symbol u could be a variable or a Uniform Resource Identifier (URI), which is a string used to identify a resource. The symbol P_1 stands for another SPARQL query. The SERVICE operator is supposed to evaluate the query P_1 over the SPARQL endpoint defined by u. If u is a URI, the endpoint the query P_1 gets evaluated over is the one identified by the URI u. If u is a variable the semantic of the query Q is not defined by the W3C standard [SHb]. Obviously the query cannot be evaluated over all the possible endpoints on the internet, so we need to bind the variable to finitely many URIs. In [BAACP13] two notations called boundedness and strong boundedness are used to cope with this problem. Deciding boundedness of a variable in a query is undecidable whereas deciding strong boundedness of a variable in a query is doable in polynomial time. Even though these two notions have been brought up for practical use, the difference of the two remains unknown.

We will analyze the evaluation problem of SPARQL with the SERVICE and GRAPH operator but we will restrict ourselves to well-designed SPARQL. The reason for this restriction is that we don't want the complexity of full SPARQL to cloud the complexity of the SERVICE and GRAPH operators. We will thus extend the well-designed SPARQL fragment to a new fragment which additionally allows the free use of the SERVICE and GRAPH operator. Then we will analyze the complexity of the evaluation problem in this fragment. We will also discuss the difference between boundedness and strong boundedness and provided an example of a query where a variable is bounded but not strongly bounded. We will also conjecture how a procedure to decide boundedness could look like. Even though it was shown in [BAACP13] that deciding if a variable is bounded in a pattern is undecidable for general SPARQL it might be decidable for fragments of SPARQL where the query equivalence problem is not undecidable.

While writing the thesis another interesting fragment was published, namely weakly well-designed SPARQL, which further extends well-designed SPARQL by relaxing several restrictions without increasing the complexity [KK16]. We will also look into weakly well-designed SPARQL.

After the introduction, Chapter 2 provides the most important basic definitions (like conjunctive queries, RDF and SPARQL). In the second part of Chapter 2 we will discuss the peculiarities of the semantics of the SERVICE operator. In Chapter 3 well-designed SPARQL is introduced and we look into the results of [PS14]. The proofs of the results in [PS14] provide useful methods for proving the complexity results with GRAPH and SERVICE. Following up in Chapter 4 we analyse the evaluation problem of well-designed SPARQL which includes the GRAPH and SERVICE operator. In Chapter 5 we study how SERVICE queries could be evaluated in practice. The second part of Chapter 5 discusses the difference of boundedness and strong boundedness. In Chapter 6 we will look at weakly well-designed SPARQL which was introduced in [KK16].

CHAPTER 2

Preliminaries

We will introduce conjunctive queries first and then proceed with RDF and SPARQL. After defining these three concepts we will discuss the semantics of the GRAPH and SERVICE operator in SPARQL.

2.1 Conjunctive Queries, RDF, SPARQL

We will start out with the definition of a relational schema, database and conjunctive query.

A relational name describes the name and arity of a relation.

Definition 1 (Relational Schema)**.** *A relational schema is a nonempty finite set of relational names.*

Following up we have the definition of a database and relational atom over a schema..

Definition 2 (Database and relational atom)**.** *Let σ be a relational schema,* **U** *be an infinite set of constants and* **V** *an infinite set of variables. A* relational *atom over σ is an expression of the form $R(v)$ where R is a relational name with arity n over σ and v is an n-tuple over* $\mathbf{U} \cup \mathbf{V}$*.*

A database *D over σ is a set of relational atoms over σ and each n-tuple is in* **U***.*

After defining database and schema we are ready to define conjunctive queries.

Definition 3 (Conjunctive Queries)**.** *Let σ be a relational schema. A conjunctive query (CQ) q over σ is a rule of the form:*

$$X \leftarrow R_1(\vec{v_1}), \ldots, R_m(\vec{v_m}).$$

Each $R_i(\vec{v_i})(1 \leq i \leq m)$ is a relational atom over σ and X is a subset of the set of variables that appear in the $\vec{v_i}$s.

In order to evaluate a CQ given a database over a relational schema, we need to define the semantics which is given in terms of homomorphisms.

Definition 4 (Semantics of Conjunctive Queries)**.** *Let D be a database over schema σ. A homomorphism from a CQ Q to D is a partial mapping $h : X \to U$ such that $R_i(h(\vec{v_i})) \in D$, for $1 \leq i \leq m$. $h_{|X}$ is the restriction of h to the variables in X. The evaluation $Q(D)$ is the set of all mappings of the form $h_{|X}$, such that h is a homomorphism from q to D.*

We will now define a simplified but absolutely (for our purposes) sufficient version of the original formalisation of the RDF W3C-recommendation [LS98]: URI stands for Uniform Resource Identifier and is a string which is used to identify abstract and physical resources in the web. We will focus on ground RDF graphs which means that we assume them to be composed of URIs only. Usually RDF graphs would also support the usage of blank nodes, which work like existential variables (see [HAMP14] for details).

Definition 5 (RDF)**.** *Let $\mathbf{U}$ is the infinite set of URIs. An RDF triple is a tuple in $\mathbf{U} \times \mathbf{U} \times \mathbf{U}$, whereas an RDF graph is a finite set of RDF triples. Let G be an RDF graph. Then $dom(G) \subseteq \mathbf{U}$ is the set of URIs actually appearing in G.*

RDF is the data format and SPARQL will be our query language. To use SPARQL we need to define the SPARQL syntax and semantics.

Definition 6 (SPARQL Syntax [BAACP13, PAG06])**.** *Again $\mathbf{U}$ is an infinite set of URIs. $\mathbf{V}$ is an infinite set of variables with $\mathbf{U} \cap \mathbf{V} = \emptyset$. We will denote Variables in $\mathbf{V}$ with the letters $x, y, z, x', y', z', \ldots$ and symbols which are in $\mathbf{V} \cup \mathbf{U}$ with $u, v, w, u', v', w', \ldots$ and symbols which are in $\mathbf{U}$ with $a, b, c, d, e, f, a', b', c', d', e', f', \ldots$.*

Filter constraints are conditions of the following form:

1. *$\top, x = a, x = y$, or $bound(x)$ (atomic constraints),*
2. *If R_1, R_2 are filter constraints, then $\neg R_1$, $R_1 \wedge R_2$, or $R_1 \vee R_2$ are filter constraints.*

A SPARQL triple pattern is a tuple in $(\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V}) \times (\mathbf{U} \cup \mathbf{V})$. SPARQL graph patterns are recursively defined as follows:

1. *A triple pattern is a graph pattern.*
2. *If P_1 and P_2 are graph patterns, then $(P_1 \circ P_2)$ for $\circ \in \{AND, OPT, UNION\}$ is a graph pattern.*

3. *If P is a graph pattern and R a filter constraint, then $(P\ FILTER\ R)$ is a graph pattern.*

4. *If P is a graph pattern and $u \in (\mathbf{U} \cup \mathbf{V})$, then $(GRAPH\ u\ P)$ is a graph pattern.*

5. *If P is a graph pattern and $u \in (\mathbf{U} \cup \mathbf{V})$, then $(SERVICE\ u\ P)$ is a graph pattern.*

If P is a graph pattern, $vars(P)$ denotes the set of variables occurring in P.

We proceed by defining the semantics of SPARQL. To begin with, we need to define what mappings are:

Definition 7 (Mapping [PAG06])**.** *A mapping is a function $\mu : A \to \mathbf{U}$ for some $A \subset \mathbf{V}$.*

For a triple pattern t with $vars(t) \subseteq dom(\mu)$, we write $\mu(t)$ to denote the triple after replacing the variables in t by the corresponding URIs according to μ.

We can then proceed to define when two mappings are compatible.

Definition 8 (Compatible Mappings [PAG06])**.** *Two mappings μ_1 and μ_2 are called compatible (denoted $\mu_1 \sim \mu_2$) if $\mu_1(x) = \mu_2(x)$ for all $x \in dom(\mu_1) \cap dom(\mu_2)$.*

We denote $\mu_\emptyset$ as the mapping with empty domain. This means it is compatible with any other mapping.

A given pattern is subsumed by another pattern if we can "extend" the original pattern to the other pattern.

Definition 9 ([PAG06])**.** *A mapping μ_1 is subsumed by μ_2 (written $\mu_1 \sqsubseteq \mu_2$) if $\mu_1 \sim \mu_2$ and $dom(\mu_1) \subseteq dom(\mu_2)$. μ_2 is then called "extension" of μ_1.*

We proceed with the definition of a dataset.

Definition 10 ([BAACP13, PAG06])**.** *A dataset is a set $DS = \{(def, G), (g_1, G_1), \ldots, (g_k, G_k)\}$, with $k \geq 0$. It contains pairs of URIs and graphs, where the default graph G is identified by the special symbol $def \notin \mathbf{U}$ and the remaining so-called "named" graphs (G_i) are identified by URIs $(g_i \in \mathbf{U})$.*

We assume that any query is evaluated over a fixed dataset DS and that any SPARQL endpoint that is identified by an URI $c \in \mathbf{U}$ evaluates its queries against its own fixed dataset $DS_c = \{(def, G_c), (g_{c,1}, G_{c,1}), \ldots, (g_{c,k}, G_{c,k})\}$.

The functions *graph*, *names* and *ep* are further assumed to define the semantics of SPARQL.

Definition 11 (The functions graph, names and ep [BAACP13, PAG06])**.** *We assume a function $graph(g, DS)$ which, given a Dataset $DS = \{(def, G), (g_1, G_1), \ldots, (g_k, G_k)\}$ and a graph name g as input returns the graph corresponding to the symbol g. The function $names(DS)$ returns the set of names of the input dataset DS: For example $names(DS) = \{g_1, g_2, \ldots, g_k\}$. We also assume a partial function $ep : \mathbf{U} \to DS$, such that for every $c \in \mathbf{U}$, if $ep(c)$ is defined, then $ep(c) = DS_c$, i.e., the dataset associated with the endpoint accessible via the URI c.*

Finally we proceed to define the evaluation given a graph pattern, a dataset and a graph.

Definition 12 (SPARQL Semantics [BAACP13, PAG06])**.** *The evaluation of graph patterns over an RDF Graph G and Dataset DS, where $(g_1, G) \in DS$ for $g_1 \in \mathbf{U} \cup \{def\}$ is formalized as a function $[\![\cdot]\!]_G^{DS}$, which, given a SPARQL graph pattern returns a set of mappings. For a graph pattern P, it is defined recursively:*

1. $[\![t]\!]_G^{DS} = \{\mu \mid dom(\mu) = vars(t) \text{ and } \mu(t) \in G\}$ *for a triple pattern t.*
2. $[\![P_1 \ AND \ P_2]\!]_G^{DS} = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]_G^{DS},\ \mu_2 \in [\![P_2]\!]_G^{DS} \text{ and } \mu_1 \sim \mu_2\}$.
3. $[\![P_1 \ OPT \ P_2]\!]_G^{DS} = [\![P_1 \ AND \ P_2]\!]_G^{DS} \cup \{\mu_1 \in [\![P_1]\!]_G^{DS} \mid \forall \mu_2 \in [\![P_2]\!]_G^{DS} : \mu_1 \not\sim \mu_2\}$.
4. $[\![P_1 \ UNION \ P_2]\!]_G^{DS} = [\![P_1]\!]_G^{DS} \cup [\![P_2]\!]_G^{DS}$.
5. $[\![GRAPH \ u \ P_1]\!]_G^{DS} = \begin{cases} [\![P_1]\!]_{graph(u,DS)}^{DS} & \text{if } u \in names(DS) \\ \{\} & \text{if } u \in \mathbf{U} \backslash names(DS) \\ * & \text{if } u \in \mathbf{V} \end{cases}$
   $* = \{\mu \cup [u \to s] \mid s \in names(DS), \mu \in [\![P_1]\!]_{graph(s,DS)}^{DS} \wedge [u \to s] \sim \mu\}$
   for $u \in \mathbf{U} \cup \mathbf{V}$.
6. $[\![SERVICE \ u \ P_1]\!]_G^{DS} = \begin{cases} [\![P_1]\!]_{graph(def,ep(u))}^{ep(u)} & \text{if } u \in dom(ep) \\ \{\mu_\emptyset\} & \text{if } u \in \mathbf{U} \backslash dom(ep) \\ *' = & \text{if } u \in \mathbf{V} \end{cases}$
   $*' = \{\mu \cup [u \to s] \mid s \in dom(ep), \mu \in [\![P_1]\!]_{graph(def,ep(s))}^{ep(s)} \wedge [u \to s] \sim \mu\}$
   for $u \in \mathbf{U} \cup \mathbf{V}$.
7. $[\![(P' \ FILTER \ R)]\!]_G = \{\mu \mid \mu \in [\![P']\!]_G \text{ and } \mu \models R\}$,

where μ satisfies a filter constraint R, denoted $\mu \models R$, if one of the following holds:

1. *R is $\top$;*
2. *R is $x = a$, $x \in dom(\mu)$ and $\mu(x) = a$;*
3. *R is $x = y$, $\{x, y\} \subseteq dom(\mu)$ and $\mu(x) = \mu(y)$;*

4. *R is bound*(x) *and* $x \in dom(\mu)$*;*

5. *R is a boolean combination of filter constraints evaluating to true under the usual interpretation of* $\neg, \wedge$ *and* $\vee$*.*

When we don't use the SERVICE or GRAPH operators and assume a graph G we will implicitly assume a dataset $DS = \{(def, G)\}$ and we will denote the evaluation function with $[\![\cdot]\!]_G$ instead of $[\![\cdot]\!]_G^{DS}$.

We often use the term "destination" of a subquery containing the SERVICE or GRAPH operator in the top level. The destination refers to the URI or variable in between the graph pattern and the GRAPH or SERVICE operator.

Definition 13 (Destination of a SERVICE- or GRAPH-operator)**.** *Given a pattern* P *of the form* $P = (SERVICE\ u\ P_1)$ *or* $P = (GRAPH\ u\ P_1)$ *we call* u *the destination of the pattern* P*.*

2.2 The Semantics of the SERVICE and GRAPH Operator

We were introduced to the GRAPH and SERVICE operators by [SHa] but in the first formalization of it by Buil-Aranda et al. [BAACP13] the definition varied from the standard. We tweaked their definitions so it reflected the standard. Conversely to our semantic definition of the GRAPH operator in Definition 12, they provided the following definition for GRAPH (note the difference in the second case):

1. $[\![\text{GRAPH } u\ P_1]\!]_G^{DS} = \begin{cases} [\![P_1]\!]_{graph(i,DS)}^{DS} & \text{if } u \in names(DS) \\ \{\mu_\emptyset\} & \text{if } u \in \mathbf{U} \backslash names(DS) \\ {}^* & \text{if } u \in \mathbf{V} \end{cases}$

   ${}^*\{\mu \cup [u \to s] \mid \exists s \in names(DS), \mu \in [\![P_1]\!]_{graph(s,DS)}^{DS} \wedge [u \to s] \sim \mu\}$
   for $u \in \mathbf{U} \cup \mathbf{V}$.

Consider the following example:

Example 1. $DS = \{(def, G_0), (a, G_1)\}$ *where* $G_0 = \{(c, k, g), (a, k, g)\}$*,* G_1 *is arbitrary and* $Q = (GRAPH\ x\ (GRAPH\ c\ P_1))\ AND\ (x, k, g)$*.*

Evaluating the first part of the query, namely $[\![(GRAPH\ x\ (GRAPH\ c\ P_1))]\!]_{G_0}^{DS}$ where P_1 could be any arbitrary pattern yields an interesting result. Because $c \notin names(DS)$ and thus $[\![(GRAPH\ c\ P_1)]\!]_{G_1}^{DS} = \mu_\emptyset$ we obtain $x \to a$ ($\mu_\emptyset$ is compatible with every mapping). This seems unintuitive because it affects the rest of the query. If we would evaluate the right side of the AND we would obtain the following: $[\![(x, k, g)]\!]_{G_0}^{DS} = \{(x \to a), (x \to c)\}$ considering the left part of the AND-Operator we get the result of the query: $\{(x \to a)\}$.

Note that even though the graph with the URI c does not exist and we are not able to evaluate pattern P_1 over it, we are able to somehow receive results using the AND operator. This syntax might be fitting for the SERVICE operator where we query SPARQL endpoints in the web which might be currently unavailable, but for the GRAPH operator where we query different graphs within a reachable endpoint this semantics is not needed.

Considering containment/equivalence of two queries which just use the GRAPH Operator, there are specific instances where it is not obvious if one query is contained in the other query even though it should be intuitively.

Example 2. Q_1 = *(GRAPH x (GRAPH c (GRAPH y)* P_1*))*,
Q_2 = *(GRAPH y (GRAPH c (GRAPH x)* P_1*))*
$?X, ?Y \notin P_1$.

Approaching this example intuitively, one could say that $[\![Q_1]\!]_G^{DS} = [\![Q_2]\!]_G^{DS}$ for all datasets DS and graphs G in DS because x and y get bound to every variable in the dataset because neither x nor y occur in P_1, the triple pattern at the end of the query is the same and also the graph queried second is the same. To show the opposite, just consider the following dataset: $DS = \{(def, G), (a, G_1)\}$. Note that the URI c does not occur in the dataset. But this means that $[\![Q_1]\!]_G^{DS} = \{x \to a\}$ and $[\![Q_2]\!]_G^{DS} = \{y \to a\}$. In the W3C-recommendation[SHa] the following definition can be found:

Algorithm 2.1: W3C-recommendation on evaluating the GRAPH operator

```
1 if IRI is a graph name in D
2    eval(D(G), Graph(IRI,P)) = eval(D(D[IRI]), P)
3 if IRI is not a graph name in D
4    eval(D(G), Graph(IRI,P)) = the empty multiset
5    eval(D(G), Graph(var,P)) =
6 Let R be the empty multiset
7    foreach IRI i in D
8       R := Union(R, Join( eval(D(D[i]), P) , Ω(?var->i) )
9 the result is R
```

If we look at the second case, we can see that the empty multiset (because the definition goes with bag semantics) is returned if the URI is not a graph name in the Dataset resulting in our original definition of SPARQL semantics.

However remembering the definition of the SERVICE operator in Definition 12, we emphasize that the second case of the SERVICE-operator (if $c \in \mathbf{U} \backslash dom(ep)$ return $\mu_\emptyset$) is intended and correct. It makes sure that if a SPARQL endpoint is temporarily unavailable the query does not fail.

CHAPTER 3

Well-Designed SPARQL

We will start out the section with defining well-designed SPARQL. Then we will study the results of [PS14], in three seperate sections namely decidable containment, undecidable containment and equivalence.

3.1 Introduction to well-designed SPARQL

Well-designed SPARQL is a good fundamental basis for complexity analysis because the evaluation problem is only coNP-complete in contrast to general SPARQL where it is PSPACE-complete [PAG09].

Definition 14 (Well-designed SPARQL [PAG09])**.** *A graph pattern P built only from AND and OPT is well-designed if there does not exist a subpattern $P' = (P_1\ OPT\ P_2)$ of P and a variable $x \in vars(P_2)$ that occurs in P outside P', but not in P_1.*

A graph pattern $P = P_1\ UNION\ \ldots\ UNION\ P_n$ is well-designed if each subpattern P_i is UNION-free and well-designed.

In [LPPS13] it was shown that every well-designed graph pattern can be transformed into OPT normal form in polynomial time.

Definition 15 (OPT normal form)**.** *A graph pattern containing only the operators AND and OPT is in OPT normal form if the OPT operator never occurs in the scope of an AND operator.*

Graph Patterns in OPT normal form can be displayed in a natural tree representation, formalized by so-called well-designed pattern trees. But before defining the well-designed pattern trees we need some basic notation about pattern trees.

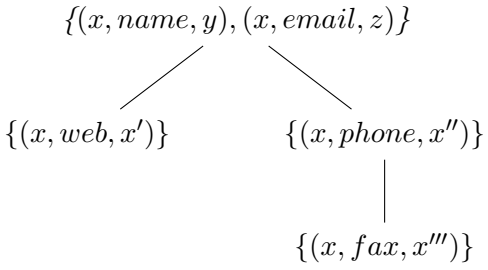
Definition 16 (Pattern Tree)**.** *A pattern tree (PT) T is a pair (T, P) where $T = (V, E, r)$ is a rooted, unordered tree and $P = (P_n)_{n \in V}$ is a labelling of the nodes in V, so that P_n is a non-empty set of triple patterns for every $n \in V$.*

In Example 3 we can see how a pattern in OPT normal form is transformed into a pattern tree.

Example 3 ([PS14])**.** *Consider the graph pattern*

$$P = (((x, name, y)\ AND\ (x, email, z))\ OPT\ (x, web, x'))$$
$$OPT\ ((x, phone, x'')\ OPT\ (x, fax, x'''))$$

and the corresponding pattern tree:



We will now define further details of a pattern tree and after that proceed with defining well-designed pattern trees.

Definition 17 (Components of a pattern tree [PS14])**.** *Let $T = ((V, E, r), P)$ be a pattern tree.*

- *We call the pattern tree $T' = ((V', E', r'), (P_n)_{n \in V'})$ a subtree of T if (V', E', r') is a subtree of T.*
- *An extension of $\hat{T}'$ of a subtree T' of T is a subtree $\hat{T}'$ of T, so that T' is in turn a subtree of $\hat{T}'$. A subtree or extension is proper if some node of the bigger tree is missing in the smaller tree.*
- *Given T, we write $V(T)$ to denote the set V of vertices.*
- *We denote with $pat(T)$ the set $\bigcup_{n \in V(T)} P_n$ and write $vars(T)$ as an abbreviation for $vars(pat(T))$.*
- *Given a node $n \in V(T)$, we define $branch(n) = n^1, \ldots, n^k$ with $n^1 = r$ and $n^k = n$ as the unique sequence of nodes from the root r to n.*

- *For nodes $n, \hat{n} \in V(T)$, so that $\hat{n}$ is the parent of n, let $newvars(n) = vars(n) \backslash vars(branch(\hat{n}))$.*
- *A node n is a child of a PT T if $n \notin V(T)$ and n is the child of some node $n' \in V(T)$.*

After thoroughly describing notions of pattern trees we will use later on, we are now ready to look at the definition of well-designed pattern trees(wdPTs).

Definition 18 (Well-designed pattern tree (wdPT) [PS14])**.** *A well-designed pattern tree (wdPT) is a pattern tree $T = (T, P)$ where for every variable $x \in V(T)$, the nodes $\{n \in V(T) \mid x \in vars(n)\}$ induce a connected subgraph of T.*

We can observe that the pattern tree in Example 3 is well-designed. The restriction of pattern trees to to well-designed pattern trees has two important effects [PS14]:

1. The natural translation from a wdPT to a graph pattern yields a well-designed graph pattern.
2. Every PT derived from a well-designed graph pattern in OPT normal form yields a wdPT.

We can now in polynomial time transform a wdPT into an equivalent well-designed pattern and vice versa. There are two important properties of wdPTs that need to be mentioned. The first property says that we can transform every wdPT into NR normal form. The second property says that there is a unique node in a wdPT where a variable occurs first.

Proposition 1 ([LPPS12, PS14])**.**

1. *Let T be a wdPT. If $newvars(n) \neq \emptyset$ for every $n \in V(T)$ we say that T is in NR normal form. Every wdPT T can be transformed efficiently into NR normal form. From now on we assume w.l.o.g. that all wdPTs are in NR normal form.*
2. *For every variable $x \in vars(T)$, there is a unique node $n \in V(T)$, s.t. $x \in newvars(n)$ and all other nodes $n' \in V(T)$ with $x \in vars(n')$ are descendants of n.*

We denote the results of evaluating a wdPT T over some RDF graph G by $[\![T]\!]_G$.

Lemma 1 describes a characterization which decides whether a mapping $\mu \in [\![T]\!]_G$ in terms of maximal subtrees of a wdPT T.

Lemma 1 (Semantics of pattern trees [LPPS12])**.** *Let T be a wdPT in NR normal form and G an RDF graph. Then $\mu \in [\![T]\!]_G$ iff there exists a subtree T' of T, s.t.*

1. *$dom(\mu) = vars(T')$, and*

2. T' is the maximal subtree of T, s.t. $\mu \sqsubseteq [\![pat(T')]\!]_G$.

We can transform this characterization into Algorithm 3.1. The evaluation problem of well-designed SPARQL can be shown to be coNP-complete using Algorithm 3.1: Step 2 and step 3 of Algorithm 3.1 are doable in polynomial time. Step 4 of Algorithm 3.1 is in coNP because each check if a subtree is maximal is in coNP. There are linearly many nodes to check and thus the overall runtime is coNP.

Algorithm 3.1: co-NP algorithm for EVAL in the well designed fragment [LPPS12]

1. INPUT: A well-designed pattern P in OPT-normal form, a graph G and a mapping μ.
2. Transform the well-designed pattern P into a wdPT T in NR normal form.
3. Look for a subtree T' of T including the root of T where the variables of the triples in T' are equal to the variables in $dom(\mu)$. When we now use the mapping μ on all the triples, the triples must be in the graph G. If such a T' exists, T' witnesses that μ is at least part of a solution in G.
4. Check if the subtree T' is maximal. If we can extend T' with a child of T' so that the mapping μ maps all the triples into G, the subtree T' was not maximal. If it is maximal $\mu \in [\![T]\!]_G$ holds.

Projection is interpreted as a top level operator on top of a graph pattern.

Definition 19 (Projection of a mapping)**.** *For a mapping μ and a set X of variables we denote $\mu_{|X}$ as the projection of μ to the variables in X, call it μ'. $dom(\mu') := X \cap dom(\mu)$ and $\mu'(x) := \mu(x)$ for all $x \in dom(\mu')$.*

We are now ready to define projected well-designed trees: A projected well-designed tree (pwdPT) is a pair (T, X) where T is a wdPT and X a set of variables.

Definition 20 (Result of pwdPTs [PS14])**.** *Let G be a RDF graph. Then $[\![(T, X)]\!]_G = \{\mu_{|X} \mid \mu \in [\![T]\!]_G\}$.*

In a projected wdPT there are free variables and existential variables.

Definition 21 (Free and existential variables [PS14])**.** *Let (T, X) be a pwdPT. Then the free variables of (T, X) are defined as $fvars(T) = vars(T) \cap X$ and the existential variables of (T, X) are defined as $vars(T) \backslash fvars(T)$. Analogously, we write $fvars(n)$ and $evars(n)$, respectively, for nodes $n \in V(T)$.*

In [LPPS13] it was shown that a similar characterization of solution as in Lemma 1 exists. A pwdPT (T, X) is in NR normal form if T is. W.l.o.g., we assume that existential variables when we look at more than one pwdPT are always renamed apart.

Lemma 2 ([LPPS13]). *Let (T, X) be a pwdPT in NR normal form, G an RDF graph and μ a mapping with $dom(\mu) \subseteq X$. Then $\mu \in [\![(T, X)]\!]_G$ iff there exists a subtree T' of T, s.t.*

1. *$dom(\mu) = fvars(T')$ and*
2. *there exists a mapping $\lambda : evars(T') \mapsto dom(G)$, s.t. $\mu \cup \lambda \in [\![T]\!]_G$.*

The last outgrowth of pwdPTs we will consider are unions of well-designed pattern trees. Unions of well-designed patterns correspond to the usage of the UNION operator on top level, i.e., $P = P_1$ UNION ... UNION P_k where $P_1, \dots P_k$ are well-designed. They can be considered as a set $\{T_1, \dots, T_k\}$ of wdPTs. Such a set is called well-designed pattern forest (wdPF) F. We can analogously define projected well-designed pattern forests (pwdPFs) as a tuple (F', X) where $F' = \{(T_1, X), \dots, (T_k, X)\}$.

Definition 22 (Result of wdPFs and pwdPF [PS14]). *Let G be a RDF graph. Then $[\![F]\!]_G = \bigcup_{T \in F} [\![T]\!]_G$ and $[\![(F', X)]\!]_G = \bigcup_{(T,X) \in F'} [\![(T, X)]\!]_G$.*

A subforest of a forest F is a set of subtrees of F. We can extend $pat(\cdot)$ to wdPFs: Let F be a wdPF then $pat(F) = \bigcup_{T \in F} pat(T)$ and analogously $vars(F)$.

In [PS14] query containment and query equivalence of well designed SPARQL were studied. As we want to analyse the complexity of the SERVICE-operator it is crucial to understand the work that was previously done regarding the fragment of well-designed graph patterns. Research in query containment and query equivalence is very important in the context of static query analysis and optimization: Optimization of queries can be done by replacing a query with a new query which possesses better computational properties. Replacing queries preserving the meaning of the original meaning can only be done by checking if the replaced query is equivalent to the original query. Also, later on in Chapter 5, when we analyse boundedness of the destination variable of the service operator, we will see that equivalence plays a role whether a query is feasible to be evaluated. In [PS14] a fine grained analysis of well-designed SPARQL by dividing it into several fragments was done. The fragments are distinguished by the operators used. All fragments contain triple patterns and the AND-operator. This fragment corresponds to conjunctive queries regarding complexity [LPPS13] and is denoted with $\{\emptyset\}$. Adding the UNION-operator yields the fragment denoted by $\{\cup\}$. Adding projection yields the fragment denoted by $\{\pi\}$ and adding both the UNION-operator and projection yields the fragment denoted by $\{\pi, \cup\}$.

In order to define the problems **CONTAINMENT**, **EQUIVALENCE** and **SUBSUMPTION** which we will investigate over the fragments that we mentioned above, we need to define what containment, equivalence and subsumption mean in the SPARQL language.

Definition 23 (Containment)**.** *P_1 is contained in P_2 ($P_1 \subseteq P_2$) if $[\![P_1]\!]_G \subseteq [\![P_2]\!]_G$ for every RDF Graph G.*

If a graph pattern P_1 is contained by a graph pattern P_2, all solutions of P_1 are solutions of P_2 regardless of the graph they are evaluated over.

Definition 24 (Equivalence)**.** *Given two graph patterns P_1 and P_2 they are equivalent (written as $P_1 \equiv P_2$) if $[\![P_1]\!]_G = [\![P_2]\!]_G$ for every RDF Graph G.*

Equivalence between two graph patterns P_1, P_2 means that regardless of the graph they get evaluated over, they always have the same solutions.

Definition 25 (Subsumption)**.** *P_1 is subsumed by P_2 (denoted $P_1 \sqsubseteq P_2$ if for every $\mu \in [\![P_1]\!]_G$ there exists a $\mu' \in [\![P_2]\!]_G$, s.t. $\mu \sqsubseteq \mu'$.*

If a graph pattern P_1 is subsumed by the graph pattern P_2 all the solutions of P_1 can be extended to a solution of P_2. This means that if we add additional mappings to a solution of P_1 we can "build" a solution of P_2.

After reading the definitions of containment, equivalence and subsumption the following computational problems arise:

CONTAINMENT$[S_1, S_2]$
INPUT: Graph pattern P_1 from wd-SPARQL$[S_1]$, graph pattern P_2 from wd-SPARQL$[S_2]$
QUESTION: Does $P_1 \subseteq P_2$ hold?

EQUIVALENCE$[S_1, S_2]$
INPUT: Graph pattern P_1 from wd-SPARQL$[S_1]$, graph pattern P_2 from wd-SPARQL$[S_2]$
QUESTION: Does $P_1 \equiv P_2$ hold?

SUBSUMPTION$[S_1, S_2]$
INPUT: Graph pattern P_1 from wd-SPARQL$[S_1]$, graph pattern P_2 from wd-SPARQL$[S_2]$
QUESTION: Does $P_1 \sqsubseteq P_2$ hold?

S_1 and S_2 always denote a fragment of well-designed SPARQL for example $\{\cup, \pi\}$. The main results of [PS14] are nearly all the solutions to **CONTAINMENT**$[S_1, S_2]$ and

$\downarrow S_1 \backslash S_2 \rightarrow$	$\{\emptyset\}$	$\{\cup\}$	$\{\pi\}$	$\{\cup, \pi\}$
$\{\emptyset\}$	NP-c.	$\Pi_2^P - c.$	undec.	undec.
$\{\pi\}$	NP-c.	$\Pi_2^P - c.$	undec.	undec.
$\{\cup\}$	NP-c.	$\Pi_2^P - c.$	undec.	undec.
$\{\cup, \pi\}$	NP-c.	$\Pi_2^P - c.$	undec.	undec.

Table 3.1: Containment$[S_1, S_2]$ [PS14]

$\downarrow S_1 \backslash S_2 \rightarrow$	$\{\emptyset\}$	$\{\cup\}$	$\{\pi\}$	$\{\cup, \pi\}$
$\{\emptyset\}$	NP-c.	-	-	-
$\{\pi\}$	$\Pi_2^P - c.$	$\Pi_2^P - c.$	-	-
$\{\cup\}$	$\Pi_2^P - c.$	$\Pi_2^P - h.$	undec.	-
$\{\cup, \pi\}$	$\Pi_2^P - c.$	undec.	undec.	undec.

Table 3.2: Equivalence$[S_1, S_2]$ [PS14]

EQUIVALENCE$[S_1, S_2]$ where S_1, S_2 are any of the well designed SPARQL fragments mentioned before. The results are best represented by Table 3.1 and Table 3.2.

3.2 Decidable Containment

First the decidable cases of Table 3.1 are proven, i.e., S_1 is an arbitrary subset of $\{\cup, \pi\}$ and S_2 doesn't contain π. In order to minimize the number of proofs a well known strategy is applied: When membership of a general case is shown, the more specific case is implied and when showing hardness for a more specific case, the more general cases are implied. Showing the following results fill in the first two columns of the Containment$[S_1, S_2]$ table:

- NP-membership of **CONTAINMENT**$[\{\cup, \pi\}, \emptyset]$
- Π_2^P-membership of **CONTAINMENT**$[\{\cup, \pi\}, \{\cup\}]$
- Π_2^P-hardness of **CONTAINMENT**$[\emptyset, \{\cup\}]$

NP-hardness of **CONTAINMENT**$[\emptyset, \emptyset]$ follows immediately from the NP-hardness of **EQUIVALENCE**$[\emptyset, \emptyset]$ which was shown in [LPPS13].

For the NP-membership of **CONTAINMENT**$[\{\cup, \pi\}, \emptyset]$, first **CONTAINMENT**$[\{\pi\}, \emptyset]$ is shown and extended to **CONTAINMENT**$[\{\cup, \pi\}, \emptyset]$. We notice that we deal with both pwdPTs and wdPTs as S_1 contains projection. Theorem 1 provides a necessary and sufficient criterion to decide $(T_1, X) \subseteq T_2$.

Theorem 1. *[PS14] Let (T_1, X) be a pwdPT and let T_2 be a wdPT. Then $(T_1, X) \subseteq T_2$ iff for every subtree T_1' of T_1,*

1. *either there exists a child node n of T_1' and a homomorphism $h : pat(n) \to pat(T_1')$ with $h(x) = x$ for all $x \in vars(n) \cap vars(T_1')$,*
2. *or there exists a subtree T_2' of T_2, s.t.*
   a) *$fvars(T_1') = vars(T_2')$*
   b) *$pat(T_2') \subseteq pat(T_1')$, and*
   c) *for all extensions $\hat{T}_2'$ of T_2' there exists an extension $\hat{T}_1'$ of T_1' and a homomorphism $h : pat(\hat{T}_1') \mapsto pat(T_1') \cup pat(\hat{T}_2')$ with $h(x) = x$ for all $x \in vars(T_1')$.*

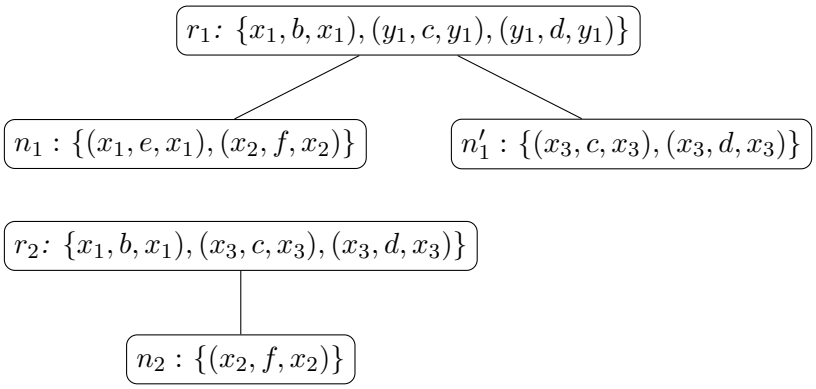
Proof Idea. Let G be an arbitrary RDF graph. Let T_1' be an arbitrary subtree of T_1. Let σ be a mapping s.t. $dom(\sigma) = vars(T_1')$ and $\sigma \in [\![T_1]\!]_G$. We need to show that $\sigma_{|X} \in [\![T_2]\!]_G$.

The first case of the theorem captures the case where we have a subtree which is not maximal. For this subtree we could take any $\sigma \in [\![T_1']\!]_G$ and extend it with the variables in n because $h : pat(n) \to pat(T_1')$ with $h(x) = x$ for all $x \in vars(n) \cap vars(T_1')$ holds. But such a mapping σ can not be in $[\![T_1]\!]_G$. Thus the subset relation is obviously fulfilled.

The second case captures solutions σ with $dom(\sigma) = vars(T_1')$. It remains to check if $\mu = \sigma_{|X}$ is in $[\![T_2]\!]_G$. The first two conditions make sure that μ also maps T_2' into G. To make it a solution for T_2 it remains to show that there is no extension μ' of μ so that $\mu' \in [\![T_2]\!]_G$. Property (2c) makes sure that if μ' exists, we can find an extension σ' such that it binds more variables than σ. But then σ is no solution. □

To visualize the procedure consider Example 4.

Example 4 ([PS14])**.** *Assume a $pwdPT(T_1, X)$ with $X = \{x_1, x_2, x_3\}$ and $wdPT\ T_2$ as shown below.*



We will now use the criteria in Theorem 1 to check if $(T_1, X) \subseteq T_2$ holds. Consider all the subtrees of T_1: $\{r_1\}, \{r_1, n_1\}, \{r_1, n_1'\}, \{r_1, n_1, n_1'\}$ and check if either of the two properties in Theorem 1 hold. Consider the subtree $\{r_1, n_1'\}$ and property (2): We can easily see that

the properties (2a) and (2b) are satisfied by the subtree of T_2 that contains only r_2. It remains to check (2c) and this is where the problem occurs: The required homomorphism from $pat(n_1)$ into $pat(r_1) \cup pat(n_1') \cup pat(r_2) \cup pat(n_2)$ doesn't exist. We can also provide the following counterexample: $G = \{(g_{x1}, b, g_{x1}), (h_{y1}, c, h_{y1}), (h_{y1}, d, h_{y1}), (j_{x2}, f, j_{x2})\}$. Clearly $\mu \cup \lambda \in [\![T_1]\!]_g$ for $\mu(x_1) = g_{x1}, \mu(x_3) = h_{y1}$ and $\lambda(y_1) = h_{y1}$ and thus $\mu \in [\![(T_1, X)]\!]_G$. However $\mu \notin [\![T_2]\!]_G$ because μ' an extension of μ can be created with $\mu'(x_2) = j_{x2}$ and $\mu'(pat(n_2)) \subseteq G$ and thus $(T_1, x) \not\subseteq T_2$. We could change $pat(n_2)$ to get $(T_1, x) \subseteq T_2$: Let $pat(n_2) = \{(x_2, f, x_2, x_1, e, x_1)\}$ and observe that a homomorphism h for property (2c) of Theorem 1 exists if we define h as the identity. In fact, then $(T_1, X) \subseteq T_2$.

It is easy to see that Theorem 1 can be transformed into a Π_2^P-algorithm 3.2.

Algorithm 3.2: Π_2^P-algorithm for **CONTAINMENT**$[\{\pi\}, \emptyset]$ from Theorem 1

```
1 INPUT: A pwdPT (T_1, X), a wdPT T_2
2 For all subtrees T_1' of T_1
3 guess the subtree T_2' of T_2 which produces the same mappings as T_1'.
4 If no such tree exists return false.
5 If the property holds for all T_1' of T_1 return true.
```

On the other hand it is far from obvious, that the problem is not Π_2^P-hard: One can in fact get rid of one source of complexity and push the complexity of the **CONTAINMENT**$[\{\pi\}, \emptyset]$ problem down to NP-completeness. The crucial idea is, that we don't need to look at all the subtrees T_1' of T_1 but polynomially many. The subtrees of interest can be described by defining the closure of a variable.

It is easy to test if $vars(T_2) = fvars(T_1)$ by traversing the trees once, so we assume w.l.o.g. that this property holds. Also, we know that $vars(T_2) = fvars(T_1)$ must hold for $(T_1, X) \subseteq T_2$ to be true as this is an immediate consequence from Theorem 1 (just consider $T_1' = T_1$).

Definition 26 (Closure $(C_1(x), C_2(x))$ of a variable [PS14])**.** *Let (T_1, X) be a pwdPT and let T_2 be a wdPT with $vars(T_2) = fvars(T_1)$. Consider $x \in fvars(T_1)$. The closure of x in (T_1, X) and T_2 is the pair $(C_1(x), C_2(x))$ where $C_i(x)$ (for $i \in \{1, 2\}$) is a subtree of T_i such that the following conditions are met:*

1. *$branch(new\text{-}node_{T_1}(x)) \subseteq V(C_1(x))$,*
2. *$r_2 \in V(C_2(x))$,*
3. *$fvars(C_1(x)) = vars(C_2(x))$, and*
4. *$C_1(x)$ and $C_2(x)$ are minimal with regard to properties 1-3.*

Minimality in (4) means that for all subtrees D_1 of T_1 and D_2 of T_2, if D_1 and D_2 satisfy conditions 1-3, then $V(C_1) \subseteq V(D_1)$ and $V(C_2) \subseteq V(D_2)$ meaning the nodes for the subtrees $V(C_1)$ and $V(C_2)$ are minimized.

Because we assumed $vars(T_2) = fvars(T_1)$ we can easily see that the closure always exists.

Proposition 2 ([PS14]). *Let (T_1, X) be a pwdPT and let T_2 be a wdPT with $fvars(T_1) = vars(T_2)$. Then the closure $(C_1(x), C_2(x))$ exists and can be efficiently computed.*

Proof Idea. The algorithm chooses a not yet chosen variable $x \in fvars(T_1)$ and initializes the trees $C_1(x)$ and $C_2(x)$ by setting $C_1(x) = \text{branch}(\text{new-node}_{T_1})$ and $C_2(x) = V(C_2(x)) = \{r_2\}$ where r_2 is the root of T_2. It remains to fulfill the condition $fvars(C_1(x)) = vars(C_2(x))$ in such a way that $C_1(X)$ and $C_2(X)$ are minimal with regard to the number of vertices. We either have one of the following cases:

1. $fvars(C_1(x)) = vars(C_2(x))$: This means we are done and have successfully computed the closure $(C_1(x), C_2(x))$
2. $fvars(C_1(x)) \supset vars(C_2(x))$: This means we miss the variables $fvars(C_1(x)) \backslash vars(C_2(x))$. By iteratively adding $\text{branch}(\text{new-node}_{T_2}(y))$ to $C_2(x)$ for all variables $y \in fvars(C_1(x)) \backslash vars(C_2(x))$ all the missing variables are now in $C_2(x)$.
3. $fvars(C_1(x)) \subset vars(C_2(x))$: This means we miss the variables $vars(C_2(x)) \backslash fvars(C_1(x))$. By iteratively adding $\text{branch}(\text{new-node}_{T_1}(y))$ to $C_1(x)$ for all variables $y \in vars(C_2(x)) \backslash fvars(C_1(x))$, all the missing variables are now in $C_1(x)$.

By the assumed condition $fvars(T_1) = vars(T_2)$, the procedure will eventually reach a fixpoint. □

Using the closure of the variable is inspired by the following idea: Assume μ is a solution mapping of (T_1, X) and y is part of μ, i.e., a free variable. Now we need to show that μ is also part of T_2, i.e., μ must bind all the variables that occur in T_2 on the branch from the root to the node n where y is introduced. If again an additional variable z is introduced in this path in tree of T_2, μ must bind all the free variables in (T_1, X) along the branch from the root to the first occurrence of z. Using the idea of the closure allows us to formulate an alternative characterization of $(T_1, X) \subseteq T_2$. The improvement is that we only need to check polynomially many closures and not exponentially many subtrees of (T_1, X).

Theorem 2 ([PS14]). *Let (T_1, X) be a pwdPT and let T_2 be a wdPT. Then $(T_1, X) \subseteq T_2$ if and only if $fvars(T_1) = vars(T_2)$ and for every $x \in fvars(T_1)$*

1. *$pat(C_2(x)) \subseteq pat(C_1(x))$*

2. *for every* $n \in V(C_1(x))$ $branch(new\text{-}node_{T_1}(x))$*, there exists a homomorphism* $h_1 : pat(n) \mapsto pat(branch(\hat{n})) \cup pat(branch(new\text{-}node_{T_1}(x)))$ *(where* $\hat{n}$ *is the parent node of* n *in* T_1*) with* $h_1(x) = x$ *for all* $x \in vars(n) \cap (vars(branch(\cap n)) \cup vars(branch(new\text{-}node_{T_1}(x))))$*, and*

3. *for every child node* m *of* $C_2(x)$*, and for every variable* $y \in newvars(m)$*, the following property holds: let* $n \in branch(new\text{-}node_{T_1}(y))$*. Then there exists a homomorphism* $h_2 : pat(n) \mapsto pat(C_1(x)) \cup pat(m) \cup pat(branch(\hat{n}))$ *(where* $\hat{n}$ *is the parent node of* n*) with* $h_2(x) = x$ *for all* $x \in vars(n) \cap (vars(C_1(x)) \cup vars(branch(\hat{n})))$*.*

Proof Idea. The first property is the most obvious one: every mapping σ which is a solution for $C_1(x)$ assuming G as our arbitrary graph, results in $\mu = \sigma_{|X}$ also being a solution for $C_2(x)$. The second condition makes sure that when x is in the domain of the solution mapping σ all of $C_1(x)$ was used when retrieving the mapping σ from G. The last condition is similar to the condition (2c) in the Theorem1: When $\mu = \sigma_{|X}$ with $dom(\mu) = vars(C_2(x))$ is not a solution for T_2 because $C_2(x)$ could have been extended with some child of $C_2(x)$ it must be that it is possible to extend σ to some child of $C_1(x)$. □

In the actual proof of Theorem 2, Theorem 1 is used to show that the conditions of Theorem 2 are also sufficient.

Theorem 3 ([PS14])**.** ***CONTAINMENT****[*$\{\cup, \pi\}, \emptyset$*] is in NP.*

Proof. Similarly to Theorem 1 we can construct a procedure using Theorem 2 which is in NP. Let (F, X) be a well designed pattern forest (pwdPF) and let T be a $wdPT$. Consider now every pwdPT F' in the forest (F, X) and use our procedure to decide $(F', X) \subseteq T$. Formally we get $(F, X) \subseteq T$ iff. $(T_i, X) \supseteq T$ for every $(T_i, X) \in (F, X)$. □

The next problem that we tackle is **CONTAINMENT**$[\{\cup, \pi\}, \{\cup\}]$: First a necessary and sufficient condition for containment is given and then turned into an algorithm. For this formulation the definition of a renamed proper extension of a wdPF is important.

Definition 27 (renamed proper extension [PS14])**.** *Let* $F = \{T_i \mid with\ 1 \leq i \leq k\}$ *be a wdPF,* F' *a subforest of* F*. For every* $T_i \in F$*, an injective renaming function* ρ_i *with* $dom(\rho_i) = vars(T_i)$*, s.t.*

1. $\rho_i(x) = x$ *for all* $x \in vars(F')$*,*

2. $\rho_i(x) \neq \rho_j(y)$ *for every* $x \in vars(T_i) \backslash vars(F'), i \neq j \in \{1, \ldots, k\}$ *and* $y \in dom(\rho_j)$*. Finally let* $\hat{F}$ *be the wdPF* $\{\rho_i(T_i) \mid 1 \leq i \leq k\}$*.*

Then a renamed proper extension of F' *is a subforest of* $\hat{F}$*, call it* $\hat{F}'$ *that has* F' *as a proper subforest, i.e.,* $\hat{F}'$ *is not equal to* F'*.*

After the definition of renamed proper extensions we are ready to present the characterization for **CONTAINMENT**$[\{\cup, \pi\}, \{\cup\}]$.

Theorem 4 ([PS14])**.** *Let (T_1, X) be a pwdPT and let F_2 be a wdPF. Then $(T_1, X) \subseteq F_2$ iff for every subtree T_1' of (T_1, X):*

1. *either there exists a child node n of T_1' s.t. there is a homomorphism $h : path(n) \mapsto pat(T_1')$ with $h(x) = x$ for all $x \in varS(n) \cap vars(T_1')$*
2. *or there exists a subtree T_2' of F_2 with $vars(T_2') = fvars(T_1')$ and $pat(T_2') \supseteq pat(T_1')$ s.t. every renamed proper extension F_2' of $\{T_2'\}$ in F_2 satisfies one of the following properties:*
   a) *there exists a proper renamed extension $\hat{F}_2'$ of F_2' (i.e. a bigger extension than F_2' nodewise), and a homomorphism $h_a : pat(\hat{F}_2') \mapsto pat(F_2')$ with $h(x) = x$ for all $x \in vars(\hat{F}_2') \cap (vars(F_2') \cup vars(T_1'))$, or*
   b) *there exists an extension $\hat{T}_1'$ of T_1' and a homomorphism $h_b : pat(\hat{T}_1') \mapsto pat(F_2') \cup pat(T_1')$ with $h(x) = x$ for all $x \in vars(T_1')$, or*
   c) *case (2a) does not apply and there exists a tree $T \in F_2'$ with $vars(T) = fvars(T_1')$.*

Proof Idea. The conditions presented in Theorem 4 are similar to those presented in Theorem 1: First we choose an arbitrary subtree T_1' of (T_1, X) and inspect the mappings σ that T_1' induces assuming an arbitrary Graph G:

- Either σ can be extended to some child node of T_1' but then again σ is not a solution of (T_1, X) over G
- or $\sigma_{|X} \in [\![F_2]\!]_G$.

Condition (1) takes care of subtrees for which σ could be extended to some additional child node of the subtree. This would mean that the subtree is not a valid solution for (T_1, X). Condition (2) extends condition (2) from Theorem1: It is easy to see that we fulfill condition (2a) and condition (2b) from Theorem 1 similarly by saying that the subtree T_2' of F_2 with $vars(T_2') = fvars(T_1')$ and $pat(T_2') \subseteq pat(T_1')$. The condition(2c) of Theorem 1 was used to make sure that any extension μ' of μ for which $\mu' \in [\![T_2]\!]_G$ holds would result in σ being no solution of T_1 over G. To extend condition (2c) of Theorem 1 the notion of proper renamed extension comes into use. This is due to the fact that we now have a pattern forest and μ might still be a solution of another tree in F_2. This means that all subtrees in F_2 with $vars(T_2') = fvars(T_1')$ and $pat(T_2') \subseteq pat(T_1')$ must be eligible to be extended to show that μ is indeed not a solution of F_2 but extensions of μ are solutions. Condition (2*a*) forces a certain maximality condition onto the proper renamed extensions of T_2', i.e., we cant have more nodes in the proper renamed extension F_2' without having a pattern mismatch in the proper renamed extension F_2' and the extension

$\hat{F}_2'$ of it. A pattern mismatch means that the homomorphism $h_a : pat(\hat{F}_2') \mapsto pat(F_2')$ does not exist. Condition $(2c)$ makes sure that the proper renamed extension F_2' extends all the relevant subtrees of F_2, i.e. a relevant subtree is a tree $T \in F_2'$ where $vars(T) = fvars(T_1')$. Condition (2b) checks for the existence of a homomorphism h_b that maps $pat(\hat{T}_1')$ into the patterns of the renamed proper extension $pat(F_2) \cup pat(T_1')$. □

The characterization in Theorem 4 yields a more or less straightforward Σ_2^P-algorithm for testing $(T_1, X) \not\subseteq F_2$.

Algorithm 3.3: Π_2^P-algorithm for **CONTAINMENT**$[\{\cup, \pi\}, \{\cup\}]$ from Theorem 4

1 INPUT: A pwdPF (F_1, X) and a wdPF F_2
2 Check $(T_i, X) \subseteq F_2$ for every $(T_i, X) \in (F_1, X)$:
3 Guess T' and the proper renamed extension $\hat{F}_2'$
4 Use a coNP-oracle to check that there does not exist a child node n and homomorphism h as described by property $(2b)$.

The hardness proof of **CONTAINMENT**$[\emptyset, \{\cup\}]$ is done by a reduction from the well known Π_2^P-complete problem 3-QSAT$_{\forall,2}$ [GJ79].

3-QSAT$_{\forall,2}$
INPUT: A formula $\phi = \forall\vec{x}\exists\vec{y}\psi$,
where ψ is a Boolean formula in CNF over the variables $\vec{x} \cup \vec{y}$.

QUESTION: Can every assignment I on the variables in $\vec{x}$ be extended to an assignment J on $\vec{y}$, s.t. $J \models \psi$?

Theorem 5 ([PS14])**.** *CONTAINMENT*$[\emptyset, \{\cup\}]$ *is* Π_2^P*-hard.*

Proof Idea. Assume an arbitrary instance of 3-QSAT$_{\forall,2}$ and construct an instance of **CONTAINMENT**$[\emptyset, \{\cup\}]$ as follows: The wdPT T_1 consists of the root and two child nodes: n_i, n_i' for every variables $x_i \in \vec{x}$. In this way we are able to model the assignment I of the 3-QSAT$_{\forall,2}$ problem in form of subtrees of T_1. Additionally we have a child node n_0 containing the variables in $\vec{y}$ and an encoding of the formula ψ.

It remains to deal with the "unintended" subtrees of T_1 where given an i either both n_i and n_i', or neither n_i and n_i' are in the subtree. This is done by adding certain wdPTs to F_2 which take care of the two problems. The last wdPT added to the forest F_2 contains the triples encoding the formula ψ in its root plus the child nodes n_i, n_i'. This wdPT produces the solutions of all "intended" subtrees of T_1' if and only if every assignment I on the variables in $\vec{x}$ can be extended to an assignment J on $\vec{y}$, s.t. $J \models \psi$. □

The section dealing with decidable containment closes with settling the complexity of **SUBSUMPTION**$[S_1, S_2]$ problem for every $S_1, S_2 \subseteq \{\cup, \pi\}$. In prior work [LPPS12]

it was shown, that the simple case $S_1 = S_2 = \emptyset$ is Π_2^P-complete. Later on in [LPPS13] the Π_2^P-membership was extended to the case where $S_1 = S_2 = \{\pi\}$ holds. To establish the Π_2^P-completeness to arbitrary $S_1, S_2 \subseteq \{\cup, \pi\}$, it obviously suffices to show the Π_2^P-membership for the most general case.

Theorem 6 ([PS14]). ***SUBSUMPTION**[$\{\cup, \pi\}, \{\cup, \pi\}$] is in Π_2^P.*

Proof Idea. Let (F_1, X) and (F_2, X) be two pwdPFs. The main proofidea is that the following criteria for subsumption was found: $(F_1, X) \sqsubseteq (F_2, X)$ iff for every subtree T_1' of F_1, there exists a subtree T_2' of F_2, s.t.

1. $\mathit{fvars}(T_1') \subseteq \mathit{fvars}(T_2')$ and
2. there exists a homomorphism $h : \mathit{pat}(T_2') \mapsto \mathit{pat}(T_1')$ with $h(x) = x$ for all $x \in \mathit{fvars}(T_1')$.

Consider now the following procedure: For all subtrees T_1' of F_1 check that there exists a subtree T_2' of F_2 together with a homomorphism of the desired property. This procedure can be executed in Π_2^P because we have need to check all subtrees for a property which would be in co-NP, but then the property is a homomorphism check which is in NP. So we have a co-$\mathrm{NP}^{\mathrm{NP}}$ runtime. □

In [LPPS12] the authors noticed an interesting feature of subsumption in SPARQL: In the fragment of wd-SPARQL[$\emptyset$] subsumption is able to characterize equivalence. Assume graph patterns $P_1, P_2 \in$ wd-SPARQL[$\emptyset$]. Then $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$. In [LPPS13] the authors were able to find a counterexample for wd-SPARQL[π]. Now in [PS14] the result is strengthened: Assuming one pattern P_1 in wd-SPARQL[$\emptyset$] and one pattern P_2 in either wd-SPARQL[$\{\pi\}$] or wd-SPARQL[$\{\cup\}$] one can show that $P_1 \not\equiv P_2$ but $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$.

Proposition 3 ([PS14]). *There exist pairs P_1, P_2 of graph patterns P_1 from wd-SPARQL[$\emptyset$] and P_2 from either wd-SPARQL[$\{\pi\}$] wd-SPARQL[$\{\cup\}$], s.t. $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$ hold but $P_1 \not\equiv P_2$.*

Proof. It is an easy observation that we need two counterexamples to prove the proposition.

1. At first we consider the case where $P_1 \in$ wd-SPARQL[$\emptyset$] and $P_2 \in$ wd-SPARQL[$\{\cup\}$]. Let $P_1 = (t_1$ OPT $t_2)$ and $P_2 = (t_1$ UNION $(t_1$ AND $t_2))$. The patterns t_1 and t_2 are assumed to be distinct. $P_1 \sqsubseteq P_2$ can be easily seen because P_2 imitates the semantics of the OPT operator. $P_2 \sqsubseteq P_1$ Also holds because again the semantics of the OPT operator are projected into P_2. But $P_1 \sqsubseteq P_2$ does obviously not hold since P_2 always has mappings which are solely created by t_1.

2. Now for the second counterexample where $P_1 \in$ wd-SPARQL[$\emptyset$] and $P_2 \in$ wd-SPARQL[$\{\pi\}$].

$$\begin{aligned} \text{Let } P_1 = (x_1, a, x_2) \text{ OPT } ((x_3, a, x_2) \text{ AND } (x_3, a, x_3)), \\ P_2 = ((x_1, a, x_2) \text{ AND } (y_1, a, Y_2)) \text{ OPT } ((x_3, a, x_2) \text{ AND } (x_3, a, x_3) \\ \text{AND } (y_3, a, y_2) \text{ AND } (y_3, a, y_3)) \\ \text{and } X = \{x_1, x_2, x_3\}. \end{aligned}$$

It remains to show that $P_1 \sqsubseteq (P_2, X)$ and $(P_2, X) \sqsubseteq P_1$ but $P_1 \not\equiv (P_2, X)$. Towards this goal we observe that the triple patterns in P_1 are contained in P_2. Also, P_2 contains triple patterns with existential variables. We can easily see that there is a homomorphism mapping the triple patterns containing the existential variables into the patterns of P_1. Thus $P_1 \sqsubseteq (P_2, X)$ holds. Also $(P_2, X) \sqsubseteq P_1$ holds, but in general $(P_2, X) \not\subseteq P_1$ since one can provide a graph G and an appropriate instantiation of the existential variables y_1, y_2 in the root of P_2 that block the extension of mapping to the child node. Consider the following RDF graph $G = \{a(1,1), a(2,3)\}$. Then $\mu = \{x_1 \mapsto 1, x_2 \mapsto 1\} \in [\![(T_2, X)]\!]_G$, because of the mapping $\lambda = \mu \cup \{y_1 \mapsto 2, y_2 \mapsto 3\} \in [\![T_2]\!]_G$ which cannot be extended to the child node of r_2. However $\mu \notin [\![T_1]\!]_G$ since μ can be extended to the child node of r_1 by adding $[x_3 \mapsto 1]$. □

3.3 Undecidable Containment

We can see in Table 3.1 that **CONTAINMENT**$[S_1, S_2]$ is undecidable, if $\pi \in S_2$. Again we don't need to show the undecidability for every entry in the table, as we can just show it for the most specific entry, i.e., **CONTAINMENT**$[\emptyset, \{\pi\}]$. We do this proof by reducing from the conjunctive query answering problem under integrity constraints in form of tuple generating dependencies (abbr. tgds) [JK84, CGK08]. This problem is well known to be undecidable [JK84, CGK08]. We are not going to reduce the original problem but a modified version of it to our problem. Three changes need to be made, to make the problem suitable for us:

1. The undecidability results for the problem refer to arbitrary databases, which would include infinite databases. Our RDF graphs however are a finite set of triples.
2. The problem allows predicates of arbitrary length but our RDF graphs contains only triples.
3. Finally, for the problem reduction in the end, it turns out to be convenient to restrict the problem from a set of tgds to a single tgd.

Before doing the first step, it is crucial to define tuple generating dependencies and the undecidable problem, namely **CQ-UNDER-TGDs**.

Definition 28 (tuple generating dependency)**.** *Let $\phi(x)$ and $\psi(x, y)$ be conjunctive queries. Also, let all variables x occur in $\phi(x)$. A tuple generating dependency (tgd) is a first-order formula of the form $\forall x(\phi(x) \rightarrow \exists y\psi(x, y))$.*

To simplify the notation of a tgd, the $\forall$-quantifiers are omitted. Let I be a database instance and τ be a tgd. Then we define $I \models \tau$ using homomorphisms. This is feasible, because a tgd is an implication: For every homomorphism $h : \phi(x) \mapsto I$ (mapping constants to themselves), which is responsible for the antecedent of the implication, there must be an extension h' of h, for which $h' : \psi(x, y) \mapsto I$ holds. It is then natural to define satisfaction for a set of tgds: Let Σ be a set of tgds. Then $I \models \Sigma$ iff. $I \models \tau$ for every $\tau \in \Sigma$. Let Q be conjunctive query. For a set Σ of tgds, a database instance I and a BCQ Q, we say that $I, \Sigma \models Q$ holds, if for every (possible infinite) database instance M, s.t. $M \models \Sigma$ and $I \subseteq M$, we also have $M \models Q$. We write $I, \Sigma \models_f Q$ if only finite models M are allowed.

Consider now the two problems:

CQ-UNDER-TGDs
INPUT: A set Σ of tgds, a database instance I and a CQ Q.
QUESTION: Does $\Sigma, I \models Q$ hold?

FINITE-CQ-UNDER-TGDs
INPUT: A set Σ of tgds, a database instance I and a CQ Q.
QUESTION: Does $\Sigma, I \models_f Q$ hold?

We proceed with step one procedure: By [CGK08] **CQ-UNDER-TGDS** is undecidable. The proof in [CGK08] will be changed so that **FINITE-CQ-UNDER-TGDs** remains undecidable even though $|\Sigma| = 1$. Examining the undecidability proof of **BCQ-UNDER-TGDs** in [CGK08], a reduction from the **HALTING** problem to **BCQ-UNDER-TGDs** is given: The initial configuration of the Turing machine is encoded into the instance I and several *tgds* are used to describe the transitions of the TM. The query Q describes the halting condition. It is then shown that the Turing Machine halts iff $\Sigma, I \models Q$ holds. When the machine doesn't halt, one can construct a counter-model M for $\Sigma, I \models Q$. For this model $M \models \Sigma$ and $I \subseteq M$ but $M \not\models Q$. This construction is defined by the straightforwards encoding of the infinite run of the TM. Even though the same proof cannot be used to prove **FINITE-BCQ-UNDER-TGDs** undecidable the following theorem was established:

Theorem 7. *[PS14]* ***FINITE-BCQ-UNDER-TGDs*** *is undecidable.*

Proof Idea. The main idea is that **co-HALTING** is reduced to **FINITE-CQ-UNDER-TGDs**. The initial configuration of the TM is encoded in the instance I and the

transitions of the TM are encoded by the tgds. The atoms $state(x, q), cursor(x, p)$ and $contains(x, y, s)$ are used to represent a configuration of the turing machine. Using the mentioned atoms one can express that at some time instant x, the TM is in state q, the cursor is in position p and the tape content of tape cell y is s. The successor relation $next(x, x')$ is defined that can be applied to time instants and tape positions. Now back to the adaption from the original transformation:

1. A relation $smaller(\cdot, \cdot)$ is introduced and used to encode the transitive closure of $next(\cdot, \cdot)$.
2. The query Q, remember, this was prior used to encode the halting condition, now asks for $smaller(x, x)$, i.e., if there exists some "loop" in the time instants.

$I, \Sigma \models_f Q$ holds iff the TM does not halt can be shown. Assuming the TM halts, a simple countermodel M in form of the natural encoding of the halting run of the TM can be found. Suppose that the TM does not halt. Then every model M of I, Σ contains an encoding of the infinite number of steps in the non-halting run of the TM. Now we use our assumption that M is finite and every step is identified by some time instant. Thus at least one symbol a is used to encode more than one time instant (which results in the loop). Thus $smaller(a, a) \in M$. If $smaller(x, x)$ occurs we have a (non-halting) run of the TM. Since each step(state, cursor position and cell content) is identified by some time instant and M is finite. □

For completing step two and three of our procedure, we need to strengthen the undecidability result from 7 to atoms with arity two and restrict the set of tgds in **FINITE-CQ-UNDER-TGDs** to a singleton. Notice that atoms of arity two are just a different representation of triples. $p(s, o) \sim (s, p, o)$.

Theorem 8. *[PS14]* ***FINITE-BCQ-UNDER-TGDs*** *is undecidable, even if the arity of every relation symbol is at most two and even if* Σ *consits of a single tgd.*

Proof Idea. To construct a single tgd τ from Σ all antecedents of the tgds in Σ are combined into one antecedent in τ. The variables of the various antecedents are renamed. The same is done for the consequent of τ. The implication is additionally modified: Switches are introduced such that for every tgd $\tau_i \in \Sigma$. If the i-th switch is turned-on, every switch $j \neq i$, may be turned off which means that τ_j is trivially satisfied. This switch idea models the various implications in only implication.

To only use binary atoms, every atom of arity $k > 2$ is replaced by k binary atoms such that a chain of equivalences hold: for any such binary atom in the tgd or query, there exists a homomorphism into an instance I iff the homomorphism can be extended to map all k atoms into I iff this homomorphism is also a homomorphism in the original non-binary case. □

Having the strenghtened version of **FINITE-CQ-UNDER-TGDs** we can now prove that **CONTAINMENT**$[\emptyset, \{\pi\}]$ is undecidable.

Theorem 9. *[PS14]* ***CONTAINMENT***$[\emptyset, \{\pi\}]$ *is undecidable.*

Proof Idea. Assume an arbitrary instance of **FINITE-CQ-UNDER-TGDs** containing only a single tgd. We construct our instance of **CONTAINMENT**$[\emptyset, \{\pi\}]$ in the following way: Let T_1 be a wdPT and T_2 be a pwdPT (T_2, X) each consisting of a root node, with one child node. Both root nodes contains the antecedent of the single tgd τ and the instance I. The root r_2 of T_2 contains in addition another copy of the antecedent of τ, such that the variables in the antecedent are realized by existential variables in $evars(r_2)$. The consequent of the tgd is contained in the child nodes n_1, n_2 of r_1 and r_2. The child node n_1 in T_1 contains the query. The child node n_2 in T_2 contains in addition another copy of the consequent of τ realized by existential variables in $evars(n_2)$. There are auxiliary graph patterns in r_1 and n_1 which deal with the lack of projection. The construction ensures that $T_1 \sqsubseteq (T_2, X)$ holds. Hence the only reason for $T_1 \not\sqsubseteq (T_2, X)$ is that for some RDF graph G, we have the following situation: Some solution $\mu \in [\![T_1]\!]_G$ sends the root into G but cannot be extended to n_1, while in (T_2, X) every extension of μ can be further extended to the existential variables in the root to send also the child node n_2 into G. The following three facts are deduced:

1. Q is not satisfied by G: indeed, n_2 consists of triples from n_1 plus the triples encoding the CQ Q. Since n_2 can be mapped into G by an extension of μ this is also true for all triple patterns in n_1 excpets for those encoding Q.

2. G satisfies τ: indeed, recall that T_2 uses existential variables to encode a copy of the antecedent of *tau* in the root and a copy of the consequent of *tau* in n_2 respectively. We are assuming that every mapping on $vars(r_2)$ that maps the root into G can be extended to the existential variables in n_2 s.t. n_2 is mapped into G. Hence, G satisfies τ by the homomorphism criterion.

3. I must be contained in G, since we are assuming that μ sends the root of both, T_1 and T_2 into G.

It can be shown that G provides a countermodel for $I, \tau \models_f Q$. □

3.4 Equivalence

When looking at the equivalence table 3.2, it is not easy to distinguish the decidable cases from the undecidable ones: Even though **CONTAINMENT**$[S_1, S_2]$ becomes undecidable iff $\pi \in S_2$, **EQUIVALENCE**$[\{\pi, \cup\}, \emptyset]$ is decidable. To keep the number of proofs to an absolute minimum, the fact that membership results propagate to the more special cases and the hardness results to the more general cases is made use of. The following results are proven:

- Π_2^P-membership of **EQUIVALENCE**$[\{\cup, \pi\}, \emptyset]$

- Π_2^P-hardness of **EQUIVALENCE**$[\{\cup\}, \emptyset]$
- Π_2^P-hardness of **EQUIVALENCE**$[\{\pi\}, \emptyset]$
- Undecidability of **EQUIVALENCE**$[\{\pi, \cup\}, \{\cup\}]$
- Undecidability of **EQUIVALENCE**$[\{\pi\}, \{\pi\}]$

After the completion of the above proofs we can conclude all the complexity results in the cells of table 3.2 except two:

1. **EQUIVALENCE**$[\emptyset, \emptyset]$: This result was shown in [LPPS12].
2. **EQUIVALENCE**$[\cup, \cup]$ follows immediately from the Π_2^P-membership of the **CONTAINMENT**$[\{\cup\}, \{\cup\}]$ problem and the Π_2^P-hardness of **EQUIVALENCE**$[\{\cup\}, \emptyset]$ to be shown.

Completeness for **EQUIVALENCE**$[\pi, \cup]$ is not established. The hardness result carries over from the proof of **EQUIVALENCE**$[\{\pi\}, \emptyset]$ and **EQUIVALENCE**$[\{\cup\}, \emptyset]$.

We begin with a proof for Π_2^P-membership of **EQUIVALENCE**$[\{\cup, \pi\}, \emptyset]$.

Theorem 10. *[PS14] Let T be a wdPT and (F, X) be a pwdPF. Then $T \equiv (F, X)$ iff.*

1. *$T \sqsubseteq (F, X)$ and*
2. *$(F, X) \subseteq T$.*

Proof. It is obvious that both properties are necessary for equivalence because $T \subseteq (F, X)$ implies $T \sqsubseteq (F, X)$ and if $T \subseteq (F, X)$ and $T \supseteq (F, X)$ are assumed then $T \equiv (F, X)$ holds by definition of equivalence.

It thus remains to show that under assumption of $(F, X) \subseteq T$, $T \sqsubseteq (F, X)$ iff. $T \subseteq (F, X)$ holds. The "only if" direction is trivial as mentioned before. We now sketch the proof of the if direction: Assume $(F, X) \subseteq T$ and $T \sqsubseteq (F, X)$. Now proceed with a proof by contradiction: Assume $T \subseteq (F, X)$ doesn't hold. Thus there exists a graph G, where some solution μ of T is not a solution of $F(, X)$ over G. But there we can find some extension μ' of μ which is a solution of (F, X). But then μ' must include mappings μ didn't, and is thus a proper extension of μ. But then again by condition (2), μ' is a also a solution of T. But this cannot be true because a mapping and its proper extension are both solutions to a wdPT. □

We can easily see that the characterization in Theorem 10 can be transformed into an algorithm which yields the membership proof for **EQUIVALENCE**$[\emptyset, \{\cup, \pi\}]$.

Theorem 11. *[PS14]* ***EQUIVALENCE***$[\emptyset, \{\cup, \pi\}]$ *is in* Π_2^P.

Proof. By Theorem3 deciding the second property is NP-complete and by Theorem6 deciding the first property is Π_2^P-complete rendering the complexity of the algorithm Π_2^P-complete. □

Following up we have the hardness result of
EQUIVALENCE$[\emptyset, \{\cup\}]$ and **EQUIVALENCE**$[\emptyset, \{\pi\}]$.

Theorem 12. *[PS14]* ***EQUIVALENCE***$[\emptyset, \{\cup\}]$ *is* Π_2^P*-hard*

Proof Idea. The same construction as in Theorem 5 can be used to prove the desired result: The same reduction from **3-QSAT**$_{\forall,2}$ is used. Remembering that in this construction a wdPT T_1 and a wdPF F_2 such that ϕ is valid iff $T_1 \subseteq F_2$ holds is constructed. One can not only show $T_1 \subseteq F_2$ but also $T_1 \supseteq F_2$ (iff ϕ is valid of course). This argumentation yields the desired result. □

Theorem 13. *[PS14]* ***EQUIVALENCE***$[\emptyset, \{\pi\}]$ *is* Π_2^P*-hard*

Proof Idea. A reducion from **3-QSAT**$_{\forall,2}$ to **EQUIVALENCE**$[\emptyset, \{\pi\}]$ is needed to obtain the desired result. □

The following two theorems are proven by adapting the reduction from **FINITE-BCQ-UNDER-TGDs** to **CONTAINMENT**$[\emptyset, \{\pi\}]$ in the proof of Theorem 9.

Theorem 14. *[PS14]* ***EQUIVALENCE***$[\{\cup, \pi\}, \{\cup\}]$ *is undecidable.*

Theorem 15. *[PS14]* ***EQUIVALENCE***$[\{\pi\}, \{\pi\}]$ *is undecidable.*

CHAPTER 4

Complexity of well-designed SPARQL with GRAPH and SERVICE

We will introduce an equivalent definition of wdPTs in this section. The main structural difference to Definition 16 is that the nodes in the tree are labelled with a set of relational atoms over a schema instead of a set of triples. We will also define the fragment P_{wdgs} which is crucial for proving the complexity results. P_{wdgs} is a fragment which allows the arbitrary use of the GRAPH and SERVICE operator in any part of the query.

Definition 29 (P_{wdgs})**.** *A pattern Q is in P_{wdgs} if there does not exist a subpattern $Q' = (Q_1\ OPT\ Q_2)$ of Q and a variable $x \in vars(Q_2)$ that occurs in Q outside Q' but not in Q_1 and it adheres to the following grammar:*

$$\begin{aligned} Q &::= Y \mid (Y\ OPT\ R) \mid B \\ Y &::= (Y\ AND\ Y) \mid (SERVICE\ u\ Y) \mid (GRAPH\ u\ Y) \mid B \\ R &::= (R\ OPT\ R) \mid (GRAPH\ u\ R) \mid (SERVICE\ u\ R) \mid B \\ B &::= (u, v, w) \end{aligned}$$

where $u, v, w \in \mathbf{U} \cup \mathbf{V}$. In the first layer we separate the pattern into an AND-part and an OPT-part. In the AND- and OPT-part we can use SERVICE and GRAPH freely.

We will now proceed to define wdPTs where the nodes are labelled with a set of relational atoms.

Definition 30 (wdPTs [BPS15])**.** *A wdPT over a relational schema σ is a tuple $(T, \lambda, \overline{x})$ such that the following holds:*

1. *T is a tree rooted in a distinguished node r, the root and λ maps each node t in T to a set of relational atoms over σ.*

2. *For every variable y that appears in T, the set of nodes of T where y is mentioned is connected.*

3. *We have that $\overline{x}$ is a tuple of distinct variables occurring in T. They are the free variables of the wdPT.*

Definition 31. *A wdPT $(T, \lambda, \overline{x})$ is called projection-free if $\overline{x}$ contains all variables mentioned in T.*

Definition 32. *Assume $p = (T, \lambda, \overline{x})$ is a wdPT over σ. We write r to denote the root of T. Given a subtree T' of T rooted in r we define $q_{T'}$ to be the CQ $Y \leftarrow R_1(\overline{v_1}), \ldots, R_m(\overline{v}_m)$, where the $R_i(\overline{v_i})$'s are the reational atoms that label the nodes of T', i.e.,*

$$\{R_1(\overline{v}_1), \ldots, R_m(\overline{v_m})\} = \bigcup_{t \in T'} \lambda(t)$$

and $\overline{y}$ are all the variables that are mentioned in T'.

The main idea of this equivalent but different semantics of a wdPT is to look at each subtree T' of T rooted in r. As mentioned above, each of them describes a pattern, i.e., the conjunctive query CQ q'_T. A mapping h satisfies (T, λ) over a database D if h satisfies the pattern defined by a subtree T' and there is no subtree T'' which is bigger than T' and h can be extended to satisfy T''.

Definition 33 (Semantics of wdPTs [BPS15])**.** *Let $p = (T, \lambda, \overline{x})$ be a wdPT and D a database over σ.*

1. *A homomorphism from p to D is a partial mapping $h : X \to U$, where X is an infinite set of a variables and U an infinite set of constants, for which it is the case that there is a subtree T' of T rooted in r such that $h \in q_{T'}(D)$.*

2. *The homomorphism h is maximal if there is no homomorphism h' from p to D such that $h \sqsubset h'$.*

The evaluation of wdPT $p = (T, \lambda, \overline{x})$ over D denoted $p(D)$, corresponds to all mappings of the form $h_{\overline{x}}$, such that h is a maximal homomorphism from p to D.

It is important to notice that wdPTs properly extend CQs. Given a CQ $q(x)$ of the form $X \leftarrow R_1(v_1), \ldots, R_m(v_m)$ it is easy to see that $q(x)$ is equivalent to the wdPT $p = (T, \lambda, x)$, where T consists of a single node r and $\lambda(r) = \{R_1(v_1), \ldots, R_m(v_m)\}$. In other words, $q(D) = p(D)$ for each database D. Further on we will not distinguish

between a CQ and the single node wdPT that represents it. wdPTs can on the other hand represent interesting properties that cannot be expressed as CQs, namely the optional matching feature.

To capture the UNION operator the definition of well-designed pattern trees need to be modified.

Definition 34 (Unions of wdPTs or well-designed pattern Forests (wdPF))**.** *A Union of wdPTs is an expression ϕ of the form $\bigcup_{1\leq i\leq n} p_i$, where each p_i is a wdPT over σ. We denote $\varphi(D)$ as the evaluation of ϕ over database D. It corresponds to the set $\bigcup_{1\leq i\leq n} p_i(D)$. Unions of wdPTs are also called well-designed pattern forests(WDPF).*

First we are going to show an easy example of how to translate a graph pattern only using AND and triple patterns to a conjunctive query. Then we propose a polynomial time translation from a graph pattern $P \in P_{wdgs}$ to $Q \in P_{wd}$. For this translation we need to construct a special database and a wdPT depending on the original query. After we established the translation we prove the equivalence of P and Q in Theorem 16. The last section deals with the problem **EVAL**(P_{wdgs}.

4.1 Translations to well-designed pattern forests

It is an easy observation that if we restrict SPARQL to the AND operator, we simply get conjunctive queries without existentially quantified variables. We will illustrate this in Example 5.

Example 5. *Consider the following SPARQL default Graph G in a dataset DS with the query Q in $SPARQL[\wedge]$*

$$G = \{(a, a, a), (b, c, c), (b, c, a)\}$$
$$Q = (x, y, a)\ AND\ (z, c, c)\ AND\ (x, c, y)$$

Let T be a 3-ary relation, $D = \{T\}$ be the following database and CQ be the following conjunctive query:

$$T = \{(a, a, a), (b, c, c), (b, c, a)\}$$
$$CQ = ans(x, y, z) \leftarrow T(x, y, a), T(z, c, c), T(x, c, y)$$

It is easy to see that the mappings in $CQ(D)$ are the same as in $[\![Q]\!]_G^{DS}$.

Similar to Example 5 we are going to transform the dataset and query not into conjunctive queries but an extension of them: well-designed pattern trees. We will define a polynomial time translation from a graph pattern $P \in P_{wdgs}$ to a well-designed pattern tree. This enables us to use well known algorithms for which the computational complexity is known.

4.1.1 Creating the database

We first describe the function *data* which transforms a dataset into the database.

Consider the function $data : DS \mapsto D$, where DS is a dataset and D is a database. *data* then is defined as follows: Let DS be an arbitrary dataset and u_{DS} the URI such that $ep(u_{DS}) = DS$.

$$DS = \{(def, G), (u_1, G_1), \ldots, (u_n, G_n)\}$$

The output of *data* is the database $D = \{T, LOC\}$ where T is a 5-ary relation containing all the triples of a graph, the corresponding graph URI and the dataset URI. The binary relation LOC captures all graph URIs and their corresponding dataset URI. For our dataset this would mean, assuming

$$\begin{aligned} G &= \{(x_1, y_1, z_1), \ldots, (x_a, y_a, z_a)\}, \\ G_1 &= \{(x_{11}, y_{11}, z_{11}), \ldots, (x_{1b}, y_{1b}, z_{1b})\}, \ldots, \\ G_n &= \{(x_{n1}, y_{n1}, z_{n1}), \ldots, (x_{nc}, y_{nc}, z_{nc})\} \end{aligned}$$

that we construct our output database D as follows:

$$\begin{aligned} D = \{&T(u_{DS}, def, x_1, y_1, z_1), T(u_{DS}, def, x_a, y_a, z_a), \ldots, T(u_{DS}, u_1, x_{11}, y_{11}, z_{11}), \ldots, \\ &T(u_{DS}, u_1, x_{1b}, y_{1b}, z_{1b}), \ldots, T(u_{DS}, u_n, x_{n1}, y_{n1}, z_{n1}), \ldots T(u_{DS}, u_n, x_{nc}, y_{nc}, z_{nc}), \\ &LOC(u_{DS}, def), LOC(u_{DS}, u_1), \ldots LOC(u_{DS}, u_n)\}. \end{aligned}$$

4.1.2 Transforming the pattern to a wdPT

We proceed in defining the function *trans* which will in polynomial time transform a graph pattern in P_{wdgs} into an equivalent well-designed pattern tree without the SERVICE and GRAPH operators. Lemma 3 concludes that the output pattern and the input pattern are equivalent.

The transformation function $trans : P \times \mathbf{U} \cup \{\mathbf{V}\} \times \mathbf{U} \cup \{def\} \cup \{\mathbf{V}\} \mapsto Q$ takes three parameters as input: P is a graph pattern in OPT normal form which allows the usage of AND, OPT, GRAPH, SERVICE and UNION, $\mathbf{U} \cup \mathbf{V}$ is the infinite set of URIs with the infinite set of variables and $\mathbf{U} \cup \{def\} \cup \mathbf{V}$ is the infinite set of URIs containing the def identifier conjoined with the infinite set of variables. The output Q is a well-designed pattern tree.

Assume the input (P, ds, g) and let each of the parameters be arbitrary.

1. If P is a triple pattern (u, v, w),

$$trans(P, ds, g) = vars(ds, g, u, v, w) \leftarrow T(ds, g, u, v, w).$$

2. If P is (P_1 AND P_2), let

$$O_1 \leftarrow q_1 = trans(P_1, ds, g),$$
$$O_2 \leftarrow q_2 = trans(P_2, ds, g),$$
$$trans(P, ds, g) = O_1 \cup O_2 \leftarrow q_1, q_2.$$

3. If P is (P_1 OPT P_2), let

$$(T_1, \lambda_1, x_1) = trans(P_1, ds, g) \text{ and}$$
$$(T_2, \lambda_2, x_2) = trans(P_2, ds, g).$$

   $trans(P, ds, g) = (T, \lambda, x)$ for which $T = T_1 \cup T_2 \cup (r_1, r_2)$ where r_1, r_2 are the roots of T_1, T_2 respectively, $\lambda = \lambda_1 \cup \lambda_2$ and $x = x_1 \cup x_2$.

4. If P is (GRAPH u P_1), let
   $(T_1, \lambda, x_1) = trans(P_1, ds, u)$. Assuming r_1 is the root of T_1, and $\lambda(r_1) = q_1$ we define
$$\lambda'(x) = \begin{cases} q_1, LOC(u, ds), LOC(g, ds) & \text{if } x = r_1 \\ \lambda(x) & \text{otherwise} \end{cases}$$
   and $trans(P, ds, g) = (T_1, \lambda', x_1)$.

5. P is of the form (SERVICE u P_1). Case distinction:

   a) If $u \in \mathbf{U}$ and $u \notin dom(ep)$: $trans(P, ds, g) = \{\} \leftarrow$.

   b) Otherwise let $(T_1, \lambda, x_1) = trans(P_1, u, def)$. Assuming r_1 is the root of T_1, and $\lambda(r_1) = q_1$ we define
$$\lambda'(x) = \begin{cases} q_1, LOC(def, u), LOC(g, ds) & \text{if } x = r_1 \\ \lambda(x) & \text{otherwise} \end{cases}$$
   and $trans(P, ds, g) = (T_1, \lambda', x_1)$.

Observe that the function is well-defined since only patterns P_{wdgs} are considered: This allows us to argue over conjunctive queries for the case P is P_1 AND P_2 as the AND-operator may not occur in the scope of an OPT-operator in the fragment P_{wdgs}.
We need to prove Lemma 3 first towards our goal to show that for all datasets DS identified by URI ds and graph patterns P the following property holds: Let $Q = trans(P, def, ds)$ and $D = \bigcup_{c \in dom(ep)} data(ep(c))$. Then $Q(D) = [\![P]\!]^{DS}_{graph(def,DS)}$.

Lemma 3. *Let $P \in P_{wdgs}$, DS a dataset so that $ep(ds) = DS$ and G a graph in the dataset DS so that $(g, G) \in DS$. Let $D = \bigcup_{c \in dom(ep)} data(ep(c))$. Let $Q = trans(P, g, ds)$.*

Then

$$Q(D) = \begin{cases} [\![P]\!]^{ep(ds)}_{graph(g,ep(ds))} & \text{if } g \in \mathbf{U}, ds \in \mathbf{U} \\ \bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid & \\ \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds))}, \mu \sim [ds \mapsto ds']\} & \text{if } g \in \mathbf{U}, ds \in \mathbf{V} \\ \bigcup_{g' \in names(ep(ds))} \{\mu \cup [g \mapsto g'] \mid & \\ \mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu \sim [g \mapsto g']\} & \text{if } g \in \mathbf{V}, ds \in \mathbf{U} \\ \bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid & \\ \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} & \text{if } g \in \mathbf{V}, ds \in \mathbf{V} \end{cases}$$

holds.

Proof Idea. The four different cases in the lemma are needed to distinguish if we are currently inside a GRAPH pattern, inside a SERVICE pattern or inside both a GRAPH and a SERVICE pattern where the destination was a variable. The evaluation operator of SPARQL, assuming P is a graph pattern, $[\![P]\!]^{DS}_{G}$ only allows DS to be a dataset and G to be a graph in the dataset DS, we take the union of the mappings to simulate the evaluation of P over all possible datasets and their respective graphs if we are inside both a GRAPH and a SERVICE operator. If the pattern is only inside a GRAPH operator the evaluation of P over all the graphs in the current dataset is simulated. If the pattern is only inside a SERVICE operator the evaluation of P over all datasets in $range(ep)$ is simulated. After receiving each result mapping of P over the corresponding dataset and graph we add an additional mapping to the result mapping, depending again if we are inside a SERVICE operator, a GRAPH operator or inside both: This mapping has the variable(s) of either the GRAPH operator or SERVICE operator or both in the domain. The variable(s) map(s) to the current URI(s) the union operator is looking at. This again could be the URI of a dataset, a graph or both. If this mapping is compatible with our result mapping the union of both mappings is added to the results.

The statement is shown with an induction over the structure of a graph pattern. Because of the construction of P_{wdgs}, which doesn't allow that an OPT operator occurs in the scope of an AND operator we are able to argue over conjunctive queries in the cases where P is a triple pattern or P is of the form $P = (P_1 \text{ AND } P_2)$. In all the other cases we need to argue over well-designed pattern trees. In all cases we need to distinguish if we are inside a GRAPH operator, a SERVICE operator, in both or in none. These distinction lead to the big number of case distinctions we need to consider. The cases were P is a triple pattern or a pattern of the form $P = (P_1 \text{ AND } P_2)$ are described in Example5. The only difference is that we now have to consider the cases where we are inside a GRAPH operator, a SERVICE operator or both. For this reason we are using

5-tuples in our T relation of our database, where the first two positions of the tuple describe the dataset we are currently in and the graph of the dataset we want to evaluate the tuple over. For the case that P is (P_1 OPT P_2) we merge the wdPT of P_1 and the wdPT of P_2 so that the roots of the two wdPTs are connected. We then use the semantics of wdPTs to show that this "joining" of trees models the semantics of OPT. The GRAPH and the SERVICE operators are similarly designed: The query P_1 inside the GRAPH or SERVICE operator is evaluated by $trans(P_1, ds, u)$ for GRAPH (ds is the current dataset URI) and $trans(P_1, u, def)$ for SERVICE to a wdPT Q_1. We add two relational atoms to the root of Q_1: For the GRAPH operator $LOC(u, ds)$ and $LOC(g, ds)$ are added and needed in the proof to justify that if u and ds are URIs there is a dataset ds so that $ds \in dom(ep)$ and $u \in names(ds)$. $LOC(g, ds)$ is needed in the case that g is a variable and we have a nested occurrence of GRAPH (e.g. (GRAPH g (GRAPH u P_1)). The mapping of $\mu(g)$ would otherwise be lost in the query. The same arguments hold for the two relational atoms added in the case of a SERVICE operator except by semantics of the SERVICE operator $LOC(def, u)$ is needed because SERVICE always changes to the default graph of a dataset. The only real difference in GRAPH and SERVICE is that again by semantics of SERVICE we need to return the empty mapping instead of the empty set when an endpoint is not reachable or it doesn't exist, which is modelled by $u \notin dom(ep)$. We thus need to check the ep function if the URI is in the domain of ep and if it isn't return $\{\} \leftarrow$. □

The full proof of Lemma 3 can be found in the Appendix.

The next step is to show that the result of the *trans* function is indeed a pattern tree, as it was defined by Definition 30.

Lemma 4. *Let $ds \in dom(ep)$, $g \in names(ep(ds))$ and $P \in P_{wdgs}$ be a SPARQL pattern. Then $Q = trans(P, ds, g)$ is a well-designed pattern tree.*

Proof. We prove each property of Definition 30 separately:

1. T is rooted in a distinguished node r, the root and maps each node t in T to a set of relational atoms over σ: This is easy to see as we only use LOC and T in our construction. Also T always has a distinguished node r as root because the only step of *trans* that changes the structure of T is if a subpattern of the form (P_1 OPT P_2) occurs: $Q_1 = (T_1, \lambda_1, x_1) = trans(P_1, ds, g)$, $Q_2 = (T_2, \lambda_2, x_2) = trans(P_2, ds, g)$. It connects the roots of T_1 and T_2, call them r_1, r_2 with an edge (r_1, r_2) making r_1 the new root.

2. For every variable x that appears in T, the tree of the wdPT, the set of nodes of T where x is mentioned is connected. It is important to remember that we assumed that our inputpattern $P \in P_{wdgs}$ and is well-designed. The induction will be over the structure of the subpatterns $\hat{P}$ of P.

   a) Basecase: If $\hat{P}$ is a triple pattern we only return one node, so there can't be any violation of the property.

b) Induction Step: If $\hat{P}$ is (P_1 AND P_2) there can also never be a violation of the property because we assumed that P is in P_{wdgs} and thus OPT never occurs in the scope of AND. We thus also only return a single node making a violation impossible.

c) Induction Step: If $\hat{P}$ is (P_1 OPT P_2): By induction hypothesis $Q_1 = trans(P_1, ds, g)$ and $Q_2 = trans(P_2, ds, g)$ fulfill the property and $P \in P_{wdgs}$ and is a well-designed SPARQL pattern by assumption.

Assume that making T_2 a child of the root of T_1 results in a wdPT Q which does not fulfill the well-designedness property. There must thus be a variable x in T (the roots of T_1 and T_2 get connected) which appears in two different subgraphs of T. This two subgraphs can only be situated in T_1 and T_2 respectively by induction hypothesis. Because these subgraphs are not connected by our assumption we proceed by case distinction and assume

i. T_1 contains a subgraph containing x but the root r_1 does not contain the variable x. There must be a subpattern of P_1 by our construction (P' OPT P'') where P'' contains x but P' doesn't. As (P' OPT P'') is part of P_1 and we have (P_1 OPT P_2) and P_2 contains x we have a contradiction to the assumption that P is a well-designed SPARQL patterns.

ii. T_2 contains a subgraph containing x but the root r_2 does not contain the variable x. This can be shown analogously.

d) Induction Step: If $\hat{P}$ is (GRAPH u P_1): We know that $trans(P_1, ds, u) = (T, \lambda, x)$ is part of the end result Q and by induction hypothesis, we know that (T, λ, x) fulfills the property. Let r_1 be the root of T. $trans(\cdot)$ adds the conjunct $LOC(u, ds)$ and $LOC(g, ds)$ to r_1. W.l.o.g. assume u,g and ds are variables. For every of those three variables it remains to prove that the property holds. Let $x \in \{u, g, ds\}$ and assume that our resulting wdPT Q is not fulfilling the property. There must thus be a node n_1 in Q which doesn't contain x and a node n_2 for which n_1 is a parent which again contains x creating the conflict. By the definition of the function $trans(\cdot)$ we know that there must have been a subpattern ($P'OPTP''$) and both n_1, n_2 must have been created by this subpattern. But this again means, that P' did not contain x and P'' did contain x. Depending on whether $x = u$, $x = g$ or $x = ds$ we have a contradiction:

i. Let $x = u$. Because we assumed $\hat{P}$ = (GRAPH u P_1) we know that P is not well-designed because P_1 contains the subpattern (P' OPT P'').

ii. Let $x = g$. This means that we are inside a graph pattern (GRAPH g $P^{\sim}$) and thus P is not well-designed because P_1 contains the subpattern (P' OPT P'').

iii. Let $x = ds$. This means that we are inside a SERVICE pattern (SERVICE g $P^{\sim}$) and thus P is not well-designed because P_1 contains the subpattern (P' OPT P'').

e) Induction Step: If $\hat{P}$ is (SERVICE u P_1): The proof is analogously to case where $\hat{P}$ is (GRAPHu P_1).

3. The last property is that x is a tuple of distinct variables occurring in T. As we don't use projection and use set operations to merge our free variables in the OPT case, this is an obvious observation. □

We will now write our final result: When we have a graph pattern P in P_{wdgs} with a dataset DS and a function ep we want to transform it into a wdPT Q and database D with our function $trans$ so that $[\![P]\!]^{DS}_{def} = Q(D)$.

Theorem 16. *Let P be a graph pattern in P_{wdgs}, DS a dataset and G a graph in DS. Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $Q = trans(P, ds, g)$ and $D = \bigcup_{c \in dom(ep)} data(c)$. Then $[\![P]\!]^{DS}_{G} = Q(D)$.*

Proof. The database D is the same database that is created in Lemma 3. Use Lemma 3: $DS = ep(ds)$ and $G = graph(g, DS)$ hold and thus $ds, g \in \mathbf{U}$: $[\![P]\!]^{DS}_{G} = Q(D)$ follows. □

4.2 The complexity of evaluating patterns in P_{wdgs}

The fragments $\mathcal{U}_{wdgs}$ and $\mathcal{S}_{wdgs}$ describe extension to the fragment P_{wdgs}.

Definition 35 ($\mathcal{U}_{wdgs}$ and $\mathcal{S}_{wdgs}$)**.** *$\mathcal{U}_{wdgs}$ extends the fragment P_{wdgs} with top level union. $\mathcal{S}_{wdgs}$ extends the fragment $\mathcal{U}_{wdgs}$ with projection over the top level union expression.*

We will now look at the complexity results we receive as corollaries from Theorem 16.

Corollary 1. *The problem* **EVAL**($\mathbf{P_{wdgs}}$) *is coNP-complete.*

Proof. The problem **EVAL** is defined in the following way:

EVAL($\mathcal{L}$)
INPUT: Dataset DS, graph pattern $P \in \mathcal{L}$ and a mapping μ.
QUESTION: Is μ in $[\![P]\!]^{DS}_{def}$.

Hardness of the problem follows immediately from the hardness of **EVAL(P_{wd})**. Assume $ds = ep^{-1}(DS)$. For the membership we propose the following procedure: Use the transformation function on the input graph pattern P to obtain a wdPT Q, and the data function to obtain the database D. More formally: $Q = trans(P, ds, def)$ and $D = \bigcup_{c \in dom(ep)} data(ep(c))$. This transformation is obviously possible in polynomial time if we assume $dom(ep)$ to be finite. We know by Theorem 16 that $[\![P]\!]^{DS}_{def} = Q(D)$. For

$EVAL(P_{wdgs})$ we can use the results in [PAG09], i.e., that the evaluation problem for well-designed pattern trees without projection is coNP-complete and conclude a coNP runtime to check if $\mu \in Q(D)$. □

Corollary 2. *The problem* $\mathbf{EVAL}(\mathcal{U}_{\mathbf{wdgs}})$ *is coNP-complete.*

Proof. Hardness of the problem follows immediately from the hardness of $\mathbf{EVAL}(\mathcal{U}_{wd})$. Assume $ds = ep^{-1}(DS)$. For the membership we propose the following procedure: If we have top-level union in the input graph pattern $P = P_1$ UNION $\cdots$ UNION P_n use the transformation function on all the graph patterns of P, i.e., $P_1, \ldots, P_n$ to obtain a wdPF. More formally: $\bigcup_{i=1,\ldots,n} Q_i = trans(P_i, ds, def)$ and $D = \bigcup_{c \in dom(ep)} data(ep(c))$. This transformation is obviously possible in polynomial time if we assume $dom(ep)$ to be finite. We know by Theorem 16 that $[\![P]\!]^{DS}_{def} = Q(D)$. For $EVAL(\mathcal{U}_{wdgs})$ we can use the results in [PAG09], i.e., that the evaluation problem for well-designed pattern trees without projection is coNP-complete and conclude a coNP runtime to check if $\mu \in Q(D)$. □

Corollary 3. *The problem* $\mathbf{EVAL}(\mathcal{S}_{\mathbf{wdgs}})$ *is* Π^P_2*-complete.*

Proof. Hardness of the problem follows immediately from the hardness of $\mathbf{EVAL}(\mathcal{S}_{wd})$. Assume $ds = ep^{-1}(DS)$. For the membership we propose the following procedure: If we have top-level projection of the input graph pattern P we use the transformation function on the wdPT. More formally: $\bigcup_{i=1,\ldots,n} Q_i = trans(P_i, ds, def)$ and $D = \bigcup_{c \in dom(ep)} data(ep(c))$. This transformation is obviously possible in polynomial time if we assume $dom(ep)$ to be finite. We know by Theorem 16 that $[\![P]\!]^{DS}_{def} = Q(D)$. For $EVAL(\mathcal{S}_{wdgs})$ we can use the results in [LPPS13], i.e., that the evaluation problem for well-designed pattern trees with projection is Π^P_2-complete and conclude a Π^P_2 runtime to check if $\mu \in Q(D)$. □

CHAPTER 5

The SERVICE-operator in Practice

The first part of this chapter is a summary of [BAACP13, p. 4-7]. In the second part of this chapter we discuss the difference of the notions we introduced in the first part. In order to get a deeper understanding of the SERVICE operator it is mandatory to understand which problems occur when evaluating the SERVICE operator. A direct implementation of the SERVICE operator based of the semantics is infeasible in practice. Given (SERVICE x P_1), if x is not restricted to a finite set, we would have to evaluate P_1 over every possible SPARQL endpoint in $dom(ep)$. This is obviously impossible. To ensure that P_1 only gets evaluated over a finite set of URIs, x needs to be limited to exactly those. In the W3C standard only indications are provided on how to evaluate the service operator [SHb] when the location is a variable and not an URI. In [BAACP13] the authors deal with this issue by providing a notion of boundedness. In order to demonstrate how one could evaluate a service operator using a variable to evaluate a query over more than one endpoint the following example is given:

Example 6 ([BAACP13])**.** *Let G be an RDF graph that uses triples of the form $(a, service\_address, b)$ with the intention to express that b is a SPARQL endpoint URI with name a. Then we consider the following query P over the graph G in the dataset DS:*

$$P = ((x, service\_address, y)\ AND\ (SERVICE\ y\ (z_n, email, z_e)))$$

It is easy to see that P is used to compute the list of names and email addresses that can be retrieved from the SPARQL endpoints stored in the RDF graph G through the service_address triple. The whole point of this example is to point out that there is a simple practical way to evaluate P over G that is also feasible: By evaluating $[\![(x, service\_address, y)]\!]_G^{DS}$ first and then for every mapping μ in this set we further evaluate $[\![(SERVICE\ a\ (z_n, email, z_e)]\!]_G^{DS}$, where $a = \mu(y)$.

Throughout the chapter we will provide four different definitions on how the destinations of a SERVICE-operator can be bounded within a graph pattern if the destination is a variable. Those definitions were first introduced in [BAACP13].

1. Boundedness: Boundedness is a naïve semantic approach to the problem. After formally defining the property, we will show that deciding whether a variable is bounded in a graph pattern is undecidable and thus not feasible for practical use.
2. Strong boundedness: Strong boundedness is a syntactical approach to the problem. We will, by defining the property provide a recursive procedure on how to decide which variables in a graph pattern are strongly bounded. It will also be shown that if a variable is strongly bounded, it is bounded aswell. Although this procedure would be feasible for practical use complexity-wise, it is not able to decide whether a graph pattern can be evaluated in practice. We will provide an example in form of a graph pattern which is feasible to be evaluated in practice but not strongly bounded.
3. Service-boundedness: Service-boundedness would be the optimal solution for deciding which graph patterns can be evaluated in practice. If a pattern is service-bounded, it can be evaluated in practice. The problem however is, that the definition builds on the definition of boundedness which is undecidable. Therefore service-boundedness is not feasible for practical use.
4. Service-safeness: The definition of service-safeness builds on the definition of strong boundedness. service-safeness is easy to decide and we will show that if a pattern is service-safe, it is also service-bounded. This solution should be used in practice for the original problem.

In the last section we will discuss the difference of boundedness and strong boundedness (and thus service-boundedness and service-safeness because the latter definitions build on former definitions).

5.1 The four different ways to bind the destination of a SERVICE-operator

To describe boundedness we need three definitions namely the domain of a graph, a dataset and a graph pattern.

Definition 36 (Domain of a graph, a dataset and a graph pattern,[BAACP13])**.** *The domain of a graph G denoted $dom(G)$ is defined as $dom(G) = \bigcup_{(u,v,w)\in G} vars(u,v,w)$. The domain of a dataset DS, denoted $dom(DS)$ is defined as $dom(DS) = \bigcup_{G\in names(DS)} dom(G)$. The domain of a graph pattern P is denoted $dom(P)$ and refers to the set of URIs that are mentioned in P.*

Boundedness of a variable x in a graph pattern P makes sure that the variable is always in the domain of every solution μ and the image $\mu(x)$ is either in the domain of the dataset P gets evaluated over, it is a graph name or it is in the domain of the pattern.

Definition 37 (Boundedness,[BAACP13]). *Let P be a graph pattern and $x \in var(P)$. Then x is bounded in P if the following condition holds: For every dataset DS, every graph G in DS and every $\mu \in [\![P]\!]_G^{DS}$:*

$$x \in dom(\mu) \text{ and } \mu(x) \in (dom(DS) \cup names(DS) \cup dom(P)).$$

Resulting from this definition a very naive way to ensure that a graph pattern P can be evaluated in practice seems to arise: Assuming we want to evaluate a subpattern (SERVICE x P_1) of P we require x to be bounded in P. We can then define the problem for deciding if a variable is bounded in a graph pattern P:

BOUND IN PATTERN
INPUT: A graph pattern P and a variable $x \in var(P)$.
QUESTION: Is x bounded in P?

Unfortunately **BOUND IN PATTERN** is undecidable which can be shown by reducing from **SPARQL SAT** to **BOUND IN PATTERN**.

SPARQL SAT
INPUT: A graph pattern P.
QUESTION: Does a dataset DS and a graph G in DS exist such that $[\![P]\!]_G^{DS}$?

It is a well known result that **SPARQL SAT** is undecidable [AG08].

Theorem 17 ([BAACP13]). ***BOUND IN PATTERN*** *is undecidable.*

Proof. By providing a reduction from the **SPARQL SAT** problem to the **VARIABLE BOUND IN PATTERN** Problem we will be able to prove the theorem. Let P be a graph pattern, i.e., an arbitrary instance of **SPARQL SAT** and x, y, z variables not mentioned in P. Then define the graph pattern Q, i.e. the instance of **VARIABLE BOUND IN GRAPH** pattern as follows: $Q = ((x, y, z)$ UNION $P)$ and choose x to be the potentially bounded variable. It remains to show that x is bounded in Q if and only if P is not satisfiable.
($\Rightarrow$):
Assume now that the variable x is bounded in Q, i.e., for every RDF graph G in DS and every $\mu \in [\![Q]\!]_G^{DS} : x \in dom(\mu)$ and $\mu(x) \in (dom(DS) \cup names(DS) \cup dom(P))$. Let DS be an arbitrary dataset and let G be an arbitrary graph in DS. Distinguish the following two cases:

1. $[\![Q]\!]_G^{DS} = \emptyset$. Because of $[\![Q]\!]_G^{DS} = \emptyset$, we can instantly see that P is unsatisfiable.
2. $[\![Q]\!]_G^{DS} \neq \emptyset$.
   Let $\mu \in [\![Q]\!]_G^{DS}$ be arbitrary. We can instantly see that $x \in dom(\mu)$ must hold. Because by construction P doesn't contain x, $\mu \notin [\![P]\!]_G^{DS}$. Thus P is unsatisfiable.

($\Leftarrow$):
Assume now that P is not satisfiable. Let DS be an arbitrary dataset and let G be an arbitrary graph in DS. Because of our initial assumption $[\![P]\!]_G^{DS} = \emptyset$. We will now further distinguish between two cases:

1. $[\![Q]\!]_G^{DS} = \emptyset$. Then x is trivially bounded in Q.
2. $[\![Q]\!]_G^{DS} \neq \emptyset$.
   Let $\mu \in [\![Q]\!]_G^{DS}$ be arbitrary. We can instantly see by construction of Q that $x \in dom(\mu)$ must hold. By SPARQL semantics $\mu(x) \in dom(DS)$. Thus x is bounded in Q. □

As deciding boundedness for a variable is undecidable the notion of *strong boundedness* is introduced, which is a syntactic condition and efficiently verifiable.

Definition 38 (Strong Boundedness [BAACP13])**.** *Let P be a graph pattern. Then the set of strongly bounded variables in P, denoted by $SB(P)$, is recursively defined as follows.*

- *if $P = t$, where t is a triple pattern, then $SB(P) = vars(t)$;*
- *if $P = (P_1\ AND\ P_2)$, then $SB(P) = SB(P_1) \cup SB(P_2)$*
- *if $P = (P_1\ UNION\ P_2)$, then $SB(P) = SB(P_1) \cap SB(P_2)$*
- *if $P = (P_1\ OPT\ P_2)$, then $SB(P) = SB(P_1)$*
- *if $P = (GRAPH\ u\ P_1)$, with $u \in U \cup V$, then*

$$SB(P) = \begin{cases} SB(P_1) & u \in U \\ SB(P_1) \cup \{u\} & u \in V \end{cases}$$

- *if $P = (SERVICE\ u\ P_1)$, with $u \in U \cup V$, then $SB(P) = \emptyset$.*

It is a simple observation that this recursive definition collects a set of variables that are guaranteed to be bounded in P. The following proposition documents this observation.

Proposition 4 ([BAACP13])**.** *For every graph pattern P and a variable $x \in var(P)$, if $x \in SB(P)$, then x is bounded in P.*

The proof is a very straight forward induction and can be found in [BAACP13, Appendix A].

We notice that this proposition is not an if and only if statement and hence we may be able to provide a pattern P where variables $x \in vars(P)$ exist, which are bounded but not strongly bounded. Furthermore another example is provided which makes things more complicated. In Example 7 we provide a graph pattern P where a variable in the destination of a SERVICE-operator occurs which is neither bounded nor strongly bounded. We will then provide a plan on how to evaluate P and thus show that the definition of neither boundedness nor strongly boundedness is sufficient for practical usage.

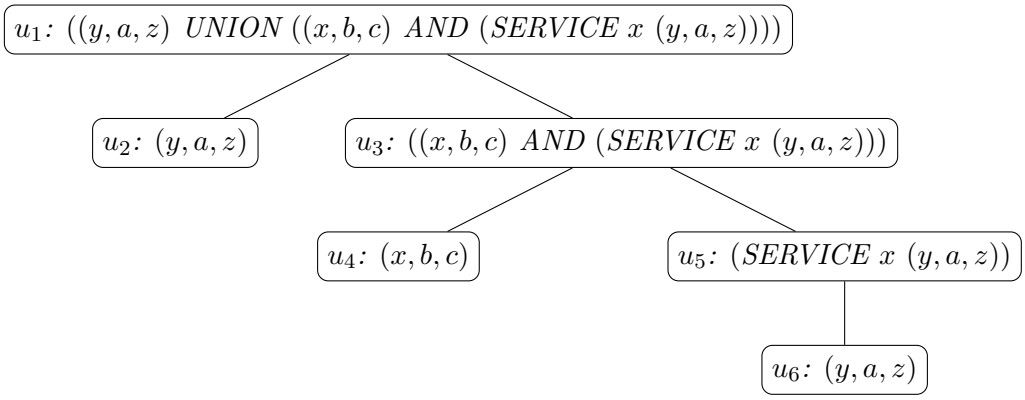
Example 7. *Consider the following graph pattern:*

$$P_1 = [(x, service\_description, z)\ UNION\ ((x, service\_address, y)\ AND\ (SERVICE\ y\ (x_n, email, x_e)))]$$

The variables x and z store the name of a SPARQL endpoint and a description of its functionalities through the service_description triple. The variables x and y store the name of a SPARQL endpoint and the URI where it is located through the service_address triple. The problem is, that variable y is neither bounded nor strongly bounded in P_1. However we can still easily evaluate the pattern by assuming a dataset DS and an RDF graph G in DS:
Compute $[\![(x, service\_description, z)]\!]_G^{DS}$, then compute
$[\![(x, service\_addres, y)]\!]_G^{DS}$ and finally for every $\mu \in [\![(x, service\_addres, y)]\!]_G^{DS}$, compute $[\![(SERVICE\ a\ (x_n, email, x_e))]\!]_G^{DS}$ with $a = \mu(y)$. We can easily see that y is bounded and strongly bounded in the subpattern
$((x, service\_address, y)\ AND\ (SERVICE\ y\ (x_n, email, x_e)))$ of P_1 and thus the evaluation is possible.

To describe a condition that ensures all SPARQL queries containing the SERVICE operator can be evaluated in practice, the definition of Service-Boundedness is introduced. The definition of service-boundedness uses a parse tree to make sure that our evaluation takes bounded subpatterns into account.

Example 8. *[[BAACP13]] Parse tree $\mathcal{T}(Q)$ for the graph pattern*
$Q = ((y, a, z)\ UNION\ ((x, b, c)\ AND\ (SERVICE\ x\ (y, a, z))))$.



Definition 39 (Parse Tree, [BAACP13])**.** *A parse tree of a graph pattern P, $\mathcal{T}(P)$ is a tree where each node is a sub-pattern of P. Each node has an identifier. In the parse tree the child relation is used to store the structure of the sub-patterns of the graph pattern. The root of the parse tree contains the pattern P. Then, in the child(ren) of P the pattern is split up into the respective sub-pattern(s). This is done recursively until a node contains only a triple.*

In Example 8 a parse tree can be found

Using the definition of a Parse Tree, Service-Boundedness can be defined.

Definition 40 ([BAACP13])**.** *A graph pattern P is service-bounded if for every node u of $\mathcal{T}(P)$ with label $(SERVICE\ x\ P_1)$, it holds that*

1. *there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and x is bounded in P_2,*
2. *P_1 is service-bounded.*

Corresponding to this definition we will introduce the **SERVICE BOUND** problem:

SERVICE BOUND
INPUT: A graph pattern P.
QUESTION: Is P service-bounded?

Intuitively one can already imagine that deciding the **SERVICE BOUND** problem is undecidable because it uses boundedness in it.

Theorem 18 ([BAACP13])**.** ***SERVICE BOUND*** *is undecidable.*

Proof. By providing a reduction from the **SPARQL SAT** problem to the **SERVICE BOUND** problem we will be able to show undecidability. Let P be a graph pattern, i.e., an arbitrary instance of **SPARQL SAT** and x, y, z, x', y', z' variables not mentioned in P. Also assume that P does not mention the operator SERVICE which is not required to make the SPARQL satisfiability problem undecidable. Then define the graph pattern Q, i.e., the instance of **SERVICE BOUND** as:

$$Q = (((x, y, z) \text{ UNION } P) \text{ AND } (\text{SERVICE } x \ (x', y', z'))).$$

It remains to show that Q is service-bounded if and only if P is not satisfiable.

($\Leftarrow$) If P is not satisfiable, then Q is equivalent to the pattern:

$$Q' = ((x, y, z)) \text{ AND } (\text{SERVICE } x \ (x', y', z')).$$

But Q' is service bounded because x is bounded in Q': Assume an arbitrary dataset DS, let G be in DS. Then for any $\mu \in [\![Q']\!]_G^{DS}$, $x \in dom(\mu)$ and for sure $\mu(x) \in (dom(DS) \cup names(DS) \cup dom(P))$ by semantics of SPARQL and the fact that x occurs in the triple pattern (x, y, z).

($\Rightarrow$) Assume that P is satisfiable. Then we know that variable $x \notin vars(P)$ and thus we have that x is not a bounded variable in Q, and thus because x occurs in a service operator, Q is not service bounded. □

As the problem of undecidability prevails with the definition of service-boundedness, we can again use the syntactic condition, i.e., strong boundedness to define service-safeness which is the same as service-boundedness but uses strong boundedness instead of boundedness. We know from Definition 38 that evaluating if a variable is strongly bounded in a graph pattern can be done efficiently.

Definition 41 (Service Safeness [BAACP13])**.** *A graph pattern P is service-safe if for every node u of $\mathcal{T}(P)$ with label $(SERVICE \ x \ P_1)$ it holds that:*

1. *there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $x \in SB(P_2)$.*
2. *P_1 is service safe.*

As corollary to Proposition 1, the following proposition is obtained:

Proposition 5 ([BAACP13])**.** *If a graph pattern P is service-safe, then P is service-bounded.*

5.2 Boundedness and strong boundedness

An interesting question is stirred up through introducing the notions of boundedness and strong boundedness: What is the difference between the two notions? Because boundedness is not equivalent to strong boundedness, there must be patterns which are bounded but not strongly bounded. The following problem could arise: Assume a pattern is bounded but not strongly bounded. Then it is feasible to be evaluated, but the proposed algorithm returns that it is not.

Example 9.

$$P = ((x, a, b)\ OPT\ (x, a, y)).$$

We can see that y is bounded in P because for every RDF graph G in DS and every $\mu \in [\![P]\!]_G^{DS}$ we have that $y \in dom(\mu)$ and $\mu(y) \in (dom(DS))$: Assume we have a $\mu \in [\![P]\!]_G^{DS}$. Then $x \in dom(\mu)$ by semantics of OPT. Assume w.l.o.g. $\mu(x) = c$. Then $(c, a, b) \in G$. But then the mapping $\{x \mapsto c, y \mapsto b\} \in [\![P]\!]_G^{DS}$. The set of strongly bounded variables, i.e., $SB(P) = \{x\}$ by Definition 38 contains only x.

Example 9 shows that there are patterns which are bounded but not strongly bounded. One could easily resolve the problem of Example 9 and make y strongly bounded again: It is easy to see that $((x, a, b)$ OPT $(x, a, y)) \equiv ((x, a, b)$ AND $(x, a, y))$ holds. Thus we replace the graph pattern P with the graph pattern $Q = ((x, a, b)$ AND $(x, a, y))$. While preserving the meaning of P, Q also makes sure that y is now strongly bound in Q, i.e., $SB(Q) = \{x, y\}$.

Example 9 also elicits a new question: Could deciding boundedness be feasible in fragments of SPARQL where the problem **EQUIVALENCE** is decidable? An idea for a decision procedure of **BOUND IN PATTERN** is illustrated in Algorithm 5.1. This procedure would not contradict the results of Theorem 17 because in Theorem 17 general SPARQL is considered and it is a well known result that **EQUIVALENCE** is undecidable in general SPARQL.

Algorithm 5.1: BSB

1 INPUT: pattern P and $x \in vars(P)$
2 If $x \in SB(P)$ return true;
3 For all subpatterns (P_1 OPT P_2) of P:
4 If (P_1 OPT P_2) $\equiv$ (P_1 AND P_2) holds, replace (P_1 OPT P_2) with (P_1 AND P_2) in P.
5 Call the resulting graph pattern Q.
6 If ($x \in SB(Q)$) return true;
7 else return false;

CHAPTER 6

Beyond well-designed SPARQL

Although well-designed SPARQL is a fragment that covers a lot of practical SPARQL queries, (50% of the queries over DBpedia that use the OPT-operator [PV11]) many practical queries are not well designed and need to be analyzed. Kaminski and Kostylev found another interesting fragment of SPARQL called weakly well-designed fragment [KK16]. This fragment captures 99% of the queries over DBpedia that use the OPT-operator. There are mainly two use cases of queries that are not well-designed but used in practice.

1. The first practical use of non well-designed patterns are the so called preference patterns: Consider the following example:

   Example 10 (Preference Pattern [KK16])**.**

   $$\begin{aligned} P_1 = \ & \mathit{SELECT}\ x, y\ \ \mathit{WHERE}\ ((x, \mathit{type}, \mathit{person}) \\ & \mathit{OPT}\ (x, \mathit{name}, y)) \\ & \mathit{OPT}\ (x, \mathit{v\_card} : \mathit{name}, y) \end{aligned}$$

   It is obvious that the pattern P in Example 10 is not well-designed because variable y does occur in two unrelated OPT parts of P. The intuitive meaning might seem unclear at the first moment, but looking at the semantics of OPT sheds light on it: If the first OPT-operator does not bind the variable y through the triple (x, name, y) and only then, y is bound through the triple $(x, \mathit{v\_card} : \mathit{name}, y)$. This could be used when we have two relations, in this case *name* and $\mathit{v\_card} : \mathit{name}$, which connect a name to an identifier but we prefer the relation *name* over $\mathit{v\_card} : \mathit{name}$.

2. The second practical use of non well-designed patterns are top level FILTER expressions. For this usage consider the following query:

Example 11 (Top level Filter [KK16])**.**

$$P_2 = \; SELECT\ x, y\ \ WHERE\ ((x, type, person)\ OPT\ (x, name, y))\ FILTER\ (\neg bound(y) \vee \neg(y = Ana)).$$

The query P_2 is not well-designed because the FILTER constraint mentions the variable y, which occurs only in the optional part. The practical intention of the query is to filter for people where the name is not 'Ana' or do only have an id which is bound by variable x.

To capture the two use-cases mentioned in Example 10 and Example 11, well-designed SPARQL is extended to weakly well-designed SPARQL. This new fragment subsumes well-designed queries and it was shown in [KK16] that it has the same complexity of query evaluation as well-designed queries. Also, 99% of the practical queries over DBPedia containing OPT are captured by the weakly-well designed fragment of SPARQL and thus proven to be efficient to evaluate. To define the fragment of weakly well-designed pattern we need to define what it means for a subpattern to be dominated by another subpattern:

Definition 42 ([KK16])**.** *Given a graph pattern P, an occurrence i_1 in P dominates another occurrence i_2 if there exists a OPT-pattern such that i_1 is inside the left argument of the OPT-pattern and i_2 is inside the right argument of the OPT-pattern.*

Now we can proceed to define weakly well-designed patterns.

Definition 43 (Weakly well-designed patterns [KK16])**.** *A pattern P is weakly well-designed (wwd-pattern) if each occurrence i of an OPT-subpattern $(P_1\ OPT\ P_2)$ variables in $vars(P_2) \backslash vars(P_1)$ appear outside i only in*

- *subpatterns whose occurrences are dominated by i, and*
- *constraints of top-level occurrences of FILTER-patterns.*

Checking if a pattern is wwd is very easy computationally:

Proposition 6 ([KK16])**.** *Checking whether a pattern P belongs to the fragment P_{wwd} can be done in time $O(|P|^2)$, where $|P|$ is the length of the string representation of P.*

Proof Idea. In a simple recursive procedure, the top-level occurrences of filters are removed. Then in yet another recursive procedure the first condition of weakly well-designed patterns is checked. □

6.1 OPT-FILTER-Normal Form and Constraint Pattern Trees

When used wd-patterns we can convert them to the so-called OPT-normal form. In the OPT-normal form, all AND- and FILTER- subpatterns are OPT-free and most importantly the pattern can then be naturally represented as a tree. The resulting tree can be used to visualize how to evaluate and optimize the original pattern [LPPS13, PS14]. The tree notation can be generalised to wwd-patterns.

Definition 44 (OPT-FILTER-normal form [KK16])**.** *A pattern P is in OPT-FILTER-normal form (or OF-normal form) if the following grammar can be tested positively:*

$$P ::= F \mid (P \ FILTER \ R) \mid (P \ OPT \ S), \qquad S ::= F \mid (S \ OPT \ S),$$
$$F ::= (B \ FILTER \ R)$$

where B is a set of triple patterns and R is a filter constraint.

As we can see from Definition 44, each triple pattern has a FILTER expression. This is no restriction because one can easily insert a dummy FILTER expression by letting $R = \top$. The sets of triple patterns with FILTER expressions form the bottom layer. On top of the bottom layer there is a combination of OPT and FILTER. The layers cause that each occurrence of a FILTER-pattern in the top layer is top-level. The normal form is AND-free: all conjunctions are expressed via a set of triple patterns.

Definition 45 (Constraint pattern tree (CPT) [KK16])**.** *A constraint pattern tree (CPT) $T(P)$ of a pattern P in OF-normal form is the directed ordered labelled rooted tree, which can be recursively constructed as follows:*

1. *if B is a set of triple patterns then $T(B \ FILTER \ R)$ is a single node v labelled by the pair (B, R);*
2. *if P' is not a set of triple patterns then $T(P' \ FILTER \ R)$ is obtained by adding a special node labelled by R as the last child of the root of $T(P')$;*
3. *$T(P_1 \ OPT \ P_2)$ is the tree obtained from $T(P_1)$ and $T(P_2)$ by adding the root of $T(P_2)$ as the last child of the root of $T(P_1)$.*

Looking at Definition 45 one can see the similarity of CPTs and patterns in OF-normal form: A CPT displays the semantic structure of OPT and FILTER nesting. The next step is to prove that every wwd-pattern can be converted to OF-normal form and can be represented by a CPT, analogously to wd-patterns, which can be transformed into OPT-normal form and thus pattern trees. Towards this goal, the following equivalence is needed:

Proposition 7 ([KK16]). *Let P_1, P_2, P_3 be patterns and R a filter constraint such that $vars(P_2) \cap vars(P_3) \subseteq vars(P_1)$ and $vars(P_2) \cap vars(R) \subseteq vars(P_1)$. Then the following equivalences hold:*

$$(P_1 \ OPT \ P_2) \ AND \ P_3 \equiv (P_1 \ AND \ P_3) \ OPT \ P_2,$$
$$(P_1 \ OPT \ P_2) \ FILTER \ R \equiv (P_1 \ FILTER \ R) \ OPT \ P_2.$$

Using the two equivalences we can achieve our goal stated in Proposition 8.

Proposition 8 ([KK16]). *Each wwd-pattern P is equivalent to a wwd-pattern in OF-normal form of size $O(|P|)$.*

Let $\prec$ be a relation which contains the topological sorting of the nodes in $T(P)$ computed by a depth first search traversal. $v \prec u$ holds if v is visited before u in the search process. Assuming such a relation $\prec$, Proposition 9 provides a condition to decide whether a pattern is weakly well-designed looking at its CPT.

Proposition 9 ([KK16]). *A pattern P in OF-normal form is weakly well-designed iff. for each edge (v, u) in its CPT $T(P)$ every variable $x \in vars(u) \backslash vars(v)$ occurs only in nodes w such that $v \prec w$. The pattern is well-designed iff for every variable x in P the set of all nodes v in $T(P)$ with $x \in vars(v)$ is connected.*

In [KK16] a unique property, which applies only for wwd-patterns was found: Each wwd-pattern is semantically equivalent to a pattern whose corresponding CPT has depth one.

Definition 46 ([KK16]). *A pattern P is in depth-one normal form if it has the structure*

$$(\cdots((B \ op_1 \ S_1) \ op_2 \ S_2) \cdots) \ op_n \ S_n,$$

where B is a set of triple patterns and each $op_i S_i, 1 \leq i \leq n$ is either $OPT \ (B_i \ FILTER \ R_i)$ with B_i a basic pattern and R_i a filter constraint, or just $FILTER \ R_i$.

Towards our goal to show that every wwd-pattern can be brought to the depth-one normal form the following equivalence can be used.

Proposition 10 ([KK16]). *For patterns P_1, P_2, P_3 with $vars(P_1) \cap vars(P_3) \subseteq vars(P_2)$ it holds that*

$$P_1 \ OPT \ (P_2 \ OPT \ P_3) \equiv (P_1 \ OPT \ P_2) \ OPT \ (P_2 \ AND \ P_3).$$

OPT operators in pattern can be nested in two different ways:

1. (P_1 OPT (P_2 OPT P_3)) (OPT-R)

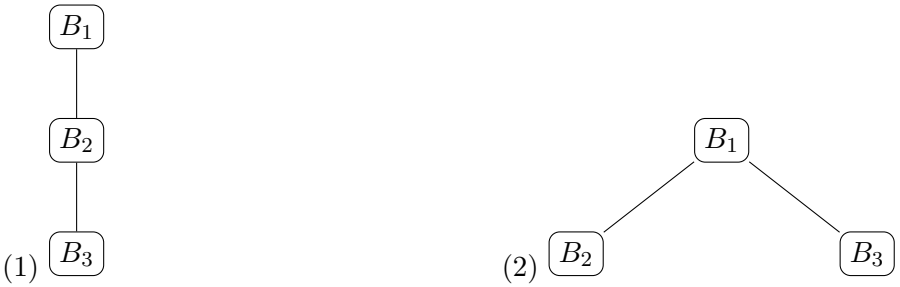


Figure 6.1: (1) is the CPT of $(B_1$ OPT $(B_2$ OPT $B_3))$ and (2) the CPT of $((B_1$ OPT $B_2)$ OPT $B_3)$

2. $((P_1$ OPT $P_2)$ OPT $P_3)$ (OPT-L)

When we then use CPTs to display this patterns using triple patterns we get the following result:

Using the equivalence from left to right keeps the pattern weak well-designed and transforms a weakly well-designed OPT nesting of type (OPT-R) to a nesting type (OPT-L). Looking at the Figure 6.1 we can see that this step reduces the depth by one.

Corollary 4 ([KK16])**.** *Every wwd-pattern is equivalent to a wwd-pattern in depth-one normal form.*

The regular structure of the depth-one normal form might prove attractive in practice but using the equivalence results in an exponential blowup in the size of the pattern. This can be seen in Proposition 10: P_2 gets copied twice in every application of the equivalence.

6.2 Evaluation of wwd-Patterns

The next step is to look at the complexity of the evaluation problem for wwd-patterns and the extensions with union and projection. The goal is to show that in all three cases complexity doesn't change in comparison to wd-patterns. Consider first the formal evaluation problem for a given SPARQL fragment $\mathcal{L}$:

EVAL($\mathcal{L}$)
INPUT: Graph G, query $Q \in \mathcal{L}$ and a mapping μ.
QUESTION: Is μ in $[\![Q]\!]_G$.

The fragment of general SPARQL graph patterns is denoted with $\mathcal{U}$ and it is a well known result that **EVAL($\mathcal{U}$)** is PSPACE-complete [PAG09]. For the fragment of general

queries extended with projection (i.e., $\mathcal{S}$) it was shown that **EVAL($\mathcal{S}$)** is PSPACE-complete [LPPS13].

Similar to Algorithm 3.1, an algorithm for wwd-patterns exists. We first define the potential partial solutions. We know from the previous section that we can transform any wwd-pattern P to a pattern in OF-normal form. This pattern P' in OF-normal corresponds to some CPT. An r-subtree of $T(P')$ is a subtree containing the root of $T(P')$ and all its special children. Every r-subtree obviously corresponds to some wwd-pattern which is obtained by dropping the rightmost arguments of some OPT-subpatterns.

Definition 47 ([KK16])**.** *Let P be a wwd-pattern and P' the corresponding pattern in OF-normal form. A mapping μ is a potential partial solution (or pp-solution for short) to a wwd-pattern P over a graph G if there is an r-subtree $T(P')$ of $T(P)$ such that $dom(\mu) = vars(P')$, $\mu(pat(P')) \subseteq G$ and $\mu \models R$ for the constraint R of any ordinary node in $T(P')$.*

It could happen that several r-subtrees correspond to the same mapping μ for a pattern P over G. We then take the union of all the nodes in exactly those r-subtrees as they are a subtree as well. (They are all connected by the root). From this observation we can see that there exists a unique maximal r-subtree corresponding to a mapping μ which we will from now on denote with $T(P_\mu)$, as this subtree corresponds to a wwd-pattern P_μ. The big difference to partial solutions for wd-patterns is that not every pp-solution can be extended to a real solution. A real solution may not only extend the domain of a pp-solution with previously undefined variables but it might also extend $T(P_\mu)$ to a child that is smaller in the order $\prec$ than some other node which was already in $T(P_\mu)$. But this means that variables bindings might get overridden. The next difference are non-well-designed top-level filters. *pp*-solutions ignore top-level filters as it would be too restrictive as real solutions do not satisfy them either.

Example 12 ([KK16])**.** *Consider the graph $G = \{(1, a, 1), (3, a, 3)\}$ and the wwd-pattern:*

$$P = (((x, a, 1)\ \mathit{OPT}\ (y, a, 2))\ \mathit{FILTER}\ \neg \mathit{bound}(y))\ \mathit{OPT}\ (y, a, 3).$$

Obviously $\neg bound(y)$ is a top level filter and the whole pattern is not well designed as y occurs in the top level filter. Also, the mapping $\mu = \{?X \mapsto 1, ?Y \mapsto 3\}$ is a solution in $[\![P]\!]_G$, but we can see, that $\mu \not\models \neg bound(y)$.

The following characterisation takes care of the two difficulties and describes solutions.

Definition 48 ([KK16])**.** *Given a wwd-pattern P, a node $v \in T(P)$, a graph G and a pp-solution μ of P over G, let $\mu_{|v}$ be the projection $\mu_{|X}$ of μ to the set X of all variables appearing in nodes u of $T(P_\mu)$ such that $u \prec v$.*

$\mu_{|v}$ is in other words the mapping $T(P_\mu)$ until the node v could be considered in the order $\prec$ would traverse the tree $T(P)$.

Lemma 5 ([KK16]). *A mapping μ is a solution to a wwd-pattern P over a graph G iff.*

1. *μ is a pp-solution to P over G.*
2. *for any child v of $T(P_\mu)$ labelled with (B, R) there is no μ' such that $\mu_{|v} \sqsubset \mu'$, $\mu' \models R$, and $\mu'(B) \subseteq G$;*
3. *$\mu_{|s} \models R$ for any special node s in $T(P)$ labelled with R.*

Intuitively we can conclude the following from the lemma: (1) is obvious and as mentioned before, each solution needs to be a pp-solution of P over G. (2) makes sure that every node v which is not added to $T(P_\mu)$ is not addable to the tree "until" that node v "appears" (i.e. in order of $\prec$) first. If it is addable it would mean, that one could use the mapping bound up to node v (looking at $\prec$, this is $\mu_{|v}$) to create a new mapping μ' which uses the node v. There are two ways of mapping μ' differently to the original mapping μ: Either $dom(\mu)$ is extended, or some variables are assigned differently because $\prec$ visits some node first which contains the variable in a different triple resulting in a different assignment. (3) makes sure that the mapping fulfills the top-level filter at exactly the time point where the traversal reaches the corresponding special node s, which is denoted by $\mu_{|s}$.

Theorem 19 ([KK16]). ***EVAL(P_{wwd})** is coNP-complete.*

Proof Idea. We can easily transform the characterisation in Lemma 5 to an algorithm which is in coNP: Compute the maximal tree for μ, $T(P_\mu)$. This is doable in polynomial time. Now it remains to check that this tree is not extendible to linearly many of its children. As we then have to do a homomorphism check against G, this is in coNP. The checks for top-level filters are polynomial again. □

This result can easily be carried over to top level unions of wdd-patterns. The fragment is called $\mathcal{U}_{wwd}$. The fragment where also projection is done over unions of wdd-patterns is calles $\mathcal{S}_{wdd}$.

Corollary 5 ([KK16]). ***EVAL($\mathcal{U}_{wwd}$)** is coNP-complete and **EVAL($\mathcal{S}_{wwd}$)** is Σ_2^P-complete.*

Proof Idea. The coNP-algorithm for $\mathcal{U}_{wwd}$ applies the algorithm for $\mathbf{EVAL}(P_{wwd})$ for every part of the top level UNION. And if the mapping μ is the solution for any of the patterns we return true. Hardness follows by the coNP-completeness of $\mathbf{EVAL}(P_{wwd})$. It is a well known results that $\mathbf{EVAL}(\mathcal{S}_{wd})$ is Σ_2^P-complete [LPPS13], thus Σ_2^P-hardness of $\mathbf{EVAL}(\mathcal{S}_{wdd})$ follows. For the Σ_2^P-membership of $\mathbf{EVAL}(\mathcal{S}_{wdd})$ apply the following algorithm: Guess the values of the existential variables and then call a coNP-oracle for $\mathcal{U}_{wwd}$ on the resulting mapping. This yields a Σ_2^P algorithm. □

6.3 Expressivity of wwd-Patterns and their Extensions

In [KK16] Kaminski et al. established that one has a greater set of tools available if one uses weakly-well designed SPARQL instead of well-designed SPARQL. Even though this fact may seem to be of obvious nature it must be proven mathematically. This characteristic of a SPARQL fragment (or language) is called expressive power.

Definition 49 (Expressive power of languages [KK16])**.** *A language $\mathcal{L}_1$ has the same expressive power as language $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \sim \mathcal{L}_2$, if for every query $Q_1 \in \mathcal{L}_1$ there is a query $Q_2 \in \mathcal{L}_2$ such that $Q_1 \equiv Q_2$ and for every query $Q_2 \in \mathcal{L}_2$ there is a query $Q_1 \in \mathcal{L}_1$ such that $Q_1 \equiv Q_2$ holds. We write $\mathcal{L}_2 < \mathcal{L}_1$ if we can only find a query $Q_1 \in \mathcal{L}_1$ for every query $Q_2 \in \mathcal{L}_2$ but not vice versa.*

The goal for this section is to show the following results:

1. $P_{wd} < P_{wwd} < P$
2. $\mathcal{U}_{wd} < \mathcal{U}_{wwd} < \mathcal{U}$
3. $\mathcal{S}_{wwd} \sim \mathcal{S}$

To prove the first item on the list we need the definition of "weakly monotone". This characteristic was already used by Arenas and Perez in [AP11] to prove that $P_{wd} < P$. They showed that unlike P the fragment P_{wd} is weakly monotone and hence $P_{wd} < P$.

Definition 50 ([KK16])**.** *A query Q is weakly monotone if $[\![Q]\!]_{G_1} \sqsubseteq [\![Q]\!]_{G_2}$ for any two graphs G_1 and G_2 with $G_1 \subseteq G_2$. A fragment $\mathcal{L}$ is called weakly monotone if it contains only weakly monotone queries.*

Now we are ready to establish our desired result.

Theorem 20 ([KK16])**.** *It holds that $P_{wd} < P_{wwd}$.*

Proof. We show the desired result by observing that P_{wwd} is not weakly monotone. Consider the pattern

$$Q = ((x, \mathit{type}, \mathit{person}) \text{ OPT } (x, \mathit{name}, y)) \\ \text{OPT } (x, \mathit{v\_card} : \mathit{name}, y)$$

and the two graphs

$$G_1 = \{(\mathit{P1}, \mathit{type}, \mathit{person})(P1, \mathit{v\_card} : \mathit{name}, \mathit{Anastasia})\} \text{ and}$$
$$G_2 = \{(\mathit{P1}, \mathit{type}, \mathit{person}), (P1, \mathit{v\_card} : \mathit{name}, \mathit{Anastasia}), \\ (\mathit{P1}, \mathit{name}, \mathit{Ana})\}.$$

Clearly $G_1 \subseteq G_2$. Let $\mu_1 = \{x \mapsto P1, y \mapsto Anastasia\}$ and let $\mu_2 = \{x \mapsto P1, y \mapsto Ana\}$. It is also clear considering that Q is a priority pattern that $\mu_1 \in [\![Q]\!]_{G_1}$ and $\mu_2 \in [\![Q]\!]_{G_2}$. But we cannot extend μ_1 to μ_2. Thus $\mu_1 \not\sqsubseteq \mu_2$ and thus $P_{wd} < P_{wwd}$ follows from the fact that P_{wd} is weakly monotone and the fact that $P_{wd} \subseteq P_{wwd}$. □

Definition 51 ([KK16])**.** *A query Q is non-reducing if for any two graphs G_1, G_2 such that $G_1 \subseteq G_2$ and any mapping $\mu_1 \in [\![Q]\!]_{G_1}$, there is no $\mu_2 \in [\![Q]\!]_{G_2}$ such that $\mu_2 \sqsubset \mu_1$ (i.e.$\mu_2 \sqsubset \mu_1$ and $\mu_2 \neq \mu_1$). A fragment $\mathcal{L}$ is non-reducing if it contains only non-reducing queries.*

If a query is non-reducing it can happen that previously bound variable will get unbound. It is easy to see that *wwd*-patterns are non-reducing as we can only replace prior bindings but not erase them.

Theorem 21 ([KK16])**.** *It holds that $P_{wwd} < P$.*

Proof. Let

$$P = (x, a, 1) \text{ OPT } ((y, a, 2) \text{ OPT } (x, a, 3)),$$

$$G_1 = \{(1, a, 1), (2, a, 2)\} \text{ and } G_2 = G_1 \cup \{(3, a, 3)\}.$$

Then $\mu_1 = \{x \mapsto 1, y \mapsto 2\}$ is the only mapping in $[\![P]\!]_{G_1}$ while $\mu_2 = \{x \mapsto 1\}$ is the only mapping in $[\![P]\!]_{G_2}$. Hence $[\![P]\!]_{G_2} \sqsubset [\![P]\!]_{G_1}$ even though $G_1 \subseteq G_2$. As we already mentioned *wwd*-patterns are non-reducing but this pattern is not non-reducing. Thus the desired result follows. □

The next step is to inspect the fragments $\mathcal{U}_{wd}$, $\mathcal{U}_{wwd}$ and $\mathcal{U}$. It is easy to see that $\mathcal{U}_{wd}$ inherits weak monotonicity from P_{wd}. Thus we can see that $\mathcal{U}_{wd} < \mathcal{U}_{wwd}$. Unfortunately we cannot use non-reducibility as it doesn't propagate to unions. Therefore a new condition is defined, namely extension-witnessing.

Definition 52. *A query Q is extension-witnessing if for any two graphs $G_1 \subseteq G_2$ and mapping $\mu \in [\![Q]\!]_{G_2}$ such that $\mu \notin [\![Q]\!]_{G_1}$ there is a triple t in Q such that $vars(t) \subseteq dom(\mu)$ and $\mu(t) \in G_2 \backslash G_1$. A fragment is extension-witnessing if all its queries are extension-witnessing.*

With this new condition, we can show that unions of wwd-patterns are e-witnessing.

Theorem 22 ([KK16])**.** *It holds that $\mathcal{U}_{wd} < \mathcal{U}_{wwd} < \mathcal{U}$.*

Proof. Again consider the counterexample from before:

$$P = (x, a, 1) \text{ OPT } ((y, a, 2) \text{ OPT } (x, a, 3)),$$

$$G_1 = \{(1, a, 1), (2, a, 2)\} \text{ and } G_2 = G_1 \cup \{(3, a, 3)\}.$$

Then $\mu_1 = \{x \mapsto 1, y \mapsto 2\}$ is the only mapping in $[\![P]\!]_{G_1}$ while $\mu_2 = \{x \mapsto 1\}$ is the only mapping in $[\![P]\!]_{G_2}$. This is clearly not e-witnessing. □

Queries over unions of wwd-patterns are as expressive as full SPARQL.

Theorem 23 ([KK16]). *It holds that* $\mathcal{S}_{wwd} \sim \mathcal{S}$.

Proof Idea. The Tr-normal form can be described by the following grammar:

$$S ::= T \text{ FILTER } R \qquad T ::= B \mid T \text{ OPT } B$$

B is a set of triple patterns and R is a filter condition. A pattern P is called Tr-normal if P is a union of patterns S described in the grammar. Every Tr-normal pattern is contained in $\mathcal{U}_{wwd}$. Let Q be a graph pattern of the form
SELECT X WHERE UNION U. Where $U = \{P_1, \ldots, P_n\}$ is a set of UNION-free patterns and UNION U stands for $(\ldots(P_1$ UNION $P_2)$ UNION $\ldots)$ UNION P_n. $Tr(Q)$ describes the graph pattern SELECT X WHERE UNION $\{Tr_X(P) \mid P \in U\}$. A UNION-normal pattern is a graph pattern of the form (P_1 UNION P_2 UNION P_3 UNION $\cdots$ UNION P_n), where each P_i $(1 \leq i \leq n)$ is UNION-free.

Assume a pattern P from general SPARQL which is UNION-free. The authors in [KK16] developed a recursive procedure called $Tr_S(\cdot)$ which transforms a graph pattern into a graph pattern in a special form. The following facts could be shown for $Tr_S(\cdot)$: $P|_S \equiv Tr_S(P)|_S$ for any finite set S which means that $Tr_s(\cdot)$ preserves the meaning of projection. $Tr(P)$ has been shown to have a result of the form UNIONU where every pattern $P' \in U$ is weakly well-designed. Additionally it has been shown that $Tr(P)$ is Tr-normal and hence in $\mathcal{U}_{wwd}$. For any UNION-normal pattern Q, it follows immediately that $Q \equiv Tr(Q)$. And as every query is equivalent to a UNION-normal pattern we are done [PAG09].

□

6.4 Static Analysis of wwd-Patterns

The following results technically imply some of the results in Section 3.2 but the results in the Section 3.2 were also of algorithmical interest because one could derive an "out of the box" procedure to decide the problems. The proof in this section is technically correct but doesn't go into detail about the witnessing mapping that would allow for such procedure.

Theorem 24 ([KK16]). *The problems* ***EQUIVALENCE($\mathcal{L}$)***, ***CONTAINMENT($\mathcal{L}$)*** *and* ***SUBSUMPTION($\mathcal{L}$)*** *are* Π_2^P*-complete for any* $\mathcal{L} \in \{P_{wwd}, U_{wwd}\}$.

Proof Idea. The membership result for **CONTAINMENT**($\mathbf{U}_{wwd}$) follows from the following counterexample property: if $P \not\subseteq P'$ for some $P, P' \in \mathbf{U}_{wwd}$, then there is a witnessing mapping of size $O(|P| + |P'|)$. Consider the following Π_2^P-algorithm: Guess a mapping μ and a graph G of linear size and check that $\mu \notin [\![P']\!]_G$ and then call a coNP oracle for checking if $\mu \in [\![P]\!]_G$. Using this result we can show that **EQUIVALENCE**($\mathcal{U}_{wwd}$) and **SUBSUMPTION**($\mathcal{U}_{wwd}$) are also in Π_2^P. The hardness proofs for **SUBSUMPTION**(P_{wwd}) and
EQUIVALENCE(P_{wwd}) are by reduction from $3 - QSAT_{\forall,2}$.
CONTAINMENT(P_{wwd}) is Π_2^p-hard by the results from [PS14]. □

CHAPTER 7

Conclusion

In this thesis we presented an extension of the work of Perez et al. [PAG09]: In [PAG09] well-designed SPARQL was analyzed and we extended the well-designed fragment with two SPARQL operators, namely GRAPH and SERVICE. We then analyzed the evaluation problem of this fragment. This was done by transforming well-designed patterns with GRAPH and SERVICE to a well-designed pattern tree for which the complexity of the evaluation problem is well known. Our results showed that the complexity of the evaluation problem did not increase even though we added the GRAPH and SERVICE operator to the well-designed SPARQL fragment.

We also investigated the difference between boundedness and strong boundedness, two notions which were introduced by [BAACP13] to evaluate SERVICE queries which gather information from several endpoints at once. In [BAACP13] we learned that deciding if a variable is bounded in a graph pattern is undecidable and that deciding if a variable in a graph pattern is strongly bounded is easily decidable. After comparing the two definitions we found an example of a pattern where a variable was bound but not strongly bound. We also conjectured that deciding if a variable is bound in pattern might be decidable in fragments of SPARQL where the problem **EQUIVALENCE** is feasible.

7.1 Future Work

There is still work to do on the theoretical side: Most importantly we did not do a static analysis of the well-designed fragment of SPARQL which allows the usage of the GRAPH and SERVICE operator. The problems **CONTAINMENT**, **SUBSUMPTION** and **EQUIVALENCE** still need to be analyzed in this fragment. One could use our translation described in Section 4.1 as a foundation for this work, even though one would likely need to modify it for this purpose.

The results for our fragment P_{wdgs} can be extended to weakly well-designed SPARQL. As 99% of all queries containing OPT are covered by the weakly-well designed SPARQL fragment we shall have this fragment also support the GRAPH and SERVICE operators.

When evaluating the SERVICE operator with a variable as destination, semantics and algorithms for evaluation are still needed. There are more problems than just binding the destination variables: The limited bandwidth of the servers hosting RDF data demands specialized semantics and algorithms. Another situation where adapted algorithms are needed is when RDF data is distributed among several servers.

The conjecture that deciding if a variable is bound is decidable in fragments of SPARQL where the problem **EQUIVALENCE** is feasible needs to be proven. This could be achieved by proving the correctness of Algorithm 5.1.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AG08] Renzo Angles and Claudio Gutierrez. *The expressive power of SPARQL*. Springer, 2008.

[all] `http://franz.com/agraph/allegrograph`.

[AMMH07] Daniel J Abadi, Adam Marcus, Samuel R Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*, pages 411–422. VLDB Endowment, 2007.

[AP11] Marcelo Arenas and Jorge Pérez. Querying semantic web data with sparql. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316. ACM, 2011.

[AS] Carlos Buil-Aranda Axel Polleres Lee Feigenbaum Gregory Todd Williams Andy Seaborne, Eric Prud'hommeaux. Rdf dataset, sparql 1.1 federated query. `https://www.w3.org/TR/sparql11-federated-query`.

[AVL+11] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In *The Semantic Web–ISWC 2011*, pages 18–34. Springer, 2011.

[BAACP13] Carlos Buil-Aranda, Marcelo Arenas, Oscar Corcho, and Axel Polleres. Federating queries in sparql 1.1: Syntax, semantics and evaluation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 18(1):1 – 17, 2013. Special Section on the Semantic and Social Web.

[BAP14] Carlos Buil-Aranda and Axel Polleres. Towards equivalences for federated sparql queries. 2014.

[BAPU14] Carlos Buil-Aranda, Axel Polleres, and Jürgen Umbrich. Strategies for executing federated queries in sparql1. 1. In *The Semantic Web–ISWC 2014*, pages 390–405. Springer, 2014.

[BKPR14] Stefan Bischof, Markus Krötzsch, Axel Polleres, and Sebastian Rudolph. Schema-agnostic query rewriting in sparql 1.1. In *The Semantic Web–ISWC 2014*, pages 584–600. Springer, 2014.

[BL] T. Berners-Lee. Linked data - design issues. http://www.w3.org/DesignIssues/LinkedData.html.

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. 2001.

[BPS15] Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation of well-designed pattern trees. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, page 18, 2015.

[CEGL11] Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. Psparql query containment. 2011.

[CGK08] Andrea Calı, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. 2008.

[CTL] T. Heath C.Bizer and T.Berners-Lee. Linked data - the story so far.

[data] http://data.gov.

[datb] http://data.gov.uk.

[GHM04] Claudio Gutierrez, Carlos Hurtado, and Alberto O Mendelzon. Foundations of semantic web databases. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 95–106. ACM, 2004.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GS11] Olaf Görlitz and Steffen Staab. Splendid: Sparql endpoint federation exploiting void descriptions. *COLD*, 782, 2011.

[HAMP14] Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:42–69, 2014.

[HS12] Katja Hose and Ralf Schenkel. Towards benefit-based rdf source selection for sparql queries. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*, page 2. ACM, 2012.

[jen] http://jena.apache.org.

[JK84] D.S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167 – 189, 1984.

[KG14] Egor V Kostylev and Bernardo Cuenca Grau. On the semantics of sparql queries with optional matching under entailment regimes. In *The Semantic Web–ISWC 2014*, pages 374–389. Springer, 2014.

[KK16] Mark Kaminski and Egor V. Kostylev. Beyond Well-designed SPARQL. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory (ICDT 2016)*, volume 48 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[LPPS12] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *In PODS*, pages 89–100, 2012.

[LPPS13] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Transactions on Database Systems (TODS)*, 38(4):25, 2013.

[LS98] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification, 1998.

[MSMMV15] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Federated sparql queries processing with replicated fragments. In *The Semantic Web-ISWC 2015*, pages 36–51. Springer, 2015.

[NW10] Thomas Neumann and Gerhard Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113, 2010.

[PAG06] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International semantic web conference*, volume 4273, pages 30–43. Springer, 2006.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

[Pol07] Axel Polleres. From sparql to rules (and back). In *Proceedings of the 16th international conference on World Wide Web*, pages 787–796. ACM, 2007.

[PS14] Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed sparql. 2014.

[PV11] Francois Picalausa and Stijn Vansummeren. What are real sparql queries like? In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 7:1–7:6, New York, NY, USA, 2011. ACM.

[QL08] Bastian Quilitz and Ulf Leser. *Querying distributed RDF data sources with SPARQL*. Springer, 2008.

[ses] `http://rdf4j.org`.

[SGK+08] Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, and Stefan Manegold. Column-store support for rdf data management: not all swans are white. *Proceedings of the VLDB Endowment*, 1(2):1553–1563, 2008.

[SHa] Eric Prud'hommeaux Steve Harris, Andy Seaborne. Rdf dataset, sparql 1.1 query language w3c recommendation. `http://www.w3.org/TR/sparql11-query/#rdfDataset`.

[SHb] Eric Prud'hommeaux Steve Harris, Andy Seaborne. Service variables, sparql 1.1 query language w3c recommendation. `https://www.w3.org/TR/2012/PR-sparql11-federated-query-20121108/diff#variableService`.

[SHH+11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *The Semantic Web–ISWC 2011*, pages 601–616. Springer, 2011.

[SHK+08] Michael Schmidt, Thomas Hornung, Norbert Küchlin, Georg Lausen, and Christoph Pinkel. *An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario*. Springer, 2008.

[SKCT05] Giorgos Serfiotis, Ioanna Koffina, Vassilis Christophides, and Val Tannen. Containment and minimization of rdf/s query patterns. In *The Semantic Web–ISWC 2005*, pages 607–623. Springer, 2005.

[SML10] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, pages 4–33. ACM, 2010.

[SNP+13] Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, Helena F Deus, and Manfred Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *The Semantic Web–ISWC 2013*, pages 574–590. Springer, 2013.

[SSB+08] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604. ACM, 2008.

[vir] `virtuoso.openlinksw.com`.

[WEGL12] Melisachew Wudage, Jérôme Euzenat, Pierre Genevès, and Nabil Layaıda. Sparql query containment under shi axioms. In *Proceedings 26th AAAI Conference*, pages 10–16, 2012.

[WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

Appendix

Lemma 6. *Let $P \in P_{wdgs}$, DS a dataset so that $ep(ds) = DS$ and G a graph in the dataset DS so that $(g, G) \in DS$. Let $D = \bigcup_{c \in dom(ep)} data(ep(c))$. Let $Q = trans(P, g, ds)$. Then*

$$Q(D) = \begin{cases} [\![P]\!]^{ep(ds)}_{graph(g,ep(ds))} & \text{if } g \in \mathbf{U}, ds \in \mathbf{U} \\ \bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid & \\ \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds))}, \mu \sim [ds \mapsto ds']\} & \text{if } g \in \mathbf{U}, ds \in \mathbf{V} \\ \bigcup_{g' \in names(ep(ds))} \{\mu \cup [g \mapsto g'] \mid & \\ \mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu \sim [g \mapsto g']\} & \text{if } g \in \mathbf{V}, ds \in \mathbf{U} \\ \bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid & \\ \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} & \text{if } g \in \mathbf{V}, ds \in \mathbf{V} \end{cases}$$

holds.

Proof. We proceed to prove the statement using structural induction on P.

1. For the base case we assume that $P = (u, v, w)$ is a triple pattern:
   By construction we have that $Q : vars(ds, g, u, v, w) \leftarrow T(ds, g, u, v, w)$.
   $\subseteq$:
   Let $ds, g \in \mathbf{U}$. Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in [\![P]\!]^{DS}_G$ be arbitrary. Then $dom(\mu) = vars(P)$ and $\mu(P) \in G$ by SPARQL semantics. Because we have $\mu(P) \in G$ and our database D contains by construction $T(ds, g, \mu(x), \mu(y), \mu(z))$ we have that $\mu \in Q(D)$.

   Let $ds \in \mathbf{V}$ and $g \in \mathbf{U}$. Because ds is a variable we have to show that $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \subseteq Q(D)$. Let $ds' \in dom(ep)$

be arbitrary, let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ where $\mu \sim [ds \mapsto ds']$. Because we have $\mu(P) \in graph(g, ep(ds'))$ by SPARQL semantics and our database D contains by construction
$T(ds', g, \mu(x), \mu(y), \mu(z))$ we have that $\mu \cup [ds \to ds'] \in Q(D)$.

Let $ds \in \mathbf{U}$ and $g \in \mathbf{V}$. Let $DS = ep(ds)$. Because g is a variable we have to show that $\bigcup_{g' \in names(ep(ds))} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} \subseteq Q(D)$. Let $g' \in names(DS)$ be arbitrary, let $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ where $\mu \sim [g \mapsto g']$. Because we have $\mu(P) \in graph(g', DS)$ by SPARQL semantics and our database D contains by construction
$T(ds, g', \mu(x), \mu(y), \mu(z))$ we have that $\mu \cup [g \to g'] \in Q(D)$.

Let $ds, g \in \mathbf{V}$. Because ds and g are variables we have to show that $\bigcup_{g' \in names(ds'), ds' \in dom(ep)} \{\mu \cup \{[ds \mapsto ds'][g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'][g \mapsto g']\}\} \subseteq Q(D)$. Let $ds' \in dom(ep), g' \in names(ep(ds'))$ be arbitrary, let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ where $\mu \sim \{[ds \mapsto ds'][g \mapsto g']\}$. Because we have $\mu(P) \in graph(g', ep(ds'))$ by SPARQL semantics and our database D contains by construction
$T(ds', g', \mu(x), \mu(y), \mu(z))$ we have that $\mu \cup \{[g \to g'][ds \to ds']\} \in Q(D)$. This can be done because $\mu \sim \{[ds \to ds'][g \to g']\}$ was assumed.

$\supseteq$:
Let $ds, g \in \mathbf{U}$. Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in Q(D)$ be arbitrary. We thus have a mapping μ from the variables in $vars(P)$ to constants s.t. $T(ds, g, \mu(u), \mu(v), \mu(w)) \in D$. But by construction of D this means that $(\mu(u), \mu(v), \mu(w)) \in G$ and obviously $dom(\mu) = vars(P)$ thus $\mu \in [\![P]\!]^{DS}_{G}$.

Let $ds \in \mathbf{V}$, $g \in \mathbf{U}$. Because ds is a variable we have to show that $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q(D)$. Let $\sigma \in Q(D)$ be arbitrary. We thus have a mapping σ from the variables in $vars(ds, g, u, v, w)$ to constants s.t.
$T(\sigma(ds), g, \sigma(u), \sigma(v), \sigma(w))) \in D$. Because $\sigma \in Q(D)$ and $ds \in \mathbf{V}$, we have $\sigma(ds) = ds'$, s.t. by construction $ds' \in dom(ep)$. Let $\mu = \sigma \backslash [ds \mapsto ds']$. By construction this means that $(\mu(u), \mu(v), \mu(w)) \in graph(g, ep(ds'))$ and obviously $dom(\mu) = vars(P)$. Thus $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ holds. It remains to show that $\mu \sim [ds \mapsto ds']$ but this is obvious because σ contains $[ds \mapsto ds']$ and $\mu = \sigma \backslash [ds \mapsto ds']$.

Let $ds \in \mathbf{U}$, $g \in \mathbf{V}$. Let $DS = ep(ds)$. Because g is a variable we have to show that $\bigcup_{g' \in names(ep(a))} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{DS}_{graph(g,DS)}, \mu \sim [g \mapsto g']\} \supseteq Q(D)$.

Let $\sigma \in Q(D)$ be arbitrary. We thus have a mapping σ from the variables in $vars(ds, g, u, v, w)$ to constants s.t.
$T(ds, \sigma(g), \sigma(u), \sigma(v), \sigma(w))) \in D$. Because $\sigma \in Q(D)$ and $g \in \mathbf{V}$, we have $\sigma(g) = g'$, s.t. by construction $g' \in names(ds)$. Let $\mu = \sigma \backslash [g \mapsto g']$ be a mapping. By construction this means that $(\mu(u), \mu(v), \mu(w) \in graph(g', DS)$ and obviously $dom(\mu) = vars(P)$. Thus $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ holds. It remains to show that $\mu \sim [g \mapsto g']$ but this is obvious because σ contains $[g \mapsto g']$ and $\mu = \sigma \backslash [g \mapsto g']$.

Let $ds \in \mathbf{V}$, $g \in \mathbf{V}$. Because ds, g are variables we have to show that
$\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[g \mapsto g'][ds \mapsto ds']\}$
$\mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g', ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} \supseteq Q(D)$. Let $\sigma \in Q(D)$ be arbitrary. We thus have a mapping σ from the variables in $vars(ds, g, u, v, w)$ to constants s.t. $T(\sigma(ds), \sigma(g), \sigma(u), \sigma(v), \sigma(w)) \in D$. Because $\sigma \in Q(D)$ and $g \in \mathbf{V}, ds \in \mathbf{V}$, we have $\sigma(g) = g', \sigma(ds) = ds'$, s.t. by construction $g' \in names(ds)$ and $ds \in dom(ep)$. Let $\mu = \sigma \backslash \{[g \mapsto g'], [ds \mapsto ds']\}$. By construction this means that $\mu(u), \mu(v), \mu(w) \in graph(g', ep(ds'))$ and obviously $dom(\mu) = vars(P)$. Thus $\mu \in [\![P]\!]^{ep(ds')}_{graph(g', ep(ds'))}$ holds. It remains to show that $\mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}$ but this is obvious because $\sigma \supseteq \{[ds \mapsto d'], [g \mapsto g']\}$ and $\mu = \sigma \backslash \{[ds \mapsto ds'], [g \mapsto g']\}$.

2. Consider the case where P is a graph pattern (P_1 AND P_2).
By construction we have $Q = O_1 \cup O_2 \leftarrow q_1, q_2$.
$\subseteq$:
$ds, g \in \mathbf{U}$: Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in [\![P]\!]^{DS}_G$ be arbitrary. By induction hypothesis we know that $[\![P_1]\!]^{DS}_G = Q_1(D)$ and $[\![P_2]\!]^{DS}_G = Q_2(D)$ hold. By semantics of SPARQL and $\mu \in [\![P]\!]^{DS}_G$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{DS}_G, \mu_2 \in [\![P_2]\!]^{DS}_G$ and $\mu_1 \sim \mu_2\}$. That means that there is a $\mu_1 \in [\![P_1]\!]^{DS}_G$ and a $\mu_2 \in [\![P_2]\!]^{DS}_G$ such that $\mu = \mu_1 \cup \mu_2$. By induction hypothesis $\mu_1 \in Q_1(D)$ and $\mu_2 \in Q_2(D)$ hold. Because $\mu_1 \sim \mu_2$ holds we know that $\mu \in Q(D)$ holds.

$ds \in \mathbf{V}$, $g \in \mathbf{U}$: Let ds' be an arbitrary URI in $dom(ep)$. Let $\mu \in [\![P]\!]^{ep(ds)}_{graph(g, ep(ds))}$ be arbitrary. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_i \cup [ds \mapsto ds'] \mid \mu_i \in [\![P_i]\!]^{ep(ds')}_{graph(g, ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_i(D)$ for $i = 1, 2$. By semantics of SPARQL and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g, ep(ds'))}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g, ep(ds'))}, \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g, ds')}$
and $\mu_1 \sim \mu_2\}$. That means that there is a $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g, ep(ds'))}$ and a $\mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g, ep(ds'))}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu_1' = \mu_1 \cup [ds \mapsto ds']$ and $\mu_2' = \mu_2 \cup [ds \mapsto d']$.By induction hypothesis we have a $\mu_1' \in Q_1(D)$ and a $\mu_2' \in Q_2(D)$. Because $\mu_1 \sim \mu_2$ and $\mu_1 \sim [ds \mapsto ds']$ and $\mu_2 \sim [ds \mapsto ds']$ we have $\mu \cup [ds \mapsto ds'] \in Q(D)$.

$ds \in \mathbf{U}$, $g \in \mathbf{V}$: Let $DS = ep(ds)$. Let g' be an arbitrary URI in $names(ep(a))$. Let $\mu \in [\![P]\!]^{ep(ds)}_{graph(g',ds)}$ be arbitrary. By induction hypothesis we know that $\bigcup_{g' \in names(DS)} \{\mu_i \cup [g \mapsto g'] \mid \mu_i \in [\![P_i]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} = Q_i(D)$ for $i = 1, 2$. By semantics of SPARQL and $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)}$ and $\mu_1 \sim \mu_2\}$. That means that there is a $\mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}$ and a $\mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu_1' = \mu_1 \cup [g \mapsto g']$ and $\mu_2' = \mu_2 \cup [g \mapsto g']$. By induction hypothesis we have a $\mu_1' \in Q_1(D)$ and a $\mu_2' \in Q_2(D)$. Because $\mu_1 \sim \mu_2$ and $\mu_1 \sim [g \mapsto g']$ and $\mu_2 \sim [g \mapsto g']$ we have $\mu \cup [g \mapsto g'] \in Q(D)$.

$ds, g \in \mathbf{V}$: Let ds' be an arbitrary URI in $dom(ep)$. Let g' be an arbitrary URI in $names(ep(d))$. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ be arbitrary. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu_i \cup \{[g \mapsto g'], [ds \mapsto ds']\} \mid \mu_i \in [\![P_i]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[g \mapsto g'], [ds \mapsto ds']\}\} \subseteq Q_i(D)$ for $i = 1, 2$. By semantics of SPARQL and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ and $\mu_1 \sim \mu_2\}$. That means that there is a $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ and a $\mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu_1' = \mu_1 \cup \{[ds \mapsto ds'], [g \mapsto g']\}$ and $\mu_2' = \mu_2 \cup \{[g \mapsto g'], [ds \mapsto ds']\}$. By induction hypothesis we have a $\mu_1' \in Q_1(D)$ and a $\mu_2' \in Q_2(D)$. Because $\mu_1 \sim \mu_2$ and $\mu_1 \sim \{[ds \mapsto ds'], [g \mapsto g']\}$ and $\mu_2 \sim \{[ds \mapsto ds'], [g \mapsto g']\}$ we have $\mu \cup \{[g \mapsto g'], [ds \mapsto ds']\} \in Q(D)$.

$\supseteq$:
$ds, g \in \mathbf{U}$. Let $\mu \in Q(D)$ be arbitrary. Let $DS = ep(ds)$ and $G = graph(g, ep(ds))$. By induction hypothesis we know that $[\![P_1]\!]^{DS}_G = Q_1(D)$ and $[\![P_2]\!]^{DS}_G = Q_2(D)$ hold. Thus there must be some $\mu_1 \in Q_1(D)$, $\mu_2 \in Q_2(D)$ where $\mu = \mu_1 \cup \mu_2$ holds by construction of Q. But thus $\mu_1 \in [\![P_1]\!]^{DS}_G$ and $\mu_2 \in [\![P_2]\!]^{DS}_G$ by induction hypothesis. By semantics of AND we know that $\mu \in [\![P]\!]^{DS}_G$.

$ds \in V, g \in \mathbf{U}$. We want to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q(D)$. Let $\mu \in Q(D)$ be arbitrary. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_i]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu \sim [ds \mapsto ds']\} = Q_i(D)$ for $i \in \{1, 2\}$. Thus there must be some $\mu_1 \in Q_1(D)$, $\mu_2 \in Q_2(D)$ where $\mu = \mu_1 \cup \mu_2$ holds by construction of Q. By induction hypothesis we know that $\mu_i(ds) = ds'$ for some $ds' \in dom(ep)$ for $i \in \{1, 2\}$. Again by i.h. $\mu_i \backslash \{[ds \mapsto ds']\} \in [\![P_i]\!]^{ep(ds)}_{graph(g,ep(ds'))}$ for $i \in \{1, 2\}$.

By semantics of AND we know that $\mu\backslash\{[ds \mapsto ds']\} \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ and by induction hypothesis we know that we have $\mu \sim \{[ds \mapsto ds']\}$.

$ds \in \mathbf{U}, g \in \mathbf{V}$. Let $DS = ep(ds)$. We want to show $\bigcup_{g' \in names(DS)} \{\mu \cup \{[g \mapsto g']\} \mid \mu \in [\![P]\!]^{DS}_{graph(g',DS)}, \mu \sim \{[g \mapsto g']\}\} \supseteq Q(D)$. Let $\mu \in Q(D)$ be arbitrary. By induction hypothesis we know that $\bigcup_{g' \in names(DS)} \{\mu \cup \{[g \mapsto g']\} \mid \mu \in [\![P_i]\!]^{DS}_{graph(g',DS)}, \mu \sim \{[g \mapsto g']\}\} = Q_i(D)$ for $i = 1, 2$. Thus there must be some $\mu_1 \in Q_1(D)$, $\mu_2 \in Q_2(D)$ where $\mu = \mu_1 \cup \mu_2$ construction of Q. By induction hypothesis we know that $\mu_i(g) = g'$ for some $g' \in names(ds)$ for $i \in \{1, 2\}$. Again by i.h. $\mu_i\backslash\{[g \mapsto g']\} \in [\![P_i]\!]^{DS}_{graph(g',DS)}$ $i \in \{1, 2\}$ by induction hypothesis. By semantics of AND we know that $\mu\backslash\{[g \mapsto g']\} \in [\![P]\!]^{DS}_{graph(g',DS)}$ and by induction hypothesis we know that we have $\mu \sim \{[g \mapsto g']\}$.

$ds, g \in \mathbf{V}$. We want to show $\bigcup_{ds \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} \supseteq Q(D)$. Let $\mu \in Q(D)$ be arbitrary. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid \mu \in [\![P_i]\!]^{ep(ds')}_{graph(g',ep(ds))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} = Q_i(D)$ for $i = 1, 2$. Thus there must be some $\mu_1 \in Q_1(D)$, $\mu_2 \in Q_2(D)$ where $\mu = \mu_1 \cup \mu_2$ holds by semantics of $\wedge$ and construction of Q. By induction hypothesis we know that $\mu_i(ds) = ds'$ for some $ds' \in dom(ep)$ and $\mu_i(g) = g'$ for some $g' \in names(ds')$ for $i \in \{1, 2\}$. By induction hypothesis $\mu_i\backslash\{[ds \mapsto ds'], [g \mapsto g']\} \in [\![P_i]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ for $i \in \{1, 2\}$. By semantics of AND we know that $\mu\backslash\{[ds \mapsto ds'], [g \mapsto g']\} \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ and by induction hypothesis we know that we have $\mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}$.

3. Consider the case where P is a graph pattern $(P_1 \text{ OPT } P_2)$.
   By construction we have that $Q_1 = trans(P_1, ds, g) = (T_1, \lambda_1, x_1)$,
   $Q_2 = trans(P_2, ds, g) = (T_2, \lambda_2, x_2)$ and $Q = trans(P, ds, g) = (T, \lambda, x)$ for which $T = T_1 \cup T_2 \cup (r_1, r_2)$ where r_1, r_2 are the roots of T_1, T_2 respectively, $\lambda = \lambda_1 \cup \lambda_2$ and $x = x_1 \cup x_2$.

   $\subseteq$:
   Let $ds, g \in \mathbf{U}$: Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in [\![P]\!]^{DS}_G$ be arbitrary. By semantics of OPT we have that $\mu \in [\![P_1 \; AND \; P_2]\!]^{DS}_G$ or $\mu \in \{\mu_1 \in [\![P_1]\!]^{DS}_G \mid \forall \mu_2 \in [\![P_2]\!]^{DS}_G : \mu_1 \not\sim \mu_2\}$. We thus proceed by case distinction:

   a) Assume $\mu \in [\![P_1 \; AND \; P_2]\!]^{DS}_G$. By induction hypothesis and semantics of AND, we thus have some $\mu_1 \in Q_1(D)$ and some $\mu_2 \in Q_2(D)$ so that $\mu = \mu_1 \cup \mu_2$. Because $\mu_i \in Q_i(D)$ we have by semantics of wdPTs (recall Definition 32)

that $\mu_i \in Q_{i,T_i'}$ so that $T_i' \subseteq T_i$ for $i \in \{1,2\}$. Let $T' = T_1' \cup T_2'$. Because we have $\mu = \mu_1 \cup \mu_2$ we know that $\mu \in Q'_T$. It remains to show that this homomorphism is maximal: assume there was a bigger subtree $\hat{T}$ which would allow a mapping $\mu' \sqsupset \mu$. Then the node which was put additionally to T must either be in T_1 or T_2. But then either μ_1 or μ_2 were not maximal defying the assumption that $\mu_1 \in Q_1$ and $\mu_2 \in Q_2$. Thus $\mu \in Q(D)$.

b) Assume $\mu \in \{\mu_1 \in [\![P_1]\!]_G^{DS} \mid \forall \mu_2 \in [\![P_2]\!]_G^{DS} : \mu_1 \not\sim \mu_2\}$.
Because of our assumption $\mu = \mu_1$ for some $\mu_1 \in [\![P_1]\!]_G^{DS}$. We know that $\mu_1 \in Q_1(D)$ by induction hypothesis. Thus $\mu_1 \in Q_{1,T_1'}$ for some $T_1' \subset T_1$ by semantics of wdPTs. $\mu \in Q_{T_1'}(D)$ follows. It remains to show that there is no $\mu' \sqsupset \mu$. Because of our assumption we know there is no mapping $\mu_2 \in Q_2$ which would be compatible with μ. Thus μ could only be enlarged by a bigger mapping in $Q_1(D)$ but this defies the assumption that $\mu_1 \in Q_1(D)$.

Let $ds \in \mathbf{V}$ and $g \in \mathbf{U}$: We want to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \subseteq Q(D)$. Let ds' be an arbitrary URI in $dom(ep)$. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ds')}$ be arbitrary. By semantics of OPT we have that $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds')}_{graph(g,ds')}$ or $\mu \in \{\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ds')} \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ds')} : \mu_1 \not\sim \mu_2\}$. We thus proceed by case distinction:

a) Assume $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds')}_{graph(g,ds')}$. By induction hypothesis we have $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_i]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu_i \sim [ds \mapsto ds']\} \subseteq Q_i(D)$ for $i \in \{1,2\}$. By semantics of SPARQL and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ds')}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ds')}, \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ds')} \text{ and } \mu_1 \sim \mu_2\}$. Thus there is a $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ds')}$ and a $\mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ds')}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu_i' = \mu_i \cup [ds \mapsto ds']$ for $i \in \{1,2\}$. Because $\mu_i \in Q_i(D)$ we have a $T_i' \subseteq T_i$ such that $\mu_i' \in Q_{i,T_i'}(D)$ for $i \in \{1,2\}$. Let $T' = T_1' \cup T_2'$. Because $\mu_1 \sim \mu_2$ and $\mu_1 \sim [ds \mapsto ds']$ and $\mu_2 \sim [ds \mapsto ds']$ we have $\mu \cup [ds \mapsto ds'] \in Q'_T(D)$. It remains to show that this homomorphism $\mu \cup [ds \mapsto ds']$ is maximal: assume there was a bigger subtree $\hat{T}$ which would allow a mapping $\mu' \sqsupset (\mu \cup [ds \mapsto ds'])$. Then the node which was put additionally to T must either be in T_1 or T_2. But then either μ_1' or μ_2' were not maximal defying the assumption that $\mu_1' \in Q_1(D)$ and $\mu_2' \in Q_2(D)$. Thus $\mu \cup [ds \mapsto ds'] \in Q(D)$.

b) Assume $\mu \in \{\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ds')} \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ds')} : \mu_1 \not\sim \mu_2\}$. By induction hypothesis we have
$\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [ds \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_1(D)$.
Because $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g,ds')}$ we have $\mu \cup [ds \mapsto ds'] \in Q_{T_1'}(D)$, where $T_1' \subseteq T_1$

by wdPT semantics. It remains to show that there is no $\mu' \sqsupset \mu$. But because we have $\{\mu_1 \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ds')}\}$, we know there is no mapping of μ_2 of Q_2 which would be compatible with μ. Thus μ could only be a bigger mapping $\mu' \in Q_1(D)$ but this defies the assumption that $\mu_1 \in Q_1(D)$ and thus we are done.

Let $ds \in \mathbf{U}$ and $g \in \mathbf{V}$: Let $DS = ep(ds)$. We want to show $\bigcup_{g' \in names(DS)} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} \subseteq Q(D)$ Let g' be an arbitrary graph in $names(DS)$. Let $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ be arbitrary. By semantics of OPT we have that $\mu \in [\![P_1 \text{ AND } P_2]\!]^{DS}_{graph(g',DS)}$ or $\mu \in \{\mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)} \mid \forall \mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)} : \mu_1 \not\sim \mu_2\}$. We proceed by case distinction:

a) Assume $\mu \in [\![P_1 \ AND \ P_2]\!]^{DS}_{graph(g',DS)}$. By induction hypothesis we have $\bigcup_{g' \in names(DS)} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P_i]\!]^{DS}_{graph(g',DS)}, \mu_i \sim [g \mapsto g']\} = Q_i(D)$ and $i = 1, 2$. By semantics of SPARQL and $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)} \text{ and } \mu_1 \sim \mu_2\}$. Thus there is a $\mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ds)}$ and a $\mu_2 \in [\![P_2]\!]^{ep(ds)}_{graph(g',ds)}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu'_i = \mu_i \cup [g \mapsto g']$ for $i \in \{1, 2\}$. Because $\mu_i \in Q_i(D)$ we have a $T'_i \subseteq T_i$ such that $\mu'_i \in Q_{i,T'_i}(D)$ for $i \in \{1, 2\}$. Let $T' = T'_1 \cup T'_2$. Because $\mu_1 \sim \mu_2$ and $\mu_1 \sim [g \mapsto g']$ and $\mu_2 \sim [g \mapsto g']$ we have $\mu \cup [g \mapsto g'] \in Q'_T(D)$. It remains to show that this homomorphism $\mu \cup [g \mapsto g']$ is maximal: assume there was a bigger subtree $\hat{T}$ which would allow a mapping $\mu' \sqsupset (\mu \cup [g \mapsto g'])$. Then the node which was put additionally to T must either be in T_1 or T_2. But then either μ'_1 or μ'_2 were not maximal defying the assumption that $\mu'_1 \in Q_1(D)$ and $\mu'_2 \in Q_2(D)$. Thus $\mu \cup [g \mapsto g'] \in Q(D)$.

b) Assume $\mu \in \{\mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)} \mid \forall \mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)} : \mu_1 \not\sim \mu_2\}$. By induction hypothesis we have $\bigcup_{g' \in names(DS)} \{\mu_1 \cup [g \mapsto g'] \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} = Q_1(D)$. Because $\mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ds)}$ we have $\mu \cup [g \mapsto g'] \in Q_{T'_1}(D)$, where $T'_1 \subseteq T_1$ by wdPT semantics. It remains to show that there is no $\mu' \sqsupset \mu$. But because we have $\{\mu_1 \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds)}_{graph(g',ds)}\}$, we know there is no mapping of μ_2 of Q_2 which would be compatible with μ. Thus μ could only be a bigger mapping $\mu' \in Q_1(D)$ but this defies the assumption that $\mu_1 \in Q_1(D)$ and thus we are done.

Let $ds, g \in \mathbf{V}$: We want to show $\bigcup_{ds' \in dom(ep), g' \in names(ep(d))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} \supseteq Q(D)$. Let ds' be

an arbitrary URI in dom(ep), and g' be an arbitrary URI in $names(ep(d))$. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ be arbitrary. By semantics of OPT we have that $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ or
$\mu \in \{\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))} \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))} : \mu_1 \not\sim \mu_2\}$. We thus proceed by case distinction:

a) Assume $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))}$. By induction hypothesis we thus have $\bigcup_{ds' \in dom(ep), g' \in names(ep(d))} \{\mu \cup \{[ds \mapsto ds'][g \mapsto g']\} \mid \mu \in [\![P_i]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu_i \sim \{[g \mapsto g'], [ds \mapsto ds']\}\} = Q_i(D)$ and $i = 1, 2$. By semantics of SPARQL and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ we know that $\mu \in \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))} \text{ and } \mu_1 \sim \mu_2\}$. Thus there is a $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ and a $\mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ such that $\mu = \mu_1 \cup \mu_2$. Let $\mu'_i = \mu_i \cup \{[ds \mapsto ds'][g \mapsto g']\}$ for $i \in \{1, 2\}$. Because $\mu_i \in Q_i(D)$ we have a $T'_i \subseteq T_i$ such that $\mu'_i \in Q_{i,T'_i}(D)$ for $i \in \{1, 2\}$. Let $T' = T'_1 \cup T'_2$. Because $\mu_1 \sim \mu_2$ and $\mu_i \sim \{[g \mapsto g'], [ds \mapsto ds']\}$ for $i \in \{1, 2\}$, we have $\mu \cup [g \mapsto g'] \in Q'_T(D)$. It remains to show that the mapping $\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\}$ is maximal: assume there was a bigger subtree $\hat{T}$ which would allow a mapping $\mu' \sqsupset (\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\})$. Then the node which was put additionally to T must either be in T_1 or T_2. But then either μ'_1 or μ'_2 were not maximal defying the assumption that $\mu'_1 \in Q_1(D)$ and $\mu'_2 \in Q_2(D)$. Thus $\mu \cup \{[g \mapsto g'], [ds \mapsto ds']\} \in Q(D)$.

b) Assume $\mu \in \{\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))} \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ep(ds'))} : \mu_1 \not\sim \mu_2\}$. By induction hypothesis we have $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu_1 \cup \{[ds' \mapsto ds][g \mapsto g']\} \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\} = Q_1(D)$. Because $\mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ds)}$ we have $\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \in Q_{T'_1}(D)$, where $T'_1 \subseteq T_1$ by wdPT semantics. It remains to show that there is no $\mu' \sqsupset \mu$. But because we have $\{\mu_1 \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g',ds')}\}$, we know there is no mapping of μ_2 of Q_2 which would be compatible with μ. Thus μ could only be a bigger mapping $\mu' \in Q_1(D)$ but this defies the assumption that $\mu_1 \in Q_1(D)$ and thus we are done.

$\supseteq$:
Because of the construction of Q, a solution, call it μ must either adhere $\mu \in Q_{T'}(D)$ for $T' \subseteq T$. We will further distinguish two cases:

a) $\mu \in Q_{T'}(D)$ for some $T' \subseteq T_1$

b) $\mu \in Q_{T'}(D)$ for some $T' = T'_1 \cup T'_2$ where $T'_1 \subseteq T_1$ and $T'_2 \subseteq T_2$.

Let $ds, g \in \mathbf{U}$. Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in Q(D)$ be arbitrary. Case distinction:

a) $\mu \in Q_{T_1'}(D)$: Thus $\mu \in Q_1(D)$ and by i.h. $\mu \in [\![P_1]\!]_G^{DS}$ and then also $\{\mu \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]_G^{DS}\}$ by assumption. Thus $\mu \in [\![P]\!]_G^{DS}$.

b) If $\mu \in Q_{T'}(D)$ we then have $\mu_{|vars(Q_1)} \in Q_1(D)$ (restricted to the variables in Q_1) and $\mu_{|vars(Q_2)} \in Q_2(D)$(restricted to the variables in Q_2). Thus by i.h. and semantics of AND we have $\mu \in [\![P_1\ AND\ P_2]\!]_G^{DS}$ and $\mu \in [\![P]\!]_G^{DS}$.

Let $ds \in \mathbf{V}, g \in \mathbf{U}$. We want to prove $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q(D)$. Let $\sigma \in Q(D)$ be arbitrary. Because $\sigma \in Q(D)$ and $ds \in V$ we have that $\sigma(ds) = ds'$ for some $ds' \in dom(ep)$. Case distinction:

a) $\sigma \in Q_{T_1'}(D)$: Because the assumption implies $\sigma \in Q_1(D)$ we can then use the induction hypothesis $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_1(D)$. Let $\mu = \sigma \backslash [ds \mapsto ds']$. Thus $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ and we also have $\{\mu \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g,ep(ds'))}\}$ because of our assumption. Thus $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds')}$ and $\mu \sim [ds \mapsto ds']$ by induction hypothesis.

b) If $\sigma \in Q_{T'}(D)$ we can use the induction hypothesis twice:
$\bigcup_{ds' \in dom(ep)} \{\mu_i \cup [ds \mapsto ds'] \mid \mu_i \in [\![P_i]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu_i \sim [ds \mapsto ds']\} = Q_i(D)$
for $i = 1, 2$.
Let $\mu = \sigma \backslash [ds \mapsto ds']$. Because $\sigma \in Q_{T'}(D)$ we have $\mu = \mu_1 \cup \mu_2$ for some $\mu_{1|vars(Q_1)} \cup [ds \mapsto ds'] \in Q_1(D)$ (restricted to the variables in Q_1) and $\mu_{2|vars(Q_2)} \cup [ds \mapsto ds'] \in Q_2(D)$ (restricted to the variables in Q_2). Thus $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds')}_{graph(ds',g)}$ and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ds')}$. Also $\mu \sim [ds \mapsto ds']$ holds by induction hypothesis.

Let $ds \in \mathbf{U}, g \in \mathbf{V}$. Let $DS = ep(ds)$. We want to prove $\bigcup_{g' \in names(DS)} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{DS}_{graph(g,DS)}, \mu \sim [b \mapsto g]\} \supseteq Q(D)$. Let $\sigma \in Q(D)$ be arbitrary. Because $\sigma \in Q_D$ and $g \in V$ we have that $\sigma(g) = g'$ for some $g' \in names(DS)$. Case distinction:

a) $\sigma \in Q_{T_1}(D)$: Because the assumption implies $\sigma \in Q_1(D)$ we can then use the induction hypothesis $\bigcup_{g \in names(DS)} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} = Q_1(D)$. Let $\mu = \sigma \backslash [g \mapsto g']$. Thus $\mu \in [\![P_1]\!]^{DS}_{graph(g',DS)}$ and we also have $\{\mu \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{DS}_{graph(g',DS)}\}$ because of our assumption. Thus $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ and $\mu \sim [g \mapsto g']$ by induction hypothesis.

b) If $\sigma \in Q_{T'}(D)$ we can use the induction hypothesis twice:
$\bigcup_{g \in names(DS)} \{\mu_i \cup [g \mapsto g'] \mid \mu_i \in [\![P_i]\!]^{ep(ds)}_{graph(g', ep(ds))}, \mu_i \sim [g \mapsto g']\} = Q_i(D)$ for $i = 1, 2$.
Let $\mu = \sigma \backslash [g \mapsto g']$. Because $\sigma \in Q_{T'}(D)$ we have $\mu = \mu_1 \cup \mu_2$ for some $\mu_{1|vars(Q_1)} \cup [g \mapsto g'] \in Q_1(D)$ (restricted to the variables in Q_1) and $\mu_{2|vars(Q_2)} \cup [g \mapsto g'] \in Q_2(D)$ (restricted to the variables in Q_2). Thus $\mu \in [\![P_1 \text{ AND } P_2]\!]^{ep(ds)}_{graph(ds,g')}$ and $\mu \in [\![P]\!]^{ep(ds)}_{graph(g',ds)}$. Also $\mu \sim [g \mapsto g']$ holds by induction hypothesis.

Let $ds, g \in \mathbf{V}$ We want to prove $\bigcup_{g' \in names(ep(ds')), ds' \in dom(ep)} \{\mu \cup \{[ds \mapsto ds'][g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g', ep(ds'))}, \mu \sim \{[ds \mapsto ds'][g \mapsto g']\}\} \supseteq Q(D)$. Let $\sigma \in Q(D)$ be arbitrary. Because $\sigma \in Q_D$ and $ds, g \in V$ we have that $\sigma(g) = g'$ and $\sigma(ds) = ds'$ for some $g' \in names(DS)$ and $ds' \in dom(ep)$.

Case distinction:

a) $\sigma \in Q_{T_1}(D)$: we can then use the induction hypothesis
$\bigcup_{ds' \in dom(ep), g' \in names(ep(d))} \{\mu \cup \{[ds \mapsto ds'][g \mapsto g']\} \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(g', ep(ds'))}, \mu \sim \{[ds \mapsto ds'][g \mapsto g']\}\} = Q_1(D)$.
Thus $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g', ep(ds'))}$ and we also have
$\{\mu \not\sim \mu_2 \mid \forall \mu_2 \in [\![P_2]\!]^{ep(ds')}_{graph(g', ep(ds'))}\}$ because of our assumption. Thus $\mu \in [\![P]\!]^{ep(ds')}_{graph(g', ep(ds'))}$ and $\mu \sim \{[ds' \mapsto ds], [g' \mapsto g']\}$ by induction hypothesis.

b) If $\sigma \in Q_T(D)$ we can use the induction hypothesis twice:
$\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu_i \cup \{[ds \mapsto ds'][g \mapsto g'\} \mid$
$\mu_i \in [\![P_i]\!]^{ep(ds')}_{graph(g', ep(ds'))}, \mu_i \sim \{[ds \mapsto ds'][g \mapsto g']\}\} = Q_i(D)$ for $i = 1, 2$.
Let $\mu = \sigma \backslash \{[ds \mapsto ds'][g \mapsto g']\}$. Because $\sigma \in Q_{T'}(D)$ we have $\mu = \mu_1 \cup \mu_2$ for some $\mu_{1|vars(Q_1)} \cup \{[ds \mapsto ds'][g \mapsto g']\}$(restricted to the variables in Q_1) and $\mu_{1|vars(Q_1)}\{[ds \mapsto ds'][g \mapsto g']\}$(restricted to the variables in Q_2). Thus $\mu \in [\![P_1\ AND\ P_2]\!]^{ep(ds')}_{graph(g', ep(ds'))}$ and $\mu \in [\![P]\!]^{ep(ds')}_{graph(g, ep(ds'))}$. Also $\mu \sim \{[ds \mapsto ds'][g \mapsto g']\}$ holds by induction hypothesis.

4. Consider the case where P is a graph pattern (GRAPH u P_1).
Our outputquery is constructed as follows: Let $Q_1 = trans(P_1, u, g)$. Assuming r_1 is the root of T_1 and $\lambda(r_1) = q_1$ we define

$$\lambda'(x) = \begin{cases} q_1, LOC(u, ds), LOC(g, ds) & \text{if } x = r_1 \\ \lambda(x) & \text{otherwise} \end{cases}$$

and $trans(P, ds, g) = (T_1, \lambda', x_1)$.

$\subseteq$:
$ds, g \in \mathbf{U}$:
Let $DS = ep(ds)$ and $G = graph(g, DS)$. Let $\mu \in [\![P]\!]_G^{DS}$ be arbitrary. Proceed by case distinction:

a) $u \in names(DS)$: We have $\mu \in [\![P_1]\!]^{DS}_{graph(u,DS)}$ by SPARQL semantics and assumption. By i.h. we have $Q_1(D) = [\![P_1]\!]^{DS}_{graph(u,DS)}$ and thus $\mu \in Q_1(D)$. By construction of our query Q we see that $\mu \in Q(D)$.

b) $u \in \mathbf{U} \backslash names(DS)$: Then $[\![P_1]\!]^{DS}_{graph(u,DS)} = \{\}$ by SPARQL semantics. But then $Q(D) = \{\}$ because we added $LOC(u, ds)$ to the root of our query Q.

c) $u \in V$: Then $\mu \in S_1$ where
$S_1 = \left\{ \mu_1 \cup [u \to s] \mid s \in names(DS), \mu_1 \in [\![P_1]\!]^{DS}_{graph(s,DS)}, [u \to s] \sim \mu_1 \right\}$.
By induction hypothesis we know that for $Q_1 = trans(P_1, ds, u)$ we have $\bigcup_{g' \in names(DS)} \{\mu_1 \cup [u \mapsto g'] \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu_1 \sim [u \mapsto g']\} = Q_1(D)$.
Because $\mu \in S_1$ there is an $s \in names(DS)$ such that there is a $\mu_1 \in [\![P_1]\!]^{DS}_{graph(s,DS)}$, $\mu = \mu_1 \cup [u \to s]$ and $\mu_1 \sim [u \mapsto s]$. By induction hypothesis we get that $\mu \in Q_1(D)$. Looking at the construction of Q we get $\mu \in Q(D)$.

$ds \in \mathbf{V}, g \in \mathbf{U}$:
We need to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \subseteq Q(D)$. Let $ds' \in dom(ep)$ be arbitrary. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ds')}$ where $\mu \sim [ds \mapsto ds']$ holds be arbitrary. Proceed by case distinction:

a) $u \in names(ep(ds'))$: We have $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))}$ by SPARQL semantics. The induction hypothesis $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_1(D)$ yields $\mu \cup [ds \mapsto ds'] \in Q_1(D)$. By construction of our query Q we see that $\mu \cup [ds \mapsto ds'] \in Q(D)$.

b) $u \in \mathbf{U} \backslash names(ep(d))$: Then $[\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))} = \{\}$. But then $Q(D) = \{\}$ because we added $LOC(u, ds)$ to the root of our query Q.

c) $u \in V$: then $\mu \in S_1$ where $S_1 = \Big\{ \mu_1 \cup [u \to s] \mid s \in names(ep(ds')),$
$\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(s,ep(ds'))} \wedge [u \to s] \sim \mu_1 \Big\}$. By induction hypothesis we know that we receive a wdPT $Q_1 = trans(P_1, ds, u)$ for which
$\bigcup_{g' \in names(ep(ds')), ds' \in dom(ep)} \{\mu_1 \cup \{[u \mapsto g'], [ds \mapsto ds']\} \mid$
$\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu_1 \sim \{[u \mapsto g'], [ds \mapsto ds']\} = Q_1(D)$ holds. Because $\mu \in S_1$ there is an $s \in names(ep(ds'))$ such that there is a $\mu_1 \in$

$[\![P_1]\!]^{ep(ds')}_{graph(s,ep(ds'))}$, $\mu = \mu_1 \cup \{[u \to s]\}$. We get that $\mu \in Q_1(D)$ by induction hypothesis. Also we have that $\mu \cup [ds \to ds'] \in Q(D)$ because we conjunctively added $LOC(g, ds)$ to the root of Q and we assumed $\mu \sim [ds \sim ds']$.

$ds \in \mathbf{U}, g \in \mathbf{V}$:
Let $DS = ep(ds)$. We need to show $\bigcup_{g' \in names(DS)} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{DS}_{graph(g',DS)}, \mu \sim [g \mapsto g']\} \subseteq Q(D)$. Let $g' \in names(ep(ds))$ be arbitrary. Let $\mu \in [\![P]\!]^{DS}_{graph(g',DS)}$ where $\mu \sim [g \mapsto g']$ holds be arbitrary. Proceed by case distinction:

a) $u \in names(DS)$: By SPARQL semantics we have that $\mu \in [\![P_1]\!]^{DS}_{graph(u,DS)}$. We know that we receive a wdPT $Q_1 = trans(P_1, ds, u)$ for which by i.h. $[\![P_1]\!]^{DS}_{graph(u,DS)} = Q_1(D)$ holds. Thus $\mu \in Q_1(D)$. Because of the construction of our query and especially the conjunct $LOC(g, ds)$ in the root of Q we have that $\mu \cup [g \mapsto g'] \in Q_1(D)$.

b) $u \in \mathbf{U} \backslash names(ep(ds))$: then $[\![P_1]\!]^{DS}_{graph(u,DS)} = \{\}$. But then $Q(D) = \{\}$ because we added $LOC(u, ds)$ to the root of our query Q.

c) $u \in V$: then $\mu \in S_1$ where $S_1 = \Big\{\mu_1 \cup [u \to s] \mid s \in names(DS), \mu_1 \in [\![P_1]\!]^{DS}_{graph(s,DS)} \wedge [u \to s] \sim \mu_1\Big\}$. By induction hypothesis we know that we receive a wdPT $Q_1 = trans(P_1, ds, u)$ for which $\bigcup_{g' \in names(DS)} \{\mu_1 \cup \{[u \mapsto g']\} \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu_1 \sim \{[u \mapsto g']\}\} = Q_1(D)$ holds. Because $\mu \in S_1$ there is an $s \in names(DS)$ such that there is a $\mu_1 \in [\![P_1]\!]^{DS}_{graph(s,DS)}$, $\mu = \mu_1 \cup \{[u \to s]\}$. We get that $\mu \in Q_1(D)$ by induction hypothesis. We thus get that $\mu \cup [g \mapsto g'] \in Q(D)$ because we conjunctively added $LOC(g, ds)$ to root of Q and we asssumed $\mu \sim [g \mapsto g']$.

$ds, g \in \mathbf{V}$:
We need to show $\bigcup_{g' \in names(ep(ds')), ds' \in dom(ep)} \{\mu \cup \{[g \mapsto g'], [ds \mapsto ds']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'][g \mapsto g']\}\} \subseteq Q(D)$. Let $ds' \in dom(ep)$ and $g' \in names(ep(ds'))$ be arbitrary. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ where $\mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}$ holds be arbitrary. Proceed by case distinction:

a) $u \in names(ep(ds))$:
By SPARQL semantics we have $\mu \in [\![P_1]\!]^{DS}_{graph}(u, DS)$. We know that we receive a wdPT $Q_1 = trans(P_1, ds', u)$ for which by i.h. $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_1(D)$ holds. Thus $\mu \cup [ds \mapsto$

$ds'] \in Q_1(D)$. By construction of our query and especially the conjunct in the root of Q, i.e., $LOC(g, ds)$ we have that $\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \in Q(D)$.

b) $u \in \mathbf{U} \backslash names(ep(ds'))$ then $[\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))} = \{\}$. But then $Q(D) = \{\}$ because we added $LOC(u, ds)$ to the root of our query.

c) $u \in V$: then $\mu \in S_1$ where $S_1 = \Big\{\mu_1 \cup [u \to s] \mid s \in names(ep(ds')),$ $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(s,ep(ds'))} \wedge [u \to s] \sim \mu_1\Big\}$. By induction hypothesis we know that we receive a wdPT $Q_1 = trans(P_1, ds', u)$ for which $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [u \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} = Q_1(D)$ holds. Because $\mu \in S_1$ there is an $s \in names(ep(ds'))$ such that there is a $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(s,ep(ds'))}$, $\mu = \mu_1 \cup \{[u \to s]\}$ and $\mu_1 \sim \{[u \mapsto s]\}$ by induction hypothesis. We thus get that $\mu \cup \{[g \mapsto g'], [ds \mapsto ds']\} \in Q(D)$ because we conjunctively added $LOC(g, ds)$ to Q and we assumed $\mu \sim \{[g \mapsto g'], [ds \mapsto ds']\}$.

$\supseteq$:
$ds, g \in \mathbf{U}$
Let $\mu \in Q(D)$ be arbitrary.

a) u is a constant: Because $\mu \in Q(D)$ a part of μ must also satisfy $Q_1(D)$ by construction of Q. We have by induction hypothesis that $Q_1(D) = [\![P_1]\!]^{ep(ds)}_u$. And thus by SPARQL semantics $\mu \in [\![P]\!]^{ep(ds)}_g$ holds.

b) u is a variable: By induction hypothesis $\bigcup_{g' \in names(ep(ds))} \{\mu_1 \cup [u \mapsto g'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu_1 \sim [u \mapsto g']\} = Q_1(D)$ holds. By the fact that $\mu \in Q(D)$ and both g and ds are URIs we know that $\mu \in Q_1(D)$ by construction of Q. By induction hypothesis we have $\mu = \mu_1 \cup [u \mapsto g']$ for some $g' \in names(ep(ds))$. We know that $\mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ep(ds))}$ and $\mu_1 \sim [u \mapsto g']$ by induction hypothesis. But this means by semantics of the GRAPH operator that $\mu \in [\![P]\!]^{ep(ds)}_{graph(g,ep(ds))}$.

$ds \in \mathbf{V}, g \in \mathbf{U}$
We want to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q(D)$.
Let $\sigma \in Q(D)$ be arbitrary:

a) u is a constant: We have by induction hypothesis that $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q_1(D)$. A part of σ, call it μ

must fulfill Q_1 because of the construction of Q. Thus $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(u,ep(ds'))}$ and by SPARQL semantics $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ hold. Looking at the construction of Q we see that $\sigma = \mu \cup [ds \mapsto ds']$ for some $ds' \in dom(ep)$ thus $\mu \sim [ds \mapsto ds']$.

b) u is a variable: By induction hypothesis $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu_1 \cup \{[ds \mapsto ds'], [u \mapsto g']\} \mid \mu_1 \in [\![P_1]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu_1 \sim \{[ds \mapsto ds'], [u \mapsto g']\}\} = Q_1(D)$ holds. Some submapping of σ must fulfill $Q_1(D)$ by construction of Q and $\sigma \in Q(D)$. Call it μ. By the fact that $\mu \in Q_1(D)$ we thus know by induction hypothesis that $\mu = \mu_1 \cup \{[u \mapsto g'], [ds \mapsto ds']\}$ for some $g' \in names(ep(ds'))$ and some $ds' \in dom(ep)$. Also, we know that $\mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ and $\mu_1 \sim [u \mapsto g']$. But this means by semantics of the GRAPH operator that $(\mu_1 \cup [u \mapsto ds]) \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$. Because $\mu \in Q_1(D)$ and $\mu = \mu_1 \cup \{[ds \mapsto ds'], [u \mapsto g']\}$ implies $(\mu_1 \cup \{u \mapsto g'\}) \sim [ds \mapsto ds']$ we are done.

$ds \in \mathbf{U}, g \in \mathbf{V}$: Let $DS = ep(ds)$. Let $\sigma(g) = f$. By construction of Q and D we get that $f \in names(ep(ds))$.

We want to show $\bigcup_{f \in names(DS)} \{\mu \cup [g \mapsto f] \mid \mu \in [\![P]\!]^{DS}_{graph(f,DS)}, \mu \sim [g \mapsto f]\} \supseteq Q(D)$.

Let $\sigma \in Q(D)$ be arbitrary.

a) By induction hypothesis we know that $Q_1(D) = [\![P_1]\!]^{DS}_u$. Looking at our mapping σ we know that there is a part of σ, call it μ for which $\mu \in Q_1(D)$ and thus $\mu \in [\![P_1]\!]^{DS}_u$ and by SPARQL semantics $\mu \in [\![P_1]\!]^{DS}_f$ hold. We have $\mu \sim [g \mapsto f]$ for some $f \in names(DS)$ because of the conjunct $LOC(g, ds)$ in the root of Q and $\sigma = \mu \cup [g \mapsto f]$.

b) If u is a variable then consider $Q_1 = trans(P_1, ds, u)$. By induction hypothesis $\bigcup_{g' \in names(DS)} \{\mu_1 \cup [u \mapsto g'] \mid \mu_1 \in [\![P_1]\!]^{DS}_{graph(g',DS)}, \mu_1 \sim [u \mapsto g']\} = Q_1(D)$ holds. Some part of σ must satisfy Q_1 by the construction of Q and $\sigma \in Q(D)$, call it μ. We first show that $\mu \in [\![P]\!]^{DS}_{graph(f,DS)}$: This means, we need to show that $\mu \in [\![P_1]\!]^{DS}_{graph(s,DS}$ for $s \in names(DS)$ and $[u \mapsto s] \sim \mu$. By our i.h. and $\mu \in Q_1(D)$ we know that $\mu = \mu_1 \cup \{[u \mapsto s]\}$, for some $s \in names(DS)$. We know again by i.h. that $\mu_1 \in [\![P_1]\!]^{DS}_{graph(s,DS)}$ and $\mu_1 \sim [u \mapsto s']$. But this means by semantics of the GRAPH operator that $\mu_1 \cup [u \mapsto s] \in [\![P]\!]^{DS}_{graph(f,DS)}$ It remains to show that $\mu \sim [g \mapsto f]$. Looking at the construction of the query we know that $LOC(g, ds)$ must be fulfilled by σ. Thus $\mu \sim [g \mapsto f]$ because μ is a part of σ.

$ds, g \in \mathbf{V}$:

We want to show $\bigcup_{ds' \in dom(ep), f \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto f]\} \mid \mu \in$

$[\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto f]\}\} \supseteq Q(D)$ Let $\sigma \in Q(D)$ be arbitrary. Let $\sigma(ds) = ds'$ and $\sigma(g) = f$. By the construction of Q and D, $ds' \in dom(ep)$ and $f \in names(ep(ds'))$.

a) u is a constant:
By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} = Q_1(D)$. Looking at our mapping σ we know that there is a part of σ, call it μ for which $\mu \in Q_1(D)$ and thus $\mu \backslash [ds \mapsto ds'] \in [\![P_1]\!]^{ep(ds')}_u$. By semantics of graph we have $\mu \backslash [ds \mapsto ds'] \in [\![P_1]\!]^{ep(ds')}_f$. We have $\mu \backslash [ds \mapsto ds'] \sim [g \mapsto f]$ and $\mu \sim [ds \mapsto ds']$ because $\mu \backslash [ds \mapsto ds'] \subseteq \sigma$, $\sigma \in Q(D)$ and because of the conjunct $LOC(g, ds)$ in the root of Q.

b) u is a variable: By induction hypothesis $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [u \mapsto g']\} \mid \mu \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [u \mapsto g']\}\} = Q_1(D)$ holds. Some part of σ must satisfy Q_1 by the construction of Q and $\sigma \in Q(D)$, call it μ. Let $\mu(u) = g'$ for some $g' \in names(ds')$. Because $\mu \in Q_1(D)$ we can use the induction hypothesis: This means $\mu \backslash \{[ds \mapsto ds'], [u \mapsto g']\} \in [\![P_1]\!]^{ep(ds')}_{graph(g',ep(ds'))}$. Obviously $\mu \backslash \{[ds \mapsto ds']\} \sim [u \mapsto g']\}$. From the semantics of the GRAPH operator we get $\mu \backslash \{[ds \mapsto ds']\} \in [\![P_1]\!]^{ep(ds')}_{graph(f,ep(ds'))}$. It remains to show that $\mu \sim [g \mapsto f]$ and $\mu \sim [ds \mapsto ds']$. Looking at the construction of the query we know that $LOC(g, ds)$ must be fulfilled by σ. Thus $\mu \sim [g \mapsto f]$ because μ is a part of σ the same arguments hold for $\mu \sim [ds \mapsto ds']$.

5. Consider the case where P is a graph pattern of the form (SERVICE u P_1). $trans$ checks if $u \in \mathbf{U}$ and $u \notin dom(ep)$. If this is the case $Q : \{\} \rightarrow$. Otherwise we let $Q = trans(P_1, u, g)$ and add $LOC(u, ds)$ and $LOC(g, ds)$ to the root of Q.
$\subseteq$:
Let $ds, g \in U$. Let $DS = ep(ds)$ and $G = ep(g, DS)$. Let $\mu \in [\![P]\!]^{DS}_G$ be arbitrary.

a) $u \in dom(ep)$: that means that $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$ but by i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By construction of Q we have $\mu \in Q(D)$.

b) $u \in \mathbf{U} \backslash dom(ep)$: Thus $\mu = \mu_\emptyset$ which is the empty mapping. This is the same mapping our query $\{\} \leftarrow$ returns and we are done.

c) $u \in \mathbf{V}$: Thus $\mu \in \{\mu_1 \cup [u \rightarrow s] \mid s \in dom(ep), \mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))} \wedge [u \rightarrow s] \sim \mu_1\}$ by semantics. Assume $\mu(u) = s$. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. By semantics of SPARQL $\mu = \mu_1 \cup [u \mapsto s]$ for $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By induction hypothesis we can conclude $\mu \in Q_1(D)$. By construction of Q we have have $\mu = (\mu_1 \cup [u \mapsto s]) \in Q(D)$.

Let $ds \in \mathbf{V}, g \in \mathbf{U}$. We want to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \subseteq Q(D)$. Let $ds' \in dom(ep)$ be arbitrary. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ds')}$ so that $\mu \sim [ds \mapsto ds']$.

a) $u \in dom(ep)$: $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$ by semantics. By i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By construction of q and especially the conjunct $LOC(ds, g)$ in the root, we get $\mu \cup [ds \mapsto ds'] \in Q$.

b) $u \in I \backslash dom(ep)$: By semantics $\mu = \mu_\emptyset$ which is the empty mapping. This is the same mapping our query $\{\} \leftarrow$ returns and we are done.

c) $u \in \mathbf{V}$:
By semantics $\mu \in \{\mu_1 \cup [u \to s] \mid s \in dom(ep), \mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))} \wedge [u \to s] \sim \mu_1\}$. Assume $\mu(u) = s$. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$.
By semantics of SPARQL $\mu = \mu_1 \cup [u \mapsto s]$ for $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By induction hypothesis we can conclude $\mu \in Q_1(D)$. By construction of Q and especially the conjunct $LOC(ds, g)$ in the root, we have $\mu \cup [ds \mapsto ds'] = (\mu_1 \cup [u \mapsto s] \cup [ds \mapsto ds']) \in Q(D)$.

Let $ds \in \mathbf{U}, g \in \mathbf{V}$. We want to show $\bigcup_{g' \in names(ep(ds))} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu \sim [g \mapsto g']\} \subseteq Q(D)$. Let $g \in names(ep(ds))$ be arbitrary. Let $\mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}$ so that $\mu \sim [g \mapsto g']$.

a) $u \in dom(ep)$: By semantics $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$ and by i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By construction of Q and especially the conjunct $LOC(ds, g)$ in the root of Q we have that $\mu \cup [g \mapsto g'] \in Q$.

b) $u \in \mathbf{U} \backslash dom(ep)$: By semantics $\mu = \mu_\emptyset$ which is the empty mapping. This is the same mapping our query $\{\} \leftarrow$ returns and we are done.

c) $u \in \mathbf{V}$: By semantics $\mu \in \{\mu_1 \cup [u \to s] \mid s \in dom(ep), \mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))} \wedge [u \to s] \sim \mu_1\}$. Assume $\mu(u) = s$. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$.
By semantics of SPARQL $\mu = \mu_1 \cup [u \mapsto s]$ for $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By induction hypothesis we can conclude $\mu \in Q_1(D)$. By construction of Q and especially the conjunct $LOC(ds, g)$ we have $(\mu \cup [g \mapsto g']) = (\mu_1 \cup [u \mapsto s] \cup [g \mapsto g']) \in Q(D)$.

Let $ds, g \in \mathbf{V}$. We want to show $\bigcup_{ds' \in dom(ep), g' \in names(ep(ds'))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g', ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} = Q(D)$. Let $ds' \in dom(ep)$ and $g' \in nameS(ep(ds'))$ be arbitrary. Let $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,g')}$ so that $\mu \sim [ds \mapsto ds']$.

a) $u \in dom(ep)$: By semantics $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def, ep(u))}$ but by i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By construction of Q and especially the conjunct $LOC(ds, g$ in the root of Q we have that $\mu \cup \{[ds \mapsto ds'][g \mapsto g']\} \in Q$.

b) $u \in \mathbf{U} \backslash dom(ep)$: By semantics $\mu = \mu_\emptyset$ which is the empty mapping. This is the same mapping our query $\{\} \leftarrow$ returns and we are done.

c) $u \in \mathbf{V}$:

   By semantics $\mu \in \{\mu_1 \cup [u \to s] \mid s \in dom(ep), \mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))} \wedge [u \to s] \sim \mu_1\}$. Assume $\mu(u) = s$. We know that $s \in dom(ep)$. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. By semantics of SPARQL $\mu = \mu_1 \cup [u \mapsto s]$ for $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By induction hypothesis we can conclude $\mu \in Q_1(D)$. By construction of Q and especially its conjunct in the root $LOC(ds, g)$ we have $(\mu \cup \{[ds \mapsto ds'][g \mapsto g']\}) = \mu_1 \cup \{[ds \mapsto ds'][g \mapsto g'], [u \mapsto s]\} \in Q(D)$.

$\supseteq$:
Let $ds, g \in \mathbf{U}$ and $\mu \in Q(D)$ be arbitrary.

a) Assume u is an URI. By i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$ and thus by construction of the query Q and the fact that $\mu \in Q(D)$, $\mu \in [\![P]\!]^{ep(ds)}_{graph(g,ep(ds))}$

b) Assume u is a variable. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. Assume w.l.o.g. $\mu(u) = s$. $\mu_1 = \mu \backslash [u \mapsto s]$. By induction hypothesis we know that $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By construction of Q especially the conjunct $LOC(u, g)$, $s \in dom(ep)$ and by our induction hypothesis $\mu_1 \sim [u \mapsto s]$ holds. By semantics $\mu \in [\![P]\!]^{ds}_{graph(g,ep(ds))}$ follows.

Let $ds \in \mathbf{V}, g \in \mathbf{U}$ and $\sigma \in Q(D)$ be arbitrary. We need to show $\bigcup_{ds' \in dom(ep)} \{\mu \cup [ds \mapsto ds'] \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}, \mu \sim [ds \mapsto ds']\} \supseteq Q(D)$. Assume that $\sigma(ds) = ds'$, because of the conjunct $LOC(ds, g)$ we have that $ds' \in dom(ep)$.

a) Assume u is an URI. By i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By the fact that $\sigma \in Q(D)$ and the construction of Q we can deduce that for a part of σ, call it μ, $\mu \in Q_1(D)$ holds. By i.h. we get $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$. From this we can deduce $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}$. Because $[ds \mapsto ds'] \in \sigma$ and $\mu \subseteq \sigma$ we have $\mu \sim [ds \mapsto ds']$.

b) Assume u is a variable. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. Let $\mu = \sigma \backslash [ds \mapsto ds']$. We can see that $\mu \in [\![P]\!]^{ep(ds')}_{graph(g,ep(ds'))}$ for some $ds' \in dom(ep)$: Assume w.l.o.g. $\mu(u) = s$. $\mu_1 = \mu \backslash [u \mapsto s]$. By induction hypothesis we know that $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By our construction $s \in dom(ep)$ and $\mu_1 \sim [u \mapsto s]$ holds. It remains to show that $\mu \sim [ds \mapsto ds']$ which follows from the construction of the query Q, i.e., the conjunct $LOC(ds, g)$ in the root of Q and the fact that $\sigma \in Q(D)$.

Let $ds \in \mathbf{U}, g \in \mathbf{V}$ and $\sigma \in Q(D)$ be arbitrary. We need to show $\bigcup_{g' \in names(ep(ds))} \{\mu \cup [g \mapsto g'] \mid \mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}, \mu \sim [g \mapsto g']\} \supseteq Q(D)$. Assume that $\sigma(g) = g'$, because of the conjunct $LOC(ds, g)$ we have that $g' \in names(ep(ds))$.

a) Assume u is an URI. By i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By the fact that $\sigma \in Q(D)$ and the construction of Q we can deduce that for a part of σ, call it μ, $\mu \in Q_1(D)$ holds. By i.h. we get $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$. From this we can deduce $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}$. Because $[g \mapsto g'] \in \sigma$ and $\mu \subseteq \sigma$ we have $\mu \sim [g \mapsto g']$.

b) Assume u is a variable. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. Let $\mu = \sigma \backslash [g \mapsto g']$. We can see that $\mu \in [\![P]\!]^{ep(ds)}_{graph(g',ep(ds))}$: Assume w.l.o.g. $\mu(u) = s$. $\mu_1 = \mu \backslash [u \mapsto s]$. By induction hypothesis we know that $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By our construction $s \in dom(ep)$ and by our induction hypothesis $\mu_1 \sim [u \mapsto s]$ holds. It remains to show that $\mu \sim [g \mapsto g']$ which follows from the construction of the query Q, i.e., the conjunct $LOC(ds, g)$ and the fact that $\sigma \in Q(D)$.

Let $ds, g \in \mathbf{V}$ and $\sigma \in Q(D)$ be arbitrary. We need to show $\bigcup_{ds' \in dom(ep), g' \in names(ep(d))} \{\mu \cup \{[ds \mapsto ds'], [g \mapsto g']\} \mid \mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}, \mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}\} \supseteq Q(D)$. Assume that $\sigma(ds) = ds'$ and $\sigma(g) = g'$, because of the conjunct $LOC(ds, g)$ we have that $ds' \in dom(ep)$ and $g' \in names(ep(ds'))$.

a) Assume u is an URI. By i.h. we know that $[\![P_1]\!]^{ep(u)}_{graph(def,ep(u))} = Q_1(D)$. By the fact that $\sigma \in Q(D)$ and the construction of Q we can deduce that for a part of σ, call it μ, $\mu \in Q_1(D)$ holds. By i.h. we get $\mu \in [\![P_1]\!]^{ep(u)}_{graph(def,ep(u))}$. From this we can deduce $\mu \in [\![P_1]\!]^{ep(ds')}_{graph(g,ep(ds'))}$. Because $\{[ds \mapsto ds'], [g \mapsto g']\} \in \sigma$ and $\mu \subseteq \sigma$ we have $\mu \sim \{[ds \mapsto ds'], [g \mapsto g']\}$.

b) Assume u is a variable. By induction hypothesis we know that $\bigcup_{ds' \in dom(ep)} \{\mu_1 \cup [u \mapsto ds'] \mid \mu_1 \in [\![P_1]\!]^{ep(ds')}_{graph(def,ep(ds'))}, \mu_1 \sim [u \mapsto ds']\} = Q_1(D)$. Let $\mu = \sigma \backslash \{[g \mapsto g'][ds \mapsto ds']\}$. We can see that $\mu \in [\![P]\!]^{ep(ds')}_{graph(g',ep(ds'))}$ for some $ds' \in dom(ep)$ and $g' \in names(ep(ds'))$: Assume w.l.o.g. $\mu(u) = s$. $\mu_1 = \mu \backslash [u \mapsto s]$. By induction hypothesis we know that $\mu_1 \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))}$. By our construction $s \in dom(ep)$ and by our induction hypothesis $\mu_1 \sim [u \mapsto s]$ holds. It remains to show that $\mu \sim \{[ds \mapsto ds'][g \mapsto g']\}$ which follows from the construction of the query Q, i.e., the conjunct $LOC(ds, g)$ and the fact that $\sigma \in Q(D)$. □