SPARQL with Property Paths (version with appendix)

Egor V. Kostylev[1], Juan L. Reutter[2], Miguel Romero[3], and Domagoj Vrgoč[2]

[1] University of Oxford
[2] PUC Chile and Center for Semantic Web Research
[3] University of Chile and Center for Semantic Web Research

Abstract. The original SPARQL proposal was often criticized for its inability to navigate through the structure of RDF documents. For this reason property paths were introduced in SPARQL 1.1, but up to date there are no theoretical studies examining how their addition to the language affects main computational tasks such as query evaluation, query containment, and query subsumption. In this paper we tackle all of these problems and show that although the addition of property paths has no impact on query evaluation, they do make the containment and subsumption problems substantially more difficult.

1 Introduction

Following the initial proposal for the SPARQL 1.0 query language [19] a lot of work has been done by the theory community to study its basic properties. A seminal paper by Pérez et al. [13] gave a clean theoretical foundation for the study of the language, and by now we understand very well the complexity of query evaluation [9, 14], as well as the issues related to basic static analysis tasks such as containment and equivalence [9, 17, 18].

However, with the growth of RDF data available on the web, also came the need for new features not present in the original proposal. One such feature is the ability to navigate RDF documents and discover how different data points are connected. This becomes apparent when considering applications such as linked data where the local topology of the document often does not provide sufficient information and long chains have to be followed to obtain the desired answer. For this reason the W3C decided to include *property paths* into the recommendation proposing SPARQL 1.1 [7], an extension of the original language with several important new functionalities.

The idea of property paths is to searching the RDF graph for IRIs that are connected via a path that conforms to a regular expression. For instance to infer that one property is a subclass of another we could simply ask a query $(?x, \texttt{subclass}^*, ?y)$ and check if our pair is in the answer. Here the $\texttt{subclass}^*$ property path specifies that we traverse an arbitrary number of `subclass` links in order to reach $?y$ starting from $?x$.

Although some work was done on examining the performance of SPARQL with a restricted form of navigation [5,11], and on exploring the expressive power of the language [21], little is known about the properties of the language when property paths as specified in the latest standard [7] are allowed. Therefore, our goal is to study theoretical aspects of SPARQL with full navigational power. In particular, in this paper we focus on the fundamental problems of query evaluation, containment, and subsumption. The first one is key for understanding the properties of any query language, while the latter two are of fundamental importance in query optimization, reasoning over knowledge bases, and understanding incomplete information.

So far, these problems have been studied for SPARQL queries which allow only basic relational operators such as AND, UNION, SELECT, and OPT [6,9,17]. It is therefore interesting to see how do property paths mix with the previous results on core SPARQL. A natural approach here would be to use techniques from graph databases. After all RDF triples closely resemble edges in a labelled graph and property paths are similar to *conjunctive regular path queries* [2]. However we will show that this cannot be done directly as not only is RDF data richer than usual graphs [10], but the SPARQL standard also allows negation in property paths which is known to bring undecidability quite quickly [8, 12]. Another challenge is the presence of OPT operator and the way it interacts with property paths. We will show that techniques for SPARQL without property paths [9, 17] cannot often be straightforwardly adapted to our setting. Because of this we develop new techniques that merge the approaches of [2, 17] and use them to show complexity bounds for the problems we consider.

We begin in Section 3 with a formalisation of property paths according to the latest specification [7] and with incorporating them into SPARQL. We also pinpoint the differences between the resulting language and known formalisms, and discuss the difficulties they impose for adapting known techniques for solving the problems. After that in Section 4 we study evaluation, containment, and subsumption for SPARQL with property paths that do not allow for optional matching. In particular, using techniques from automata theory we show that in this case property paths do not increase the complexity of evaluation, but have a dramatic effect for containment and subsumption. Finally, in Section 5 we study the full language, with both property paths and optional matching. Blending standard SPARQL and graph databases techniques we can show that adding OPT usually makes evaluation more difficult, but almost always leaves the complexity of the optimisation problems intact.

This is an extended version of the paper submitted to ISWC 2015, it has an appendix that contains the proofs of all the statements in the paper.

2 Preliminaries

2.1 RDF Graphs

Let $\mathbf{I}$, $\mathbf{L}$, and $\mathbf{B}$ be countably infinite sets of *IRIs*, *literals*, and *blank nodes*, respectively. The set of *RDF terms* $\mathbf{T}$ is $\mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$. An *RDF triple* is a triple (s, p, o) from $\mathbf{T} \times \mathbf{I} \times \mathbf{T}$, where s is called *subject*, p *predicate*, and o *object*. An *(RDF) graph* is a finite set of RDF triples.

2.2 SPARQL Syntax

SPARQL is the standard pattern-matching language for querying RDF graphs. In what follows we rely on formalisation of the language proposed in [14]. In particular, for now we concentrate on the core fragment, and introduce property paths in a separate section.

Formally, let $\mathbf{V}$ be an infinite set $\{?x, ?y, \ldots\}$ of *variables*, disjoint from $\mathbf{T}$. SPARQL *graph patterns* are defined recursively as follows:

1. a triple in $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$ is a graph pattern, called *triple pattern*;
2. if P_1 and P_2 are graph patterns, then $P_1 \,\mathsf{AND}\, P_2$, $P_1 \,\mathsf{OPT}\, P_2$, and $P_1 \,\mathsf{UNION}\, P_2$ are graph patterns, called AND-, OPT-, and UNION-*patterns*, correspondingly.

The set of all variables appearing in a graph pattern P is denoted by $\mathsf{var}(P)$.

In this paper we do not consider FILTER operator, leaving it for future work. It is also known that arbitrary graph patterns (even without FILTER) may have counter-intuitive behaviour and bad computational properties [14]. That is why we concentrate on a restricted class of graph patterns, which is widely used, has expected behaviour and better properties [14,17], namely well designed patterns [14,16]. Formally, a graph pattern P is *well designed* if it is UNION-free and each of its OPT-subpatterns $P_1 \;\mathsf{OPT}\; P_2$ is such that all the variables in $\mathsf{var}(P_2)$ that appear in P outside of this subpattern are also among $\mathsf{var}(P_1)$.

The class of well designed patterns, denoted $\mathcal{AO}$-SPARQL, is the fundamental frame class for this paper. However, we also consider its restrictions and extensions. In particular, the subclass of $\mathcal{AO}$-SPARQL that allows only for AND-subpatterns is denoted $\mathcal{A}$-SPARQL. It corresponds to conjunctive queries without non-distinguished (existential) variables. These classes extend with UNION operator on the top level to $\mathcal{AOU}$-SPARQL and $\mathcal{AU}$-SPARQL: for example, the patterns in the former have the form $P_1 \,\mathsf{UNION} \ldots \mathsf{UNION}\, P_\ell$ where all P_i are in $\mathcal{AO}$-SPARQL.

Finally, we also consider the SELECT operator which acts as a result modifier of a graph pattern. In particular, SELECT *queries* are expressions of the form

$$\mathsf{SELECT}\, X \;\mathsf{WHERE}\; P,$$

where P is a graph pattern and *distinguished variables* X are a subset of $\mathsf{var}(P)$. A class of SELECT queries with patterns from a class introduced above is denoted by adding $\mathcal{S}$ to the prefix, for example, $\mathcal{AOS}$-SPARQL stands for SELECT queries with well designed patterns. Note that patterns can be seen as queries with all the variables distinguished, so we use "query" as a general term.

2.3 SPARQL Semantics

The semantics of graph patterns is defined in terms of *mappings*, that is, partial functions from variables $\mathbf{V}$ to RDF terms $\mathbf{T}$. The *domain* $\mathsf{dom}(\mu)$ of a mapping μ is the set of variables on which μ is defined. Two mappings μ_1 and μ_2 are *compatible* (written as $\mu_1 \sim \mu_2$) if $\mu_1(?x) = \mu_2(?x)$ for all variables $?x$ that are in both $\mathsf{dom}(\mu_1)$ and $\mathsf{dom}(\mu_2)$. If $\mu_1 \sim \mu_2$, then $\mu_1 \cup \mu_2$ denotes the mapping obtained by extending μ_1 according to μ_2 on all the variables in $\mathsf{dom}(\mu_2) \setminus \mathsf{dom}(\mu_1)$.

Given two sets of mappings M_1 and M_2, the *join*, *union* and *difference* of M_1 and M_2 are defined respectively as follows:

$$\begin{aligned} M_1 \bowtie M_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1 \text{ and } \mu_2 \in M_2 \text{ such that } \mu_1 \sim \mu_2\}, \\ M_1 \cup M_2 &= \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}, \\ M_1 \setminus M_2 &= \{\mu_1 \mid \mu_1 \in M_1 \text{ and there is no } \mu_2 \in M_2 \text{ such that } \mu_1 \sim \mu_2\}. \end{aligned}$$

Based on these, the *left outer join* of M_1 and M_2 is defined as

$$M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2).$$

For a triple pattern P and a mapping $\mu : \mathsf{var}(P) \to \mathbf{T}$ we write $\mu(P)$ for the triple obtained by replacing each variable $?x \in \mathsf{var}(P)$ by $\mu(?x)$. The *evaluation* $[\![P]\!]_G$ of a graph pattern P over a graph G is defined as
1. if P is a triple pattern, then $[\![P]\!]_G = \{\mu : \mathsf{var}(P) \to \mathbf{T} \mid \mu(P) \in G\}$,
2. if $P = P_1$ AND P_2, then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$,
3. if $P = P_1$ OPT P_2, then $[\![P]\!]_G = [\![P_1]\!]_G ⟕ [\![P_2]\!]_G$,
4. if $P = P_1$ UNION P_2, then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$.

Finally, if Q is a SELECT query

$$\mathsf{SELECT}\ X\ \mathsf{WHERE}\ P,$$

then the *evaluation* $[\![Q]\!]_G$ is the set of all *projections* $\mu|_X$ of mappings μ from $[\![P]\!]_G$ to X, that is, mappings coinciding with μ on X and undefined elsewhere.

3 Property Paths in SPARQL

Property paths are a new feature introduced in SPARQL 1.1 [7] to allow for navigational querying over RDF graphs. Intuitively, property paths view RDF documents as labelled graphs where the predicate IRI in each triple acts as an edge label. They then extract the pairs of nodes connected by a path such that the word formed by edge labels along this path belongs to the language of the expression specifying the property path. Property paths resemble regular path queries studied in graph databases [1], but these formalisms have important differences both in syntax and semantics. In this section we formalize the new SPARQL operator according to the specification and compare the resulting extension with known query languages.

3.1 Property Path Expressions

We start with the syntax of property path expressions, following the SPARQL 1.1 specification [7].

Definition 1. Property path expressions *are defined by the grammar*

$$e \ := \ a \mid e^- \mid e_1 \cdot e_2 \mid e_1 + e_2 \mid e^* \mid e^+ \mid e? \mid !\{a_1, \ldots, a_k\} \mid !\{a_1^-, \ldots, a_k^-\},$$

where $a, a_1, \ldots, a_k$ are IRIs in $\mathbf{I}$. Expressions of the last two forms (i.e., starting with !) are called negated property sets*.*

When dealing with singleton negated property sets brackets may be omitted; for example, $!a$ is a shortcut for $!\{a\}$. Besides the expressions in Definition 1 the SPARQL 1.1 specification includes a third version of the negated property sets $!\{a_1, \ldots, a_k, b_1^-, \ldots, b_\ell^-\}$, which allows for negating both normal and inverted IRIs at the same time. We however do not include this extra form in our formalisation, since it is equivalent to the expression $!\{a_1, \ldots, a_k\} + !\{b_1^-, \ldots, b_\ell^-\}$.

The set of all property path expressions is denoted by $\mathbf{PP}$. Their normative semantics is given in the following definition.

Definition 2. *The* evaluation $[\![e]\!]_G$ *of a property path expression e over an RDF graph G is a set of pairs of RDF terms from $\mathbf{T}$ defined as follows:*

$$\begin{aligned}
[\![a]\!]_G &= \{(s, o) \mid (s, a, o) \in G\}, \\
[\![e^-]\!]_G &= \{(s, o) \mid (o, s) \in [\![e]\!]_G\}, \\
[\![e_1 \cdot e_2]\!]_G &= [\![e_1]\!]_G \circ [\![e_2]\!]_G, \\
[\![e_1 + e_2]\!]_G &= [\![e_1]\!]_G \cup [\![e_2]\!]_G, \\
[\![e^+]\!]_G &= \textstyle\bigcup_{i \geq 1} [\![e^i]\!]_G, \\
[\![e^*]\!]_G &= \{(s, s) \mid s \in G\} \cup [\![e^+]\!]_G, \\
[\![e?]\!]_G &= [\![e]\!]_G \cup \{(s, s) \mid s \in G\}, \\
[\![!\{a_1, \ldots, a_k\}]\!]_G &= \{(s, o) \mid \exists p \in \mathbf{I} \text{ with } (s, o) \in [\![p]\!]_G \text{ and } p \notin \{a_1, \ldots, a_k\}\}, \\
[\![!\{a_1^-, \ldots, a_k^-\}]\!]_G &= \{(s, o) \mid (o, s) \in [\![!\{a_1, \ldots, a_k\}]\!]_G\},
\end{aligned}$$

where $\circ$ is a usual composition of binary relations, and $e^i = e \cdot \ldots \cdot e$ with i copies of e on the right hand side of the equation.

Intuitively, two IRIs are connected by a negated property set if they are subject and object of a triple in the graph whose predicate is not mentioned in the set under negation. Note that, according to Definition 2, the expression $!\{a_1^-, \ldots, a_k^-\}$ retrieves the inverse of $!\{a_1, \ldots, a_k\}$, and thus it respects the direction: a negated inverted IRI gives all pairs of nodes connected by some other inverted IRI. To exemplify, consider the RDF graph G from Figure 1. For the expression $!a$ we have that $[\![!a]\!]_G = \{(y, x), (y, z), (v, w)\}$ as we can take a forward looking predicates different than a between respective pairs of IRIs. Note that between v and w there is an a-labelled edge, but since there is also a b-labelled one it is in the answer. On the other hand, for the expression $!a^-$ we have $[\![!a^-]\!]_G = \{(x, y), (z, y), (w, v)\}$, since we can traverse a backward looking predicate (either b^- or c^-) between these pairs of IRIs.

Note that $!\{a_1, \ldots, a_k\}$ is not equivalent to $!a_1 + \ldots + !a_k$. To see this consider again the graph G from Figure 1. We then have $[\![!a]\!]_G = \{(y, x), (y, z), (v, w)\}$ and $[\![!b]\!]_G = \{(x, y), (y, z), (y, v), (v, w)\}$, while $[\![!\{a, b\}]\!]_G = \{(y, z)\}$.

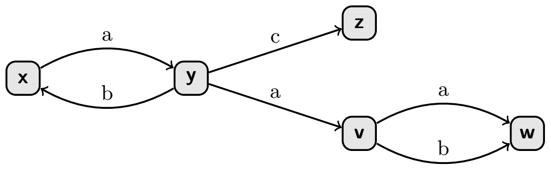

Fig. 1. RDF graph G.

Property path expressions resemble navigational query languages for graph databases. Indeed, syntactically, **PP** without negated property sets are nothing more than the well studied *2-way regular path queries* (*2RPQs*) [2], the default core navigational language for graph databases, with the only minor exception that the empty 2RPQ ε is not expressible as a property path expression (see [1] for a good survey on graph database queries). However, the negated property sets are a unique feature, which has not been properly studied before in the literature, as far as we are aware (the language nSPARQL, considered in [15,21], allows for negation in navigational queries, but in a much more expressive way). Note that if we were working with graph databases, where predicates come from a finite alphabet Σ, then one could easily replace $!a$ with a disjunction of all other symbols in Σ. But since we are dealing with RDF graphs, which have predicates from the infinite set of RDF terms $\mathbf{T}$, we cannot treat this difference in such a naive way. Nevertheless, we can still show that deciding whether a pair of IRIs belongs to the evaluation of a property path expression e over an RDF graph G is as easy as computing the answers of 2RPQs—the problem is in low polynomial time. The idea of the algorithm is in the same spirit as the ideas of standard algorithms for 2RPQs ([1,4]): we construct from G and e two non deterministic finite automata A_e and A_G of special type that can account for negated property sets, and then check that the cross product of these two automata is non empty.

Proposition 1. *For every property path e and RDF graph G the problem of deciding whether a pair (a, b) of terms belongs to $[\![e]\!]_G$ can be solved in time $O(|G| \cdot |e|)$.*

3.2 Queries with Property Paths

SPARQL 1.1 incorporates property path expressions on the atomic level of triple patterns by means of triples with RDF terms or variables on the subject and object positions, but property path expressions on the predicate position. Formally, we have the following definition.

Definition 3. *A* property path pattern *is a triple from* $(\mathbf{T} \cup \mathbf{V}) \times \mathbf{PP} \times (\mathbf{T} \cup \mathbf{V})$.

Note, however, that property path patterns are incomparable with triple patterns, because they allow for property path expressions in predicate positions, but forbid variables in these positions. We use the notion of *atomic patterns* as a general term for triple and property path patterns.

The classes of queries introduced in Section 2.1 incorporate navigational functionality by allowing arbitrary atomic patterns as graph patterns, along with complex operator patterns. In our notation this is reflected by a letter $\mathcal{P}$ in names of the classes. For example $\mathcal{AOUSP}$-SPARQL is the maximal language considered in this paper, which allows for AND, OPT, UNION, SELECT operators and arbitrary atomic patterns. Remember, however, that all the patterns we consider are (unions of) well designed patterns, assuming that for fragments with property paths this notion stays exactly the same as in Section 2, safe the extension on the atomic level.

To complete the formalization of SPARQL with property paths we need to define the semantics.

Definition 4. *For a property path pattern $P = (u, e, v)$ and an RDF graph G the evaluation $[\![P]\!]_G$ of P over G is the set of mappings*

$$\{\mu : \mathsf{var}(P) \to \mathbf{T} \mid (\mu(u), \mu(v)) \in [\![e]\!]_G\},$$

assuming that mappings extend to terms as identity, that is $\mu(t) = t$ for any μ and $t \in \mathbf{T}$.

Having this definition at hand, the semantics of graph patterns and queries with property paths is exactly the same as in Section 2.1.

Since property paths resemble 2RPQs, SPARQL with property paths has a lot in common with other graph database languages, such as *conjunctive 2RPQs* (*C2RPQs*), which extend 2RPQs with conjunction and existential quantification, and *unions of C2RPQs* (*UC2RPQs*), further extending 2RPQs with union on the top level (see again [1]). However, there are some important differences.

The first of these differences is that SPARQL with property paths allows for both property path patterns and triple patterns, which may have a variable in the middle position. This is not possible in (U)C2RPQs.

The second difference is that the UNION operator in SPARQL behaves differently from union in classical databases and UC2RPQs. In particular, it is *not null-rejecting*, that is, the patterns constituting a union may have different sets of variables, and, hence, the mappings in the evaluation may have different domains even if the query is OPT-free.

The third and most important difference is the presence of optional matching in SPARQL. As already mentioned, this unique SPARQL feature requires complete rethinking of many standard results in database theory, and as we will see, results on SPARQL are not an exception.

In the rest of the paper we study properties of the SPARQL classes with property paths. It is convenient to start in the next section with classes without OPT and then continue with the ones incorporating this operator.

4 Properties of Classes without Optional Matching

The fundamental properties of query languages considered in this paper are complexity of query answering and optimisation problems, such as containment and subsumption. We start the study of these properties with OPT-free classes of SPARQL with property paths.

4.1 Query Evaluation

We start with the most important fundamental problem for query languages—query evaluation. According to [20], this problem is formalised for any class $\mathcal{X}$-SPARQL defined in the previous sections as follows.

EVALUATION($\mathcal{X}$-SPARQL)
Input: An RDF graph G, $\mathcal{X}$-SPARQL query Q, and mapping μ.
Question: Does μ belong to $[\![Q]\!]_G$?

As discussed above, the class $\mathcal{AUS}$-SPARQL without optional matching and property paths is just the class of unions of conjunctive queries, for which the evaluation problem is well known to be NP-complete. Without selection, that is, without non-distinguished variables, it is in PTIME. Together with Proposition 1, we can show that adding property paths to OPT-free SPARQL does not affect the complexity of query evaluation, same as adding 2RPQs to conjunctive queries.

Proposition 2. *The following holds:*
– EVALUATION($\mathcal{X}$-SPARQL) *is* NP*-complete for* $\mathcal{X} \in \{\mathcal{ASP}, \mathcal{AUSP}\}$*;*
– EVALUATION($\mathcal{AUP}$-SPARQL) *is in* PTIME*.*

4.2 Query Containment

In this section we consider query containment for OPT-free SPARQL with property paths. This is one of the fundamental problems for static analysis of query languages [20], which asks whether all the answers of one query are among answers of another for any input RDF graph.

Formally, a query Q_1 is *contained* in a query Q_2, denoted by $Q_1 \subseteq Q_2$, if for every RDF graph G we have $[\![Q_1]\!]_G \subseteq [\![Q_2]\!]_G$. Then, the corresponding decision problem is defined as follows for classes of queries $\mathcal{X}_1$-SPARQL and $\mathcal{X}_2$-SPARQL.

CONTAINMENT($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL)
Input: Queries Q_1 from $\mathcal{X}_1$-SPARQL and Q_2 from $\mathcal{X}_2$-SPARQL.
Question: Is $Q_1 \subseteq Q_2$?

It is known that containment of 2RPQs and C2RPQs without projection is PSPACE-complete [3], and EXPSPACE-complete if projection is allowed. Given the resemblance of 2RPQs and property paths, it is natural to ask whether the techniques of [3] and [2] can be reused in the context of SPARQL with property paths. It turns out that, to some extend, this is indeed the case, but the nature of triples in RDF graphs and the presence of negated property sets oblige us to rework most of their definitions, including the key one—“canonical database”, adapting them to the SPARQL scenario. We illustrate some of the ideas behind this construction by means of the following two examples.

Example 1. Consider $\mathcal{ASP}$-SPARQL queries

$$Q_1 = \mathsf{SELECT}\ ?x, ?y, ?z\ \mathsf{WHERE}\ (?x, !a, ?y)\ \mathsf{AND}\ (?x, !a, ?z),$$
$$Q_2 = \mathsf{SELECT}\ ?x, ?y, ?z\ \mathsf{WHERE}\ (?x, ?v, ?y)\ \mathsf{AND}\ (?x, ?v, ?z).$$

One can easily check that Q_1 is not contained in Q_2. However, a counterexample for this fact requires a graph in which images of $?x$ and $?y$ are connected by a different property than those of $?x$ and $?z$. It means that we cannot treat $!a$ just as a usual RDF term, but we need to allow each occurrence of a negated property set to be witnessed by a fresh term.

Example 2. Consider now $\mathcal{ASP}$-SPARQL queries

$$Q_3 = \mathsf{SELECT}\ ?x, ?y\ \mathsf{WHERE}\ (?x, ?v, ?y),$$
$$Q_4 = \mathsf{SELECT}\ ?x, ?y\ \mathsf{WHERE}\ (?x, !a, ?y).$$

Again, Q_3 is not contained in Q_4. This time, however, counterexamples are formed by triples of the form (b, a, c), for IRIs b and c. Thus, we cannot just construct a canonical graph by *freezing* every variable in the query on the left, because counterexamples may need to be formed by mapping some of these variables to negated IRIs from the query on the right.

The idea, then, is to take into account these challenges and transform the machinery in [2] so that the notion of canonical graphs can be adapted to SPARQL queries and can be generalised to queries with limited negation. Then, using automata techniques, we can prove results similar to [2] for containment of OPT-free SPARQL with property paths—it is EXPSPACE-complete in general and PSPACE-complete if the right-hand side query is a pattern without projection.

Theorem 1. *The following holds:*

- Containment($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* EXPSPACE-*complete for* $\mathcal{X}_1 \in \{\mathcal{AP}, \dots, \mathcal{AUSP}\}$ *and for* $\mathcal{X}_2 \in \{\mathcal{ASP}, \mathcal{AUSP}\}$*;*
- Containment($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* PSPACE-*complete for* $\mathcal{X}_1 \in \{\mathcal{AP}, \dots, \mathcal{AUSP}\}$ *and for* $\mathcal{X}_2 \in \{\mathcal{AP}, \mathcal{AUP}\}$.

To conclude, we note that in the first case the space used depends exponentially only on the size of each of the union-free subpatterns and not on the number of these subpatterns. This property is crucial for the results of Section 5.3 (in particular, Theorem 3).

4.3 Query Subsumption

Query containment is a way of specifying that one query is more general than another, which is common across different query formalisms. However, the unique SPARQL feature is the ability to return partial answers, and Pérez et al. argued in [14] that it is more natural to compare SPARQL queries for subsumption, that is, to check whether for any answer to one query there is a more elaborate answer to the other one on any input RDF graph.

Formally, a mapping μ is *subsumed* by a mapping μ', denoted by $\mu \sqsubseteq \mu'$, if $\mathsf{dom}(\mu)$ is contained in $\mathsf{dom}(\mu')$ and $\mu \sim \mu'$. A query Q_1 is *subsumed* by a query Q_2 (written as $Q_1 \sqsubseteq Q_2$) if for every RDF graph G it holds that for each $\mu_1 \in [\![Q_1]\!]_G$ there exists $\mu_2 \in [\![Q_2]\!]_G$ such that $\mu_1 \sqsubseteq \mu_2$. The corresponding decision problem is defined as follows for classes of queries $\mathcal{X}_1$-SPARQL and $\mathcal{X}_2$-SPARQL.

SUBSUMPTION($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL)
Input: Queries Q_1 from $\mathcal{X}_1$-SPARQL and Q_2 from $\mathcal{X}_2$-SPARQL.
Question: Is $Q_1 \sqsubseteq Q_2$?

Although the notion of subsumption becomes most natural when dealing with the OPT operator, we still study this problem for the case of OPT-free SPARQL queries with property paths, both for completion and for the fact that the complexity of subsumption ends up being higher than the complexity of containment for some of the classes.

Before stating the results on subsumption, we give some intuition behind them and compare subsumption with containment. For a query Q_1 from $\mathcal{ASP}$-SPARQL to be subsumed by a query Q_2 from this class it is necessary that the set of distinguished variables of Q_1 is a subset of the distinguished variables of Q_2. Moreover, $Q_1 \sqsubseteq Q_2$ if and only if for every RDF graph G and μ_1 in $[\![Q_1]\!]_G$ one can obtain μ_1 from some μ_2 in $[\![Q_2]\!]_G$ by projecting out the distinguished variables of Q_2 that are not distinguished in Q_1. The first obvious consequence of this observation is that in this case the subsumption problem for $\mathcal{ASP}$-SPARQL is not more difficult than the containment problem, because $Q_1 \sqsubseteq Q_2$ if and only if Q_1 is contained in SELECT X WHERE P_2, with X the set of output variables of Q_1 and P_2 is the pattern of Q_2. However, rather surprisingly, the limited projection inherent to the subsumption problem is enough to make the problem EXPSPACE-hard even for patterns from $\mathcal{AP}$-SPARQL, which do not have non distinguished variables.

Proposition 3. *The problem* SUBSUMPTION($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* EXPSPACE-*complete for* $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{AP}, \ldots, \mathcal{AUSP}\}$.

5 Properties of Classes with Optional Matching

In this section we consider query evaluation, containment, and subsumption for SPARQL classes that allow for both the OPT operator and property paths. In addition to the difficulties from the previous section, such as negated property sets and non null-rejecting union, we have to deal with mixture of optional matching and property paths. To overcome these difficulties we have to develop non-trivial compositions of the usual SPARQL and graph databases techniques, as well as invent new ones.

5.1 Query Evaluation

Bounds for query evaluation of SPARQL classes with (well designed) optional matching that do not use property paths are by now well understood [9, 14]. In particular, the problem is coNP-complete for graph patterns, that is, queries with all the variables distinguished, and jumps to Σ_2^p if non-trivial SELECT clauses are allowed. In this section we show that adding property paths to the set of allowed operators preserves these bounds. To this end, we develop a characterisation similar to the one in [9], by adapting the notions of OPT normal form and pattern trees.

A graph pattern P is in OPT *normal form* if no OPT operators appear in AND-subpatterns of P. It was shown in [14, Proposition 4.11] that every well designed graph pattern without property path patterns can be transformed to an equivalent pattern in OPT normal form in polynomial time by means of a set of rewriting rules that "push" AND inside OPT (recall that well designed patterns have neither UNION nor SELECT clause). It is straightforward to check that these rules are correctly applicable to graph patterns that allow for property paths, so in what follows we assume that all patterns are in OPT normal form (in particular, AND-patterns are just AND combinations of atomic patterns).

Each graph pattern P in OPT normal form can be intuitively represented as a *pattern tree* $\mathit{Tree}(P)$, that is, a rooted tree with nodes labelled by sets of atomic (i.e., triple and property path) patterns that is recursively constructed as follows:

– if P is an AND-pattern then $\mathit{Tree}(P)$ consists of a single node labelled with the set of all atomic patterns in this AND-pattern;
– if $P = P_1$ OPT P_2 then $\mathit{Tree}(P)$ is obtained from $\mathit{Tree}(P_1)$ and $\mathit{Tree}(P_2)$ by adding an edge form the root of the former to the root of the latter.

In other words, the labels of nodes in pattern trees correspond to conjunction of atomic patterns, while edges represent the structure of optional matching. For a node n in a pattern tree, $\mathsf{and}(n)$ denotes the AND pattern consisting of the atomic patterns in its label, and $\mathsf{var}(n)$ denotes the set of all variables in these patterns, and these notations propagate to sets of nodes and subtrees of pattern trees. In fact, we concentrate only on subtrees that contain the root of the original tree, so in what follows we assume this restriction without mentioning it explicitly. A node in a pattern tree is a *child* of a subtree T if it is not in T but its parent is.

It is important to note that pattern trees are unordered, so different patterns may have the same representation. However, we disregard this syntactical mismatch, because such patterns are always equivalent. This follows from the fact that for well designed patterns $((P_1 \text{ OPT } P_2) \text{ OPT } P_3)$ is equivalent to $((P_1 \text{ OPT } P_3) \text{ OPT } P_2)$ (this was first shown in [13] for patterns without property paths, and a generalisation to our case is straightforward).

Example 3. Consider the pattern

$$((((?x, a+b, ?y) \text{ AND } (?x, c^*, ?y)) \text{ OPT } (?x, a, ?z)) \text{ OPT } (?x, b, ?w)).$$

The tree representing this pattern is as follows.

$$\begin{tikzcd}
 & \{(?x, a+b, ?y), (?x, c^*, ?y)\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{(?x, a, ?z)\} & & \{(?x, b, ?w)\}
\end{tikzcd}$$

Another interesting property of pattern trees is that for each variable the set of all nodes with this variable in the labels is always connected. This is a vivid illustration of the well-designedness property of patterns. Moreover, every pattern tree $\mathit{Tree}(P)$ (and hence every well-designed pattern) can be normalised to an equivalent tree T' (called *NR normal form* [14]) such that $\mathsf{var}(n') \not\subseteq \mathsf{var}(n)$ for every edge (n, n') in T', that is, such that every node introduces a new variable in comparison to the parent of this node. Transformation to NR normal form can be done in polynomial time by adding the label of every node without new variables to the labels of its children and then removing all such nodes from the tree. In what follows we assume all well designed patterns and corresponding pattern trees to be in NR normal form.

An intuitive coNP algorithm for evaluation of well designed patterns with property paths works in the same way as the one described in [9] for the case without property paths. It consists in the following two steps. Since the input pattern is in NR normal form, the input mapping μ uniquely defines a subtree T'_μ such that $\mathsf{dom}(\mu) = \mathsf{var}(T'_\mu)$. So, on the first step we need to check for the input graph that $\mu(\mathsf{and}(T'_\mu)) \subseteq G$, that is, all the patterns in the subtree under μ indeed materialise in the input graph G. This check can be done in polynomial time, because property paths have tractable evaluation (Proposition 1). Finally, on the second and most difficult step we need to guarantee that μ cannot be consistently extended to the variables of any child of T'_μ in T'. This can be done in coNP by guessing a counterexample (i.e., an extension) for one of these children.

Same as for patterns without property paths, this algorithm can be extended to union and selection in a straightforward way. In the latter case the complexity jumps one level of the polynomial hierarchy, because we have to guess the values of non-distinguished variables. Combining these results with matching lower bounds for the classes without property paths [9, 14] we summarise with the following proposition.

Proposition 4. *The following holds:*
– Evaluation($\mathcal{X}$-SPARQL) *is* Σ_2^p*-complete for* $\mathcal{X} \in \{\mathcal{AOSP}, \mathcal{AUOSP}\}$*;*
– Evaluation($\mathcal{X}$-SPARQL) *is* coNP*-complete for* $\mathcal{X} \in \{\mathcal{AOP}, \mathcal{AUOP}\}$*.*

The focus of this paper is SPARQL with well designed optional matching, and we leave a comprehensive study of SPARQL with property paths and arbitrary nesting of other operators considered in this paper for future work. However, as a final remark in this section, we note that it is not difficult to show PSPACE-completeness of evaluation for this class, that is, the same complexity as for any subclass of this class that allows for arbitrary optional matching [18].

5.2 Query Containment

Now we move to the containment problem of SPARQL with property paths. As shown in [17], without them the problem CONTAINMENT($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) is NP-complete for any $\mathcal{X}_1$-SPARQL that allows for optional matching and for $\mathcal{X}_2$-SPARQL = $\mathcal{AO}$-SPARQL, that is for the class of well designed patterns. If $\mathcal{X}_2$-SPARQL also allows for union, then the complexity becomes Π_2^p-complete (again, for the full range of $\mathcal{X}_1$-SPARQL), and the problem is undecidable if $\mathcal{X}_2$-SPARQL allows for arbitrary selection. Thus we focus on the most general case where we can hope for decidability: checking whether a query in $\mathcal{AOUSP}$-SPARQL is contained in a query in $\mathcal{AOUP}$-SPARQL. Our main result is that this problem is also decidable, specifically, EXPSPACE-complete.

As we saw in the previous subsection, the techniques developed in [9, 14] for checking evaluation can be extended to work with property paths with relatively little effort. Later we will see that similar strategy works for subsumption, because it can be reduced to checking containment of OPT-free queries, which is extensible to classes with property paths. However, the situation is different for containment. It is not clear how to apply known techniques [17, Theorem 3.7] to state the problem in terms of containment of OPT-free queries. To overcome this, we develop a new characterization of containment that reduces the problem to a weaker form of containment between OPT free queries. Then we take advantage of the automata techniques developed in Section 4.

In what follows we first present our new characterisation for containment for queries without property paths (which we believe is of independent interest) and then adapt it to the general case. We start with a general definition.

Definition 5. *Let*

$$Q_1 = \mathsf{SELECT}\, X \ \mathsf{WHERE}\ P \quad \textit{and} \quad Q_2 = P^1\ \mathsf{UNION}\ \ldots\ \mathsf{UNION}\ P^k$$

be queries from $\mathcal{AOSP}$-SPARQL and $\mathcal{AOUP}$-SPARQL correspondingly, with P, P^i well designed patterns. A good extension *E of Q_1 over Q_2 is an* AND *pattern*

$$\mathsf{and}(\mathit{Tree}(P))\ \ \mathsf{AND}\ \ \mathsf{and}(n_1)\ \mathsf{AND} \ldots \mathsf{AND}\ \mathsf{and}(n_m),$$

where (1) every n_i is obtained from a child of a subtree T_i of one of $\mathit{Tree}(P^1), \ldots,$ $\mathit{Tree}(P^k)$ with $\mathsf{var}(T_i) = X$ by renaming all variables not in X to fresh ones, and (2) distinct n_i's are obtained from distinct T_i's. The support *$\mathsf{sup}(E)$ of E is the set of all subtrees T_i.*

The new characterisation for the case without property paths is based on the following lemma.

Lemma 1. *Let*

$$Q_1 = \mathsf{SELECT}\, X \ \mathsf{WHERE}\ P \quad \textit{and} \quad Q_2 = P^1\ \ \mathsf{UNION} \ldots \mathsf{UNION}\ \ P^k$$

be a $\mathcal{AOUS}$-SPARQL and $\mathcal{AOU}$-SPARQL queries correspondingly. Then $Q_1 \not\subseteq Q_2$ if and only if there is a good extension E over Q_2 of some $\mathcal{AOSP}$-SPARQL *query with a pattern P^* such that Tree(P^*) is a subtree of one of the trees representing components of P and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ that satisfies the following conditions:*

(C1) *for each child n of Tree(P^*), there is no homomorphism h from $\mathsf{and}(n)$ to E such that $h(?x) = ?x$ for all variables $?x$ in $\mathsf{var}(n) \cap \mathsf{var}(E)$, and*

(C2) *for each subtree T of one of Tree(P^1), ..., Tree(P^k) with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$ there is no homomorphism h from $\mathsf{and}(T)$ to E such that $h(?x) = ?x$, for all variables $?x$ in $\mathsf{var}(T) \cap \mathsf{var}(E)$.*

The intuition behind Lemma 1 is as follows. A good extension E satisfying conditions (C1) and (C2) gives us a witness for non-containment: it suffices to consider the "frozen RDF graph" G of E obtained by replacing each variable $?x$ by a fresh IRI a_x and the mapping μ with $\mu(?x) = a_x$, for all $?x \in X^*$ and undefined for other $?x$. Then conditions (C1) and (C2) are a convenient way of stating that $\mu \in [\![Q_1]\!]_G$ and $\mu \notin [\![Q_2]\!]_G$.

Observe that the size of a good extension is polynomial in the size of Q_1 and Q_2. Thus Lemma 1 gives us an alternative proof for Π_2^p-membership of containment of a SPARQL query a pattern if both of them do not use property paths. Indeed to find a counterexample for containment we need to guess a good extension and then call for a coNP oracle to check conditions (C1) and (C2).

To extend the characterisation of Lemma 1 to queries with property paths we need the following auxiliary notion. An AND pattern P_1 is *l-compatible* with a union of AND patterns P_2 (written $P_1 \preceq P_2$) if for each RDF graph G and a mapping $\mu_1 \in [\![P_1]\!]_G$ there is a mapping $\mu_2 \in [\![P_2]\!]_G$ such that $\mu_1 \sim \mu_2$.

We analyse the complexity of containment in the presence of property paths by means of the following generalised statement.

Lemma 2. *Let*

$$Q_1 = \mathsf{SELECT}\ X\ \mathsf{WHERE}\ P \quad \textit{and} \quad Q_2 = P^1\ \mathsf{UNION} \ldots \mathsf{UNION}\ P^k$$

be a $\mathcal{AOUSP}$-SPARQL and $\mathcal{AOUP}$-SPARQL queries correspondingly. Then $Q_1 \not\sqsubseteq Q_2$ if and only if there is a good extension E over Q_2 of some $\mathcal{AOSP}$-SPARQL query with a pattern P^* such that Tree(P^*) is a subtree of one of the trees representing components of P and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ that satisfies $E \not\preceq N$ UNION S, where

(C1') *N is a union of all $\mathsf{and}(n)$ for n a child of Tree(P^*), and*

(C2') *S is a union of all $\mathsf{and}(T)$ for T a subtree of one of Tree(P^1), ..., Tree(P^k) with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$.*

The condition $E \not\preceq N \cup S$ can be checked in EXPSPACE, using techniques developed in Section 4.2. This gives us an EXPSPACE upper bound for containment of $\mathcal{AOUSP}$-SPARQL and $\mathcal{AOUP}$-SPARQL. Moreover, the matching lower bound can be derived from Proposition 3.

Theorem 2. *The problem* Containment($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* EXPSPACE*-complete for $\mathcal{X}_1 \in \{\mathcal{AOP}, \ldots, \mathcal{AOUSP}\}$ and $\mathcal{X}_2 \in \{\mathcal{AOP}, \mathcal{AOUP}\}$.*

5.3 Query Subsumption

The last problem we study in this paper is subsumption of SPARQL queries with property paths. Letelier et al. [9, 17] proved Π_2^p-completeness of this problem for all the classes with optional matching but without property paths, even if selection is allowed. Moreover, they provide the following very simple and useful characterisation for the subsumption of $\mathcal{AO}$-SPARQL patterns: a pattern P_1 is subsumed by a pattern P_2 if and only if for every subtree T_1' of $\mathit{Tree}(P_1)$ there is a subtree T_2' of $\mathit{Tree}(P_2)$ such that $\mathsf{var}(T_1') \subseteq \mathsf{var}(T_2')$ and there is a homomorphism from $\mathsf{and}(T_2')$ to $\mathsf{and}(T_1')$ that is the identity over $\mathsf{var}(T_1')$. This idea extends to patterns with union in the usual way—the subsumption holds if and only if for every component of the first pattern there is a subsuming one in the second.

How can this characterisation be extended to deal with property paths? The immediate idea is just to replace homomorphism with containment of corresponding OPT-free queries. However, in the presence of union this simple strategy does not always work. Indeed, the pattern $(?x, (a + b), ?y)$ is subsumed by the pattern $(?x, a, ?y)$ UNION $(?x, b, ?y)$ (in fact, they are equivalent), but not in any of its components.

As we see, the problem is the disjunction introduced by property paths, and our characterisation needs to account for this. By doing so we arrive at the following characterisation. A pattern P_1 is subsumed by a pattern P_2 if and only if for every subtree T_1' of $\mathit{Tree}(P_1)$ the AND-pattern $\mathsf{and}(T_1')$ is subsumed in the union of all AND-patterns $\mathsf{and}(T_2')$, where T_2' range over subtrees of $\mathit{Tree}(P_2)$ with $\mathsf{var}(T_1') \subseteq \mathsf{var}(T_2')$. With this characterisation we avoid dealing with optional matching, and can thus solve subsumption by the techniques introduced in the previous section. As an illustration, we can use this characterisation in the example above to show that $Q_1 \sqsubseteq Q_2$, by choosing the same query $(?x, a, ?y)$ UNION $(?x, b, ?y)$. By extending this characterisation for all queries that use non-trivial selection we obtain our last theorem.

Theorem 3. *The problem* SUBSUMPTION($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* EXPSPACE-*complete for* $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{AOP}, \ldots, \mathcal{AOUSP}\}$.

6 Conclusions

At a first glance it was not clear whether one could combine techniques from graph databases and Semantic Web to complete a study of SPARQL queries with property paths. In particular, graph database techniques failed short for studying proper RDF queries, since these deal with infinite alphabets and include negative assertions. On the other hand, the machinery developed to study SPARQL queries without property paths proved to be very useful and a clear inspiration for this work, but again the characterisations provided in the literature were too specific and needed to be generalized to be used in our context. In this paper we show how these two classes of techniques can be extended and combined to reason about SPARQL queries that allow property path patterns and based on this work develop algorithms for evaluating such queries and deciding their containment and subsumption. Finally we would like to note that many of the results obtained here (in particular all EXPSPACE and Π_2^p bounds) are optimal in light of the lower bounds fore more restricted classes of queries.

As for future work, the main direction we would like to tackle is the addition of the FILTER operator to the language since this feature of SPARQL was not previously considered in the literature. We have some preliminary results showing that the techniques from Section 5.2 can be extended to work in this setting.

References

1. P. Barceló Baeza. Querying graph databases. In *PODS'13*, pages 175–188.
2. D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'2000*, pages 176–185.
3. D. Calvanese, G. De Giacomo, M. Lenzerini, M.Y. Vardi. Reasoning on regular path queries. *ACM SIGMOD Record*, 32(4):83–92, 2003.
4. M. Consens, A. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS'90*, pages 404–416.
5. M.W. Chekol. Schema Query Containment. INRIA Research Report, 2014.
6. M. W. Chekol, J. Euzenat, P. Genevès, N. Layaïda. SPARQL Query Containment under RDFS Entailment Regime. In *IJCAR 2012*.
7. *SPARQL 1.1 Query Language*. `http://www.w3.org/TR/sparql11-query`.
8. E. V. Kostylev, J. L. Reutter, D. Vrgoč. Containment of Data Graph Queries. In *ICDT 2014*, pages 131–142.
9. A. Letelier, J. Pérez, R. Pichler and S. Skritek. Static analysis and optimization of semantic web queries. In *ACM TODS*, 38(4), 2013.
10. L. Libkin, J. L. Reutter and D. Vrgoč. Trial for RDF: adapting graph query languages for RDF data. In *PODS 2013*, pages 201–212.
11. K. Losemann, W. Martens. The Complexity of Regular Expressions and Property Paths in SPARQL. In *ACM TODS*, 38(4), 2013.
12. F. Neven, T. Schwentick, V. Vianu. Finite state machines for strings over infinite alphabets. *ACM TOCL* 5(3): 403–435 (2004).
13. J. Pérez, M. Arenas, C. Gutierrez. Semantics and Complexity of SPARQL. In *ISWC* 2006.
14. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3), 2009.
15. J. Pérez, M. Arenas, C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.
16. F. Picalausa, S. Vansummeren. What are real SPARQL queries like? In *SWIM'11*.
17. R. Pichler, S. Skritek. Containment and equivalence of well-designed SPARQL. In *PODS* 2014.
18. M. Schmidt, M. Meier, G. Lausen. Foundations of SPARQL query optimization. In *ICDT*, 2010.
19. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, `http://www.w3.org/TR/rdf-sparql-query/`.
20. M. Y. Vardi. The Complexity of Relational Query Languages. In *STOC*, 1982.
21. X. Zhang and J. Van den Bussche. On the Power of SPARQL in Expressing Navigational Queries. In *The Computer Journal*, 2014.

Appendix: Proofs

In this appendix we provide missing details of the proofs of all the results and provide more thorough definition of several concepts that are used in some of the proofs presented below.

Proposition 1. *For every property path e and RDF graph G the problem of deciding whether a pair (a, b) of terms belongs to $[\![e]\!]_G$ can be solved in time $O(|G| \cdot |e|)$.*

Proposition 2. *The following holds:*
- EVALUATION($\mathcal{X}$-SPARQL) *is* NP*-complete for* $\mathcal{X} \in \{\mathcal{ASP}, \mathcal{AUSP}\}$*;*
- EVALUATION($\mathcal{AUP}$-SPARQL) *is in* PTIME*.*

Proof. This proposition is an immediate corollary of Proposition 1. Indeed, if we do not have selection, then we can run a polynomial evaluation check for every atomic pattern of the query. If we have selection, then we can guess the values of non-distinguished variables in $\mathsf{NP}$and then run the polynomial algorithm. Finally, the lower bound follows from NP-completeness of evaluation of conjunctive queries.

Theorem 1. *The following holds:*
- CONTAINMENT($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* $\mathsf{EXPSPACE}$*-complete for* $\mathcal{X}_1 \in \{\mathcal{AP}, \dots, \mathcal{AUSP}\}$ *and for* $\mathcal{X}_2 \in \{\mathcal{ASP}, \mathcal{AUSP}\}$*;*
- CONTAINMENT($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* PSPACE*-complete for* $\mathcal{X}_1 \in \{\mathcal{AP}, \dots, \mathcal{AUSP}\}$ *and for* $\mathcal{X}_2 \in \{\mathcal{AP}, \mathcal{AUP}\}$*.*

Proof. For readability we assume that our queries contain no constants (it is straightforward to include them in this proof, but this makes the proof considerably more verbose).

Let us first define the notion of a canonical model for queries in $\mathcal{ASP}$-SPARQL.

A simple *semipath* (or just semipath) from an RDF term x to an RDF term y in an RDF graph G is a is a sequence of form

$$(u_0, p_1, u_1), (u_1, p_2, u_2), \dots, (u_{n-1}, p_n, u_n),$$

where $u_0 = x$, $u_n = y$, all the remaining u_i's are different, and such that for each (u_{i-1}, p_i, u_i) in the semipath either (u_{i-1}, p_i, u_i) or (u_i, p_i, u_{i-1}) belong to G.

We say that a semipath s from x to y in G *canonically conforms* to a property path e with respect to a set Σ of IRIs if the following holds.

- If $e = a$ then s is the triple (x, a, y).
- If $e = a^-$ then s is the triple (y, a, x).
- If $e = e_1 \cdot e_2$ then s can be divided as two semipaths, one s_1 from x to some IRI z that canonically conforms to e_1, and s_2 from z to y that canonically conforms to e_2.
- If $e = e_1 + e_2$ then s canonically conforms to either e_1 or to e_2.

- If $e = e_1^*$ then either $x = y$ or s is a semipath that canonically conforms to the expression $e_1 + e_1 \cdot e_1 + e_1 \cdot e_1 \cdot e_1 + \cdots$.
- If $e = e_1^+$ then s is a semipath that canonically conforms to the expression $e_1 + e_1 \cdot e_1 + e_1 \cdot e_1 \cdot e_1 + \cdots$.
- If $e = e_1?$ then either $x = y$ or s is a semipath that canonically conforms to the expression e_1.
- If $e = !\{a_1, \dots, a_k\}$ then either s is the triple (x, b, y) for some $b \in \Sigma$, $b \notin \{a_1, \dots, a_k\}$ or s is the triple (x, n, y) for a fresh IRI n not used elsewhere in G.
- If $e = !\{a_1^-, \dots, a_k^-\}$ then either s is the triple (y, b, x) for some $b \in \Sigma$, $b \notin \{a_1, \dots, a_k\}$ or s is the triple (y, n, x) for a fresh IRI n not used elsewhere in G.

From now on we assume that all queries in $\mathcal{ASP}$-SPARQL are of form $Q = \mathsf{SELECT}\, w_1, \dots, w_k \ \mathsf{WHERE}\ (Q_T \ \ \mathsf{AND}\ \ Q_P)$, where

$$Q_T = (x_1, y_1, z_1)\ \mathsf{AND}\ (x_2, y_2, z_2)\ \mathsf{AND} \dots \mathsf{AND}\ (x_n, y_n, z_n),$$
$$Q_P = (u_1, e_1, v_1)\ \mathsf{AND}\ (u_2, e_2, v_2)\ \mathsf{AND} \dots \mathsf{AND}\ (u_m, e_m, v_m).$$

where all x_i, y_i, z_i, u_j and v_j, $q \leq i \leq n$ and $1 \leq i \leq m$ are variables or IRIs, and each e_j, $1 \leq j \leq m$ is a property path expression.

Let ν be a function from the variables of a $\mathcal{ASP}$-SPARQL query Q to the IRIs of an RDF graph G, and Σ a set of IRIs. We say that G is (ν, Σ)-canonical for Q if it consists of

1. the triple $(\nu(x_i), \nu(y_i), \nu(z_i))$ for each (x_i, y_i, z_i) in Q_T;
2. one semipath for each of the m conjunts (u_j, e_j, v_j) of Q_P, such that all these semipaths are subject and object disjoint, i.e., the only subjects and objects that can be shared from these triples are the images of the starting and ending IRIs of each semipath;
3. for each conjunct (u_j, e_j, v_j) of Q_P, the associated semipath starts in $\nu(u_j)$, ends in $\nu(v_j)$, and canonically conforms to e_j with respect to Σ.

For the remainder of the proof we shall be dealing with two $\mathcal{ASP}$-SPARQL queries Q_1 and Q_2. We assume that they are of the form $Q^\ell = \mathsf{SELECT}\, w_1, \dots, w_k \ \mathsf{WHERE}\ Q_T^\ell \ \mathsf{AND}\ Q_P^\ell$, where

$$Q_T^\ell = (x_1^\ell, y_1^\ell, z_1^\ell)\ \mathsf{AND}\ (x_2^\ell, y_2^\ell, z_2^\ell)\ \mathsf{AND} \dots \mathsf{AND}\ (x_{n_\ell}^\ell, y_{n_\ell}^\ell, z_{n_\ell}^\ell),$$

$$Q_P^\ell = (u_1^\ell, e_1^\ell, v_1^\ell)\ \mathsf{AND}\ (u_2^\ell, e_2^\ell, v_2^\ell)\ \mathsf{AND} \dots \mathsf{AND}\ (u_{m_\ell}^\ell, e_{m_\ell}^\ell, v_{m_\ell}^\ell),$$

where each $x_i^\ell, y_i^\ell, z_i^\ell, u_j^\ell$ and v_j^ℓ, for $1 \leq i \leq n_\ell$ and $1 \leq i \leq m_\ell$ and $1 \leq \ell \leq 2$ are variables or IRIs, and each e_j^ℓ is a property path.

Consider a graph G that is ν-canonical for a $\mathcal{ASP}$-SPARQL query Q_1 with respect to Σ, and consider a $\mathcal{ASP}$-SPARQL query Q_2 with the same distunguished variables as Q_1. Then Q_2 can be (ν, Σ)-*mapped* to G if there is a function γ from the variables of Q_2 to the domain of G such that:

- $\gamma(?x) = \nu(?x)$ for each distinguished variable of Q_2 (and thus of Q_1);

– for each triple (x, y, z) in Q^2_T we have that $(\gamma(x), \gamma(y), \gamma(z))$ is in G;
– for each triple (u, e, v) of Q^2_P we have that $(\gamma(u), \gamma(v)) \in [\![e]\!]_G$.

Claim. Let Q^1 and Q^2 be two $\mathcal{ASP}$-SPARQL queries. Then $Q^1 \not\subseteq Q^2$ if and only if there is an RDF graph G and a function ν from the variables of Q^1 to the domain of G such that (1) G is (ν, Σ)-canonical for Q^1, and (2) Q^2 cannot be (ν, Σ)-mapped to G, where Σ is the union of the set of labels that are mentioned in a property path in either Q^1 or Q^2 and the set $\{\nu(?x) \mid ?x$ is a variable of $Q^1\}$.

Proof. Note that the fact that G is (ν, Σ) canonical for Q^1 implies that $(\nu(w_1), \dots, \nu(w_k))$ belongs to $[\![Q^1]\!]_G$.

One direction is immediate: assuming that there exists a graph G and mapping ν that satisfy the second half of the claim, and assuming for the sake of contradiction that $Q^1 \subseteq Q^2$, it must be that $(\nu(w_1), \dots, \nu(w_k))$ belongs to $[\![Q^2]\!]_G$. From this fact one obtains that there must be a mapping γ so that Q_2 can be (ν, Σ)-mapped onto G.

For the other direction, assume that $Q^1 \not\subseteq Q^2$. Then there is a graph G and a tuple $(a_1, \dots, a_k)$ of IRIs that belong to $[\![Q^1]\!]_G$ but not to $[\![Q^2]\!]_G$. From the fact that it belongs to $[\![Q^1]\!]_G$, we know that there must be a homomorphism ν from Q^1 to G so that $(\nu(w_1), \dots, \nu(w_k)) = (a_1, \dots, a_k)$ for distinguished variables $w_1, \dots, w_k$ of Q^1 and Q^2.

We now construct a graph G' by means of the following operations.

– For each triple (u^1_j, e^1_j, v^1_j) in Q^1_P, choose a word $s = s_1, \dots, s_p$ in the language of e^1_j so that $(\nu(u^1_j), \nu(v^1_j))$ belongs to the evaluation of (u^1_j, s, v^1_j) over G (i.e., choose a path from $(\nu(u^1_j)$ to $\nu(v^1_j))$ that witnesses the conformance to e^1_j in G. Define $\tau(s_i)$, $1 \leq i \leq p$ to be s_i, if s_i is either an IRI mentioned in a property path in Q^1 or Q^2, or one of the IRIs that are part of the image of ν; or otherwise $\tau(s_i) = \bot_i$, where $\bot_i$ is a fresh IRI.
– Choose $p - 1$ fresh iris $N_1, \dots, N_{p-1}$, and add to G the triples $(\nu^1_j, \tau(s_1), N_1), (N_1, \tau(s_2), N_2), \dots, (N_{p-1}, \tau(s_p), \nu(v^1_j))$. Note that by doing this we do not increase the number of tuples in $[\![Q^2]\!]_G$. This can be shown from the fact that the new graph can be homomorphically embedded into G in a way that all paths are preserved and the fact that these queries are preserved under these types of homomorphisms.
– Furthermore, note that there must be a homomorphism ν from Q_T to G so that $(\nu(w_1), \dots, \nu(w_k)) = (a_1, \dots, a_k)$. Remove from G all triples except (1) those that are the images of a triple (x, y, z) in Q^1_T for ν; or (2) those that were added in the previous step. Again, by doing this we do not increase the number of tuples in $[\![Q^2]\!]_G$.

Note that the resulting graph G' is now (ν, Σ)-canonical for Q^1. It thus follows that Q^2 cannot be mapped onto G, since otherwise we would have that $(a_1, \dots, a_k)$ would belong to $[\![Q^2]\!]_G$. This finishes the proof of the claim.

We need also another claim.

Claim. There is an $\mathsf{EXPSPACE}$ algorithm that decides, given $\mathcal{ASP}$-SPARQL queries Q^1 and Q^2, whether $Q^1 \not\subseteq Q^2$.

Proof. The proof is based on the idea of coding canonical graphs by means of words, so that they can be represented using automata, that appeared first in [2].

Indeed, consider a set Σ of properties and a set N of nodes. We represent RDF graphs containing both triples using $\Sigma \cup N$ and semipaths that start and end with nodes in $\Sigma \cup N$, as words over $\Sigma \cup N \cup \{\$, \star\}$ in the following way:

Assume that $G = G_T \cup G_P$, where G_T is just a collection of triples and G_P a collection of semipaths. Then G_T is represented by a word of form

$$\$s_1p_1o_1\$s_2p_2o_2\$\cdots\$s_no_np_n\$,$$

where each s_i, p_i, o_i range over $\Sigma \cup N$, so that they contain the concatenation of each triple in G. Furthermore, G_P is the concatenation of a word of form

$$ua_1 \star \$ \star a_2 \star \$ \star a_3 \star \$ \cdots \$ \star a_m v$$

for each semipath in G_P that starts in u and ends in v. Here each a_j belongs to Σ and u, v belong to $\Sigma \cup N$.

Let now Q^1 and Q^2 be the inputs to the algorithm. Recall that we assume that Q^1 and Q^2 are of the form $Q_\ell = \mathsf{SELECT}\ \ w_1, \ldots, w_k\ \mathsf{WHERE}\ Q_T^\ell\ \mathsf{AND}\, Q_P^\ell$, where

$$Q_T^\ell = (x_1^\ell, y_1^\ell, z_1^\ell)\ \mathsf{AND}\ (x_2^\ell, y_2^\ell, z_2^\ell)\ \mathsf{AND} \cdots \mathsf{AND}\ (x_{n_\ell}^\ell, y_{n_\ell}^\ell, z_{n_\ell}^\ell)$$

and

$$Q_P^\ell = (u_1^\ell, e_1^\ell, v_1^\ell)\ \mathsf{AND}\ (u_2^\ell, e_2^\ell, v_2^\ell)\ \mathsf{AND} \cdots \mathsf{AND}\ (u_{m_\ell}^\ell, e_{m_\ell}, v_{m_\ell}),$$

where each $x_i^\ell, y_i^\ell, z_i^\ell, u_j^\ell$ and v_j^ℓ, $1 \leq i \leq n_\ell$ and $1 \leq i \leq m_\ell$ and $1 \leq \ell \leq 2$ are variables or IRIs and each e_j^ℓ is a property path.

The idea of the proof is as follows:

– first we show how to construct an automaton A_1 that represent all graphs that are (ν, Σ)-canonical for Q^1, for some mapping ν;
– next we show how to construct an automaton A_2 that represent all graphs G so that Q_2 cannot be (ν, Σ)-mapped into G;
– we then check wether the intersection of A_1 and A_2 is nonempty.

We start with the construction of A_1. In particular, A_1 is constructed in a straightforward way, just as it is done in [2]. We take Σ as the set of all properties appearing in property paths in Q^1 or Q^2, plus an extra symbol $\bot$, used to represent fresh values in semipaths. Furthermore, let $\mathsf{var}(Q^1, Q^2)$ be the set of variables appearing in Q^1 or Q^2. We take N to be $\mathsf{var}(Q^1, Q^2) \times \Sigma' \cup \{\cdot\}$, Where Σ' is the set of all properties appearing in property paths in Q^1 or Q^2, i.e., $\Sigma \setminus \{\bot\}$. This represent the encoding of ν: elements $(x, \cdot)$ represent that the variable x is mapped to some element different from an element in Σ, and (x, a), for $a \in \Sigma'$ represents that x is mapped to a. Note that, unlike the proof of [2], we encode an equality $\nu(x) = \nu(y)$ in ν explicitly, by means of the empty word, or as a triple (x, ϵ, y) (hence we also assume that Σ contains ϵ). Having defined Σ and N, an automaton A_1 that accepts all words W that represent graphs that are (ν, Σ) canonical for Q^1, for some mapping ν, is straightforward to define as the intersection of the following two automata.

- One automaton that first checks that for each triple (x, y, z) in Q^1_T, the triple $(\nu(x), \nu(y), \nu(z))$ is in W, and then checks that for each triple (u, p, v) in Q^1_P, there is a sequence $(\nu(u), a_1, \star)\$ \cdots (\star, a_m, \nu(v))$ representing a semipath that canonically conforms to G with respect to Σ' (recall that symbols of form $!(a_1, \ldots, a_n)$ can be witnessed by any symbol in Σ different from $a_1, \ldots, a_n$).
- The second automaton checking that the mapping ν is correct: first, for every variable $x \in V$, all symbols in N containing x are the same, and second, there are no two variables x and y such that the word contains a triple $\$x\epsilon y\$$ and elements (x, a), (y, b) for $a, b \in \Sigma$ and $a \neq b$.

Note that this automaton is exponential in the size of Q^1.

Having the automaton A_1 constructed, next we describe the construction of the automaton A_2.

This idea is to build an automaton whose language corresponds to all those graphs G such that Q^2 can be (ν, Σ)-mapped to G, and then complement it. Once again, we are interested in RDF graphs built using nodes from N, and take Σ just as in the construction of A_1. The language of A_2 will thus be those graphs over N such that Q^2 can be (ν, Σ)-mapped to G. Here we assume that ν the mapping that sends each variable x to the node (x, a) if $\nu(x)$ belongs to Σ', or to $(x, \cdot)$ if $\nu(x)$ does not belong to Σ. This time we also need to *guess* a mapping γ from the variables of Q^ℓ to the domain of G such that

- $\gamma(x) = \nu(x)$ for each free variable in Q^2 (and thus in Q^1),
- for each triple (x, y, z) in Q^2_T we have that $(\gamma(x), \gamma(y), \gamma(z)$ is in G,
- for each triple (u, r, v) of Q^2_P, we have that $(\gamma(u), \gamma(v)) \in [\![r]\!]_G$.

In order to do this, we expand the alphabet of A_2 to $N \times \mathsf{var}(Q_1, Q_2)$, with the second component representing the assignment γ, and then we need to construct automata that checks the following.

1. That $\gamma(x) = \nu(x)$ for each free variable in Q^2 (and thus in Q^1). This amounts to check that for every symbol of form (p, x), with p of from (z, a) or $(z, \cdot)$ $(z \neq x)$, there must exist a path of symbols of form $(x \epsilon u_1), \ldots, (u_k \epsilon z)$, not necessarily in that order. This can clearly be done with a two way automata of polynomial size in the size Q^1 and Q^2.
2. An automata that checks that the assignment is correct, i.e. that no symbols of form $((x, a), z)$ and $((x, a), w)$ exists, for $w \neq z$, and that the presence of a symbol of form $((x, a), z)$ forces every node (x, a) in N to be joined with a z. This can also be done with an automata of polinomial size with respect to Q^1 and Q^2.

We can then finally define the automata A_2. It needs to check that the mapping γ correctly represent a (ν, Σ)-mapped of Q^2 to G. This can be done by adapting the techniques in [2] in the expected way. We can then take the product of the complement of A_2 with the other automata defined previously and obtain an automata whose language corresponds to all those graphs G such that Q^2 cannot be (ν, Σ)-mapped to G.

We have all the ingredient to show the proof Theorem 1. It is based on the proof of the claim above. Indeed, taking the union of queries just amounts to taking the union of the automata used to represent these queries. Note as well that the proof remains to hold if we have $\mathcal{AUSP}$-SPARQL queries of length exponential in the size of the maximum conjunct, since A_1 and A_2 only depend on the size of the maximum conjunct and not on the number of queries in the union.

Also, if Q^2 does not have any projection, then the intersection can be done in PSPACE. The lower bounds follows from the results of [2].

Proposition 3. *The problem* SUBSUMPTION($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* $\mathsf{EXPSPACE}$*-complete for* $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{AP}, \ldots, \mathcal{AUSP}\}$.

Proof. The upper bound follows from the observation that subsumption is not more difficult than containment noted in the body of the paper.

For the lower bound, it was shown in [2] that checking containment of C2RPQs is $\mathsf{EXPSPACE}$-hard, even for queries $Q_1(x_1, x_2)$ and $Q_2(x_1, x_2)$ of the form

$$\begin{aligned}
&Q_1(x_1, x_2) = E(x_1, x_2),\\
&Q_2(x_1, x_2) = E_1(x_1, y_1), F_1(y_1, y_2), \ldots, F_n(y_1, y_2), E_1(y_2, x_2).
\end{aligned}$$

for $n \geq 1$ and regular path queries $E, E_1, F_1, \ldots, F_n$ over an alphabet Σ. We can see each RPQ as a property path and define the queries

$$\begin{aligned}
Q_1' &= \mathsf{SELECT}\ ?x_1, ?x_2\ \mathsf{WHERE}\ (?x_1, E, ?x_2),\\
Q_2' &= \mathsf{SELECT}\ ?x_1, ?x_2\ \mathsf{WHERE}\ (?x_1, E_1, ?y_1)\ \mathsf{AND}\ (?y_1, F_1, ?y_2)\ \mathsf{AND} \ldots\\
&\qquad \mathsf{AND}(?y_1, F_n, ?y_2)\ \mathsf{AND}\ (?y_2, E_1, ?x_2).
\end{aligned}$$

It is straighforward to show that $Q_1(x_1, x_2) \subseteq Q_2(x_1, x_2)$ if and only if $Q_1' \sqsubseteq Q_2'$. Note that this shows that subsumption is $\mathsf{EXPSPACE}$-hard even when we do not use the OPT operator, that is, for queries that only use the AND operator and property paths.

Proposition 4. *The following holds:*
– EVALUATION($\mathcal{X}$-SPARQL) *is* Σ_2^p*-complete for* $\mathcal{X} \in \{\mathcal{AOSP}, \mathcal{AUOSP}\}$*;*
– EVALUATION($\mathcal{X}$-SPARQL) *is* coNP*-complete for* $\mathcal{X} \in \{\mathcal{AOP}, \mathcal{AUOP}\}$.

Proof. We start with the second item of the proposition, and prove the upper bound it on the base of the following claim, which is very similar to the one in [9] (the lower bound follows from [14]).

Claim. For any $\mathcal{AOP}$-SPARQL pattern P in NR normal form and an RDF graph G a mapping μ is a solution for P over G if and only if there is subtree T' of *Tree*(P) such that

1. $\mathsf{dom}(\mu) = \mathsf{var}(T')$ and
2. T' is the maximal subtree of *Tree*(P) such that there is a mapping $\nu \in [\![\mathsf{and}(T')]\!]_G$, with $\mathsf{dom}(\mu) \subseteq \mathsf{dom}(\nu)$ and $\mu \sim \nu$.

Since this clai shows that the same characterisation of solutions can be used for patterns with property path patterns as in the case when they are not used, the algorithm from [9] gives us the desired result (with the remark that each property path pattern can be evaluated in polynomial time and that the use of union is only at the top level).

For the first item, The lower bound follows form [9, Theorem 6.6]. For the upper bound, if the query is UNION-free then using the intuition described in the paper (similar to the one in [9]) we can devise a simple guess and check algorithm that works in Σ_2^p. That is, given an $\mathcal{AOSP}$-SPARQL query $Q = \mathsf{SELECT}\ X\ \mathsf{WHERE} P$, an RDF graph G, and a mapping μ we simply guess a subtree T of *Tree*(P) and a mapping $\lambda : \mathsf{var}(T) \setminus X \to G$ and then check that the mapping $\mu \cup \lambda$ belongs to $[\![P]\!]_G$. This can be done in coNP and we get the desired result. In case UNION is used we simply guess the part of the union and do the check for it. Note that here we use the fact that union is always at the top level and that SELECT distributes over it.

Lemma 1. *Let*

$$Q_1 = \mathsf{SELECT}\, X\ \mathsf{WHERE}\ P \quad \textit{and} \quad Q_2 = P^1\ \ \mathsf{UNION} \ldots \mathsf{UNION}\ \ P^k$$

be a $\mathcal{AOUS}$-SPARQL and $\mathcal{AOU}$-SPARQL queries correspondingly. Then $Q_1 \not\subseteq Q_2$ if and only if there is a good extension E over Q_2 of some $\mathcal{AOSP}$-SPARQL query with a pattern P^* such that Tree(P^*) is a subtree of one of the trees representing components of P and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ that satisfies the following conditions:

(C1) *for each child n of Tree(P^*), there is no homomorphism h from $\mathsf{and}(n)$ to E such that $h(?x) = ?x$ for all variables $?x$ in $\mathsf{var}(n) \cap \mathsf{var}(E)$, and*

(C2) *for each subtree T of one of Tree$(P^1), \ldots,$ Tree(P^k) with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$ there is no homomorphism h from $\mathsf{and}(T)$ to E such that $h(?x) = ?x$, for all variables $?x$ in $\mathsf{var}(T) \cap \mathsf{var}(E)$.*

Proof. We start with the backward direction.
($\Leftarrow$) We need to show that $Q_1 \not\subseteq Q_2$. Let G be the RDF graph given by the "frozen database" associated with E, that is, each variable $?x$ in E is replaced by an IRI a_x. Let μ be the mapping such that $\mathsf{dom}(\mu) = X^*$ and $\mu(?x) = a_x$, for each variable $?x \in X$. We claim that $\mu \in [\![Q_1]\!]_G$ and $\mu \notin [\![Q_2]\!]_G$, which implies $Q_1 \not\subseteq Q_2$. Indeed, consider the mapping λ with $\mathsf{dom}(\lambda) = X^*$ such that $\lambda(?x) = a_x$, for each variables $?x \in X^*$. By construction of G, we have

that $\lambda \in [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. Let T_1 be the tree representing the component of P that contains $\mathit{Tree}(P^*)$ as a subtree. Condition (1) implies that $\mathit{Tree}(P^*)$ is a maximal subtree in T_1 such that $\{\lambda\} \sqsubseteq [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. It follows that $\lambda \in [\![T_1]\!]_G$ (assuming that the semantics of a pattern tree coincides with the semantics of the pattern it represents). In particular, $\lambda|_X \in [\![Q^*]\!]_G$, where Q^* is the query with the pattern corresponding to T_1 and distinguished variables X. Since $\lambda|_X = \mu$, we have that $\mu \in [\![Q^*]\!]_G \subseteq [\![Q_1]\!]_G$.

Now we show that $\mu \notin [\![Q_2]\!]_G$. It suffices to show that $\mu \notin [\![P^i]\!]_G$, for all P^i in Q_2. If $\mathit{Tree}(P^i)$ does not contain any subtree of one of $\mathit{Tree}(P^1), \dots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$, then $\mu \notin [\![P^i]\!]_G$ holds trivially. Suppose $\mathit{Tree}(P^i)$ contains a subtree T' of one of $\mathit{Tree}(P^1), \dots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$. Assume first that T' is not in the support of E. Condition (2) implies that $\mu \notin [\![P^i]\!]_G$, as μ is not even in $[\![\mathsf{and}(T')]\!]_G$. Assume now that T' is in the support of E. Then, there is a child n of T' and a renaming ρ of the variables in n with $\rho(?x) = ?x$, for all $?x \in \mathsf{var}(T')$, such that all the renamed triples in n belong to E. In particular, all these renamed triples are in G, after freezing each variable. It follows that if $\mu \in [\![\mathsf{and}(T')]\!]_G$, then $\mu' \in [\![\mathsf{and}(T'')]\!]_G$, where T'' is the subtree resulting from the union of T' and the node n, and μ' is the extension of μ to n such that $\mu'(?x) = ?y$, for all $?x \in \mathsf{var}(n) \setminus \mathsf{var}(T')$ and $?y = \rho(?x)$. Then, T' cannot be a maximal tree of P^i with the property $\{\mu\} \sqsubseteq [\![T']\!]_G$. We conclude that $\mu \notin [\![P^i]\!]_G$. This shows that $\mu \notin [\![Q_2]\!]_G$ as required.

($\Rightarrow$) Suppose $Q_1 \not\subseteq Q_2$. Then there is an RDF graph and a mapping μ such that $\mu \in [\![Q_1]\!]_G$ and $\mu \notin [\![Q_2]\!]_G$. Since $\mu \in [\![Q_1]\!]_G$, there is a component P_1' in P such that $\mu \in [\![Q_1']\!]_G$, where Q_1' is a query with pattern P_1' and distinguished variables X. By definition, there is a pattern P^* with $\mathit{Tree}(P^*)$ being a subtree of $\mathit{Tree}(P_1')$ and a mapping λ such that $\mathsf{dom}(\lambda) = \mathsf{var}(\mathit{Tree}(P^*))$, $\lambda|_X = \mu$ and $\mathit{Tree}(P^*)$ is a maximal subtree in $\mathit{Tree}(P_1')$ with the property $\{\lambda\} \sqsubseteq [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. We construct a good extension E of the $\mathcal{AOSP}$-SPARQL query with a pattern P^* and distinguished variables X^* over Q_2 as follows. Let P^i be a pattern in Q_2 such that $\mathit{Tree}(P^i)$ contains a subtree T_2' with $\mathsf{var}(T_2') \cap X = X^*$. The subtree T_2' belongs to the support of E if and only if $\mu \in [\![\mathsf{and}(T_2')]\!]_G$. In this case, since $\mu \notin [\![P^i]\!]_G$, there exists a child n of T_2', and a mapping μ_n such that $\mathsf{dom}(\mu_n) = \mathsf{var}(n)$, μ and μ_n are compatible, and $\mu \cup \mu_n \in [\![\mathsf{and}(T_2'')]\!]_G$, where T_2'' is the union of T_2' and n. We pick this child n to be added to the expansion E. Thus we rename all the variables in $\mathsf{var}(n) \setminus \mathsf{var}(T_2')$ by fresh variables via a renaming ρ_n. Let this set of renamed triples be $\rho_n(n)$. Then we add each triple in $\rho_n(n)$ to E. Observe that there is a homomorphism h from E to G such that $h(?x) = \mu(?x)$, for all variables $?x \in X^*$. Indeed, the mapping λ is an homomorphism from $\mathsf{and}(\mathit{Tree}(P^*))$ to G. For each child n considered in the construction of E, the mapping $\mu_n \circ \rho_n^{-1}$ is an homomorphism from $\mathsf{and}(\rho_n(n))$ to G. Moreover, the mappings λ and $\mu_n \circ \rho_n^{-1}$ are compatible: the only common variables between $\mathsf{and}(\mathit{Tree}(P^*))$ and $\mathsf{and}(\rho_n(n))$, or between $\mathsf{and}(\rho_{n_1}(n_1))$ and $\mathsf{and}(\rho_{n_2}(n_2))$, for distinct children n_1, n_2, are variables in X^*. Since the image over a variable $?x \in X^*$ is always $\mu(?x)$ for all of these mappings then they are actually compatible. Thus we can define h to be the union of λ and the mappings $\mu_n \circ \rho_n^{-1}$'s. Observe that it is crucial to rename the variables in n, otherwise it is not possible to construct h (this is the reason of the renaming in the definition of good extensions). We claim that E satisfies conditions (1) and (2) in the lemma. For condition (1), assume for the sake of contradiction that there is a child n of $\mathit{Tree}(P^*)$ and a mapping ν from $\mathsf{and}(n)$ to E with $\nu(?x) =?x$, for all $?x \in \mathsf{var}(n) \cap \mathsf{var}(E)$. Consider the mapping ν_{id} from $\mathsf{var}(\mathit{Tree}(P^*))$ to E such that $\nu_{id}(?x) =?x$, for all $?x \in \mathsf{var}(\mathit{Tree}(P^*))$. It follows that ν and ν_{id} are compatible. Consider now the subtree T_1'' that is the union of $\mathit{Tree}(P^*)$ with n, and the mapping $h' = h \circ (\nu_{id} \cup \nu)$ defined over $\mathsf{var}(T_1'')$. Note that the mapping h' is an homomorphism. Moreover, for all $?x \in \mathsf{var}(\mathit{Tree}(P^*))$, $h'(?x) = h \circ (\nu_{id} \cup \nu)(?x) = h(?x) = \lambda(?x)$. Thus h' actually extends λ which contradict the fact that $\mathit{Tree}(P^*)$ is a maximal subtree in $\mathit{Tree}(P_1')$ such that $\{\lambda\} \sqsubseteq [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. For condition (2), let T be a subtree for each subtree T of one of $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$. Note that, by construction of E, $\mu \notin [\![\mathsf{and}(T)]\!]_G$. For the sake of contradiction, assume that there is a homomorphism η from $\mathsf{and}(T)$ to E such that $\eta(?x) =?x$, for all $?x \in \mathsf{var}(T) \cap \mathsf{var}(E)$. We can define the homomorphism $\eta' = h \circ \eta$. Observe that $\mathsf{var}(T) \subseteq \mathsf{var}(E)$, thus η is simply the identity. It follows that for each variable $?x \in \mathsf{var}(T)$, $\eta'(?x) = h \circ \eta(?x) = h(?x) = \mu(?x)$. Thus $\eta' = \mu$. This implies that $\mu \in [\![\mathsf{and}(T)]\!]_G$, which is a contradiction.

Lemma 2. *Let*

$$Q_1 = \mathsf{SELECT}\, X\ \mathsf{WHERE}\ P \quad \textit{and} \quad Q_2 = P^1\ \mathsf{UNION} \ldots \mathsf{UNION}\ P^k$$

be a $\mathcal{AOUSP}$-SPARQL and $\mathcal{AOUP}$-SPARQL queries correspondingly. Then $Q_1 \not\subseteq Q_2$ if and only if there is a good extension E over Q_2 of some $\mathcal{AOSP}$-SPARQL query with a pattern P^* such that $\mathit{Tree}(P^*)$ is a subtree of one of the trees representing components of P and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ that satisfies $E \not\preceq N$ UNION S, where

(C1') *N is a union of all $\mathsf{and}(n)$ for n a child of $\mathit{Tree}(P^*)$, and*

(C2') *S is a union of all $\mathsf{and}(T)$ for T a subtree of one of $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$.*

Proof. We start with the backward direction.
($\Leftarrow$) We need to show that $Q_1 \not\subseteq Q_2$. Since $E \not\preceq N$ UNION S, there is an RDF graph G and a mapping μ such that (i) $\mu \in [\![E]\!]_G$, (ii) there is no mapping in $[\![S]\!]_G$, that is in the union of all $[\![\mathsf{and}(n)]\!]_G$ for n a child of $\mathit{Tree}(P^*)$, compatible with μ, and (iii) there is no mapping in $[\![N]\!]_G$, that is in the union of all $[\![\mathsf{and}(T)]\!]_G$ for T a subtree of one of $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$, compatible with μ. Let ξ be the mapping with $\mathsf{dom}(\xi) = X^*$ such that $\xi(?x) = \mu(?x)$, for all $?x \in X^*$. We claim that $\xi \in [\![Q_1]\!]_G$ and $\xi \notin [\![Q_2]\!]_G$, which implies $Q_1 \not\subseteq Q_2$. We show first that $\xi \in [\![Q_1]\!]_G$. Since $\mu \in [\![E]\!]_G$ and all the triple and property path patterns in $\mathit{Tree}(P^*)$ are contained in E, it follows that $\mu' = \mu|_{\mathsf{var}(P^*)} \in [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. Let T_1 be the tree of one of the components

of P that contains $\mathit{Tree}(P^*)$ as a subtree. It suffices to show that $\mathit{Tree}(P^*)$ is a maximal subtree in T_1 with the property $\{\mu'\} \sqsubseteq [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. Indeed, this would imply that $\mu' \in [\![T_1]\!]_G$, hence $\xi = \mu'|_X \in [\![Q_1]\!]_G$. For contradiction suppose that $\mathit{Tree}(P^*)$ is not maximal. Then there is a child n of $\mathit{Tree}(P^*)$ and a mapping $\nu \in [\![\mathsf{and}(n)]\!]_G$ such that ν is compatible with μ'. In particular, ν is compatible with μ, which contradicts (ii). Now we show that $\xi \notin [\![Q_2]\!]_G$. Consider any $\mathit{Tree}(P^i)$ representing a component of Q_2. Assume by contradiction that $\xi \in [\![\mathit{Tree}(P^i)]\!]_G$. It must be the case that there is a subtree T_2' in $\mathit{Tree}(P^i)$ with $\mathsf{var}(T_2') = \mathsf{dom}(\xi) = X^*$. Suppose first that $T_2' \notin \mathsf{sup}(E)$. Since $\xi \in [\![\mathit{Tree}(P^i)]\!]_G$, it follows that $\xi \in [\![\mathsf{and}(T_2')]\!]_G$. By definition of ξ, we have that ξ and μ are compatible, which contradicts (iii). Now suppose $T_2' \in \mathsf{sup}(E)$. By construction of E, there is a child n of T_2' and a renaming ρ of $\mathsf{var}(n)$ that is the identity over $\mathsf{var}(n) \cap \mathsf{var}(T_2')$, such that all the triples resulting from applying ρ to n are in E. Let ν be the mapping over $\mathsf{var}(n)$ defined by $\mu \circ \rho$. Note that $\nu \in [\![\mathsf{and}(n)]\!]_G$. Moreover ν and ξ are compatible. Indeed, if $?x \in \mathsf{var}(T_2') \cap \mathsf{var}(n)$, then $\nu(?x) = \mu(?x) = \xi(?x)$. Thus $\xi \cup \nu \in [\![\mathsf{and}(T_2'')]\!]_G$, where T_2'' is the union of T_2' and n. This is a contradiction with the fact that $\xi \in [\![\mathit{Tree}(P^i)]\!]_G$. We conclude that $\xi \notin [\![Q_2]\!]_G$.

($\Rightarrow$) Suppose $Q_1 \not\sqsubseteq Q_2$. Then there is an RDF graph and a mapping μ such that $\mu \in [\![Q_1]\!]_G$ and $\mu \notin [\![Q_2]\!]_G$. Since $\mu \in [\![Q_1]\!]_G$, there is a tree T_1 representing one of the components of P such that μ is in the semantics of this component projected to X over G. By definition, there is a pattern P^* such that $\mathit{Tree}(P^*)$ is a subtree of T_1 and a mapping λ such that $\mathsf{dom}(\lambda) = \mathsf{var}(\mathit{Tree}(P^*))$, $\lambda|_X = \mu$ and $\mathit{Tree}(P^*)$ is a maximal subtree in T_1 with the property $\{\lambda\} \sqsubseteq [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. We construct a good extension E of a query with pattern P^* and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ over Q_2 as follows. Let P^i be a pattern in Q_2 such that $\mathit{Tree}(P^i)$ contains a subtree T_2' with $\mathsf{var}(T_2') = X^*$. The subtree T_2' belongs to the support of E if and only if $\mu \in [\![\mathsf{and}(T_2')]\!]_G$. In this case, since $\mu \notin [\![\mathit{Tree}(P^i)]\!]_G$, there exists a child n of T_2', and a mapping μ_n such that $\mathsf{dom}(\mu_n) = \mathsf{var}(n)$, μ and μ_n are compatible, and $\mu \cup \mu_n \in [\![\mathsf{and}(T_2'')]\!]_G$, where T_2'' is the union of T_2' and n. We pick this child n to be added to the expansion E. Thus we rename all the variables in $\mathsf{var}(n) \setminus \mathsf{var}(T_2')$ by fresh variables via a renaming ρ_n. Let this set of renamed triples be $\rho_n(n)$. Then we add each triple in $\rho_n(n)$ to E. Observe that there is a mapping h from $\mathsf{var}(E)$ to $\mathsf{dom}(G)$ such that $h \in [\![E]\!]_G$ and $h(?x) = \mu(?x)$, for all variables $?x \in X^*$. Indeed, we know that $\lambda \in [\![\mathsf{and}(\mathit{Tree}(P^*))]\!]_G$. On the other hand, for each child n considered in the construction of E, the mapping $\mu_n \circ \rho_n^{-1}$ belongs to $[\![\mathsf{and}(\rho_n(n))]\!]_G$. Moreover, the mappings λ and $\mu_n \circ \rho_n^{-1}$'s are compatible: the only common variables between $\mathsf{and}(\mathit{Tree}(P^*))$ and $\mathsf{and}(\rho_n(n))$, or between $\mathsf{and}(\rho_{n_1}(n_1))$ and $\mathsf{and}(\rho_{n_2}(n_2))$, for distinct children n_1, n_2, are variables in X^*. Since the image over a variable $?x \in X^*$ is always $\mu(?x)$ for all of these mappings then they are actually compatible. Thus we can define h to be the union of λ and the mappings $\mu_n \circ \rho_n^{-1}$'s. Observe that it is crucial to rename the variables in n, otherwise it is not possible to construct h (this is the reason of step (3) in the definition of good extensions). We claim that $E \not\preceq N \cup S$, where N is a union of all $\mathsf{and}(n)$ for n a child of $\mathit{Tree}(P^*)$, and S is a union of all $\mathsf{and}(T)$ for T a subtree of one of $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$. Indeed, for the sake of contradiction assume that $E \preceq N \cup S$. Since $h \in [\![E]\!]_G$, we have two cases.

- There is a mapping $\nu \in [\![\mathsf{and}(n)]\!]_G$, for some child n of $\mathit{Tree}(P^*)$, that is compatible with h. This implies that ν and λ are compatible. It follows that $\lambda \cup \nu \in [\![\mathsf{and}(T_1'')]\!]_G$, where T_1'' is the union of $\mathit{Tree}(P^*)$ and n. This contradicts the maximality of $\mathit{Tree}(P^*)$.
- There is a mapping $\nu \in [\![\mathsf{and}(T_2')]\!]_G$, for some T_2' among $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$, that is compatible with h. Since $\mathsf{dom}(\nu) = X^* \subseteq \mathsf{dom}(h)$, it follows that $\nu = h|_{X^*} = \mu$. Thus $\mu \in [\![\mathsf{and}(T_2')]\!]_G$ which is a contradiction with the definition of the support of E.

We conclude that $E \not\preceq N \cup S$. This finishes the proof of the lemma.

Theorem 2. *The problem* CONTAINMENT($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* EXPSPACE-*complete for* $\mathcal{X}_1 \in \{\mathcal{AOP}, \ldots, \mathcal{AOUSP}\}$ *and* $\mathcal{X}_2 \in \{\mathcal{AOP}, \mathcal{AOUP}\}$.

Proof. The exponential space algorithm proceed as follows. Given an $\mathcal{AOUSP}$-SPARQL query

$$Q_1 = \mathsf{SELECT}\ \ X\ \mathsf{WHERE} P$$

and $\mathcal{AOUP}$-SPARQL query

$$Q_2 = P^1\ \mathsf{AND} \ldots \mathsf{AND}\ P^k,$$

we iterate over all patterns P^* such that $\mathit{Tree}(P^*)$ is a subtree of one of the trees representing components of P and good extensions of queries with patterns P^* and distinguished variables $X^* = X \cap \mathsf{var}(P^*)$ over Q_2, until we find a good extension E such that $E \not\preceq N$ UNION S, where N is a union of all $\mathsf{and}(n)$ for n a child of $\mathit{Tree}(P^*)$, and S is a union of all $\mathsf{and}(T)$ for T a subtree of one of $\mathit{Tree}(P^1), \ldots, \mathit{Tree}(P^k)$ with $\mathsf{var}(T) = X^*$ that is not in $\mathsf{sup}(E)$. If this E is found, we reject; otherwise we accept. Lemma 2 tell us that this algorithm is correct. Observe that the sizes of E, N, and S are polynomial in the size of Q_1 and Q_2. Moreover, checking non-compatibility for $\mathcal{ASP}$-SPARQL queries can be done in EXPSPACE. Indeed, we can use the same proof of Proposition 1. In this proof, when checking whether $Q_1 \subseteq Q_2^1 \cup \cdots \cup Q_2^k$, we construct an automaton A_2^{map}, for each each disjunct Q_2^i, that accepts all the canonical graphs G of Q_1 such that Q_2^i can be mapped to G. To build A_2^{map}, we guess the mapping and check that it is correct. To guess the mapping from Q_2^i to G, we use "annotations" in G, in other words, we expand our alphabet and explicitly indicate the images of each variable in Q_2^i. We also have to impose that this guessed mapping sends free variables of Q_2^i to "distinguished" variables in G (that is, the $\nu(\bar{w})$ tuple). For compatibility the same construction applies. The only difference is that when we guess our mapping we have to impose that it is the identity over the "common" variables between Q_2^i and G. In other words, if $x \in \mathsf{var}(Q_2^i) \cap \mathsf{var}(Q_1)$, then it must be that case that our mapping send x to $\nu(x)$ in the canonical graph G. All the other constructions in the proof apply directly to the case of compatibility. It follows that $E \not\preceq N \cup S$ can be checked in exponential space. This shows membership in $\mathsf{EXPSPACE}$.

For the $\mathsf{EXPSPACE}$-hardness, we know from Proposition 3 that subsumption of $\mathcal{AP}$-SPARQL patterns is $\mathsf{EXPSPACE}$-hard. We reduce this problem to our problem. Let Q_1 and Q_2 be two $\mathcal{AP}$-SPARQL patterns. Consider the queries $Q_1' = Q_1 \mathbin{\mathsf{OPT}} Q_2$ and $Q_2' = Q_1 \mathbin{\mathsf{AND}} Q_2$. Note that Q_1' and Q_2' are in $\mathcal{AOP}$-SPARQL. Moreover, it is easy to see that $Q_1 \sqsubseteq Q_2$ if and only if $Q_1' \subseteq Q_2'$. This shows the last part of the theorem.

Theorem 3. *The problem* SUBSUMPTION($\mathcal{X}_1$-SPARQL,$\mathcal{X}_2$-SPARQL) *is* $\mathsf{EXPSPACE}$*-complete for* $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{AOP}, \ldots, \mathcal{AOUSP}\}$.

Proof. We prove this theorem by means of the following claim.

Claim. Let

$$Q_i = \mathsf{SELECT}\ \ X_i\ \mathsf{WHERE} P_i,$$

for $i = 1, 2$, be two $\mathcal{AOUSP}$-SPARQL queries. Then $Q_1 \sqsubseteq Q_2$ if and only if for every subtree T_1' of a tree of a component of P_1 we have that the query Q^1 with the pattern $\mathsf{and}(T_1')$ and distinguished variables $X^* = X_1 \cap \mathsf{var}(T_1')$ is contained in the union of all queries with patterns $\mathsf{and}(T_2')$ and distinguished variables X^*, where T_2' is a subtree of a tree representing a component of P_2 such that $X^* \subseteq X_2 \cap \mathsf{var}(T_2')$.

Proof. For the "only-if" part, let T_1' be a subtree of a component of P_1, let G be an RDF graph and $\mu \in [\![Q^1]\!]_G$. We need to show that $\mu \in [\![Q^2]\!]_G$, for Q^2 with pattern $\mathsf{and}(T_2')$ and distinguished variables X^*, where T_2' is a subtree of a tree representing a component of P_2 such that $X^* \subseteq X_2 \cap \mathsf{var}(T_2')$. We know that T_1' is a subtree of a component of P_1. Let P be this component. Because $\mu \in [\![Q^1]\!]_G$, there is a mapping $\lambda \in [\![\mathsf{and}(T_1')]\!]_G$ such that $\mu = \lambda|_{X^*}$. Let T_1'' be a maximal subtree of $\mathit{Tree}(P)$ such that $\lambda \sqsubseteq [\![\mathsf{and}(T_1'')]\!]_G$. Let λ' be a mapping in $[\![\mathsf{and}(T_1'')]\!]_G$ such that $\lambda \sqsubseteq \lambda'$. By definition, $\lambda' \in [\![\mathit{Tree}(P)]\!]_G$, which implies that μ' is in the semantics of the query with pattern P and distinguished variables X_1 over G, where $\mu' = \lambda'|_{X_1}$ and $\mu \sqsubseteq \mu'$. In particular, $\mu' \in [\![Q_1]\!]_G$, thus by hypothesis, there is a mapping $\nu' \in [\![Q_2]\!]_G$ such that $\mu' \sqsubseteq \nu'$. We have that there is a tree T_2 representing one of the components of P_2 such that ν' in the semantics of the query with pattern represented by T_2 and distinguished variables X_2 over G. In particular, there is a subtree T_2' of T_2 such that ν' is in the semantics of the query with pattern represented by T_2' and distinguished variables $X_2 \cap \mathsf{var}(T_2')$ over G. Since $\mu \sqsubseteq \nu'$ and $X^* = \mathsf{dom}(\mu)$, we have that $X^* \subseteq X_2 \cap \mathsf{var}(T_2')$, and $\mu \in [\![Q^2]\!]_G$, as required.

For the "if" part, let G be an RDF graph and $\mu \in [\![Q_1]\!]_G$. We need to find ν with $\nu \in [\![Q_2]\!]_G$ and $\mu \sqsubseteq \nu$. Since $\mu \in [\![Q_1]\!]_G$, there is a subtree T_1' of a component of Q_1 such that $\mu \in [\![Q^1]\!]_G$, where the query Q^1 has the pattern $\mathsf{and}(T_1')$ and distinguished variables $X^* = X_1 \cap \mathsf{var}(T_1')$. By hypothesis, there is a subtree T_2' of a tree representing a component of P_2 such that $X^* \subseteq X_2 \cap \mathsf{var}(T_2')$. Let P^2 be the pattern in P_2 such that $\mathit{Tree}(P^2)$ has T_2' as a subtree and let λ be some mapping with $\lambda \in [\![\mathsf{and}(T_2')]\!]_G$ and $\mu = \lambda|_{X^*}$. Let T_2'' be a maximal subtree of $\mathit{Tree}(P^2)$ such that $\lambda \sqsubseteq [\![\mathsf{and}(T_2'')]\!]_G$, and let λ' be some mapping in $[\![\mathsf{and}(T_2'')]\!]_G$ such that $\lambda \sqsubseteq \lambda'$. By definition, $\lambda' \in [\![\mathit{Tree}(P_2)]\!]_G$ and $\nu \in [\![Q^2]\!]_G$, where Q^2 is a query with pattern P^2 and distinguished variables X_2, and $\nu = \lambda'|_{X_2}$. In particular, $\nu \in [\![Q_2]\!]_G$. Finally, note that $\mu \sqsubseteq \nu$, because $X^* \subseteq \mathsf{var}(T_2') \cap X_2 \subseteq \mathsf{var}(T_2'') \cap X_2$. This ends the proof of the claim.

The exponential space algorithm proceeds as follows. Given two $\mathcal{AOUSP}$-SPARQL queries

$$Q_i = \mathsf{SELECT}\ \ X_i\ \mathsf{WHERE} P_i,$$

for $i = 1, 2$, for each subtree T_1' of a tree representing a component of P_1 we first construct the query $Q_{T_1'}$ with pattern being the union of all $\mathsf{and}(T_2')$ for T_2' a subtree of a tree representing a component of P_2 such that $X_1 \cap \mathsf{var}(T_1') \subseteq X_2 \cap \mathsf{var}(T_2')$ and distinguished variables $X_1 \cap \mathsf{var}(T_1')$, and then check that the query with pattern $\mathsf{and}(T_1')$ and distinguished variables $X_1 \cap \mathsf{var}(T_1')$ is contained in $Q_{T_1'}$. If this is true for each T_1' we accept; otherwise we reject. The claim above gives us the correctness of the algorithm. Moreover, the construction of each $Q_{T_1'}$ and the containment check can be done in exponential space. Indeed, although the number of disjuncts in $Q_{T_1'}$ could be exponential in the size of Q_1 and Q_2 we can still check for containment in exponential space, as long as each disjunct is of polynomial size.