Containment of Expressive SPARQL Navigational Queries

Melisachew Wudage Chekol[1(✉)] and Giuseppe Pirrò[2]

[1] Data and Web Science Group, University of Mannheim, Mannheim, Germany
mel@informatik.uni-mannheim.de
[2] Institute for High Performance Computing and Networking, ICAR-CNR, Rende, Italy
pirro@icar.cnr.it

Abstract. Query containment is one of the building block of query optimization techniques. In the relational world, query containment is a well-studied problem. At the same time it is well-understood that relational queries are not enough to cope with graph-structured data, where one is interested in expressing queries that capture *navigation* in the graph. This paper contributes a study on the problem of query containment for an expressive class of navigational queries called Extended Property Paths (EPPs). EPPs are more expressive than previous navigational extension of SPARQL (e.g., nested regular expressions) as they allow to express path conjunction and path negation, among others. We attack the problem of EPPs containment and provide complexity bounds.

1 Introduction

Research in graph query languages has emerged as a consequence of the intrinsic limitations of relational query languages when it comes to the possibility to express recursion and navigation. This lead to the design of languages like Regular Path Queries (RPQs) and their extensions, including 2RPQs [2] that include the possibility to traverse paths backwards, Extended Regular Path Queries (ERPQs) [1] that offer higher expressive power by allowing to express queries that also capture graphs [11,13]. Another well-studied class of queries is that of Nested Regular Expressions (NREs) [26], that were originally proposed as the navigational core of SPARQL. The current SPARQL 1.1 standard introduced Property Paths (PPs) as navigational core; PPs offer very limited expressive power due to the lack of features to express any type of test within a path. To cope with these issues and design a language as close as possible to the current W3C standard, the language of Extended Property Paths (EPPs) has been recently proposed [12]. EPPs are more expressive than NREs, PPs and other navigational extensions of SPARQL thanks to the possibility to express path conjunction, negation and more powerful types of tests (e.g., checking the values of nodes reached via a nested expression) while keeping query evaluation tractable.

Related Work. The problem of query containment is one of the main pillars of static analysis and query optimization. In what follows we will not consider containment under constraints as we tackle containment of EPPs without constraints.

P. Groth et al. (Eds.): ISWC 2016, Part I, LNCS 9981, pp. 86–101, 2016.
DOI: 10.1007/978-3-319-46523-4_6

In the relational world, query containment for conjunctive queries (CQs) is now well-understood; it is NP-complete for basic CQs [5] and union of CQs [29] while it is Π_2^p when considering arithmetic comparison and also bag semantics [6,19].

For graph queries, containment of 2RPQs has been shown to be PSPACE-complete [1,2]; the complexity jumps to EXPTIME-hard under the presence of functionality constraints and to 2EXPTIME when considering expressive description logics constraints [4]. The problem becomes exponential if the query on the right hand side has a tree structure (cf. for example, [3]); for *extended* RPQs containment is undecidable for Boolean queries over a fixed alphabet [1].

In the Semantic Web, the containment of PSPARQL has been studied in Chekol et al. [8], this work provides lower bounds for upper bound complexity results reported in that paper; for NRE, Reutter [28] has shown a PSPACE upper bound; for SPARQL PPs, Kostylev et al. [20] show that the containment ranges from EXPSPACE-complete for OPT-free queries to PSPACE-complete if the right-hand side query is a pattern without projection. The study in [27] provides complexity analysis for several fragments of SPARQL: the results range from NP-completeness for AND-UNION queries to undecidability for the full SPARQL.

Another related language is XPath; the problem of XPath 2.0 query containment has been studied in ten Cate and Lutz [30]. They showed that the introduction of path intersection alone (i.e., the language CoreXPath ($\cap$) leads to 2EXPTIME-completeness. Finally, Kostylev et al. [22] studied the problem of containment of navigational XPath queries (i.e., the GXpath langauge), with results ranging from undecidability (when negation is considered) to EXP-TIME completeness for the positive fragment.

Contributions. We study query containment for EPPs, a significant extension of property paths (PPs), the current navigational core of SPARQL, and NREs for which containment has been already studied. We resort on two main ingredients: (i) an encoding of EPPs into the μ-calculus; (ii) the notion of (RDF) transition systems to check the validity test of μ-calculus formulae and provide an *upper bound* on the containment of EPPs and SPARQL with EPPs. For *lower bounds* we make connections between EPPs, PDL and XPath 2.0.

Automata theoretic notions and a reduction into validity test in a logic have been widely used to address the problem of query answering and containment [2,3,10,15,20,24]. Contrary to the automata techniques, the logic based approaches are fairly implementable. In this respect, it has been shown [16] that logical combinators can provide an exponential gain in succinctness in terms of the size of a logical formula thus allowing to study containment for expressive query languages in exponential-time, even though their direct formulation into the underlying logic results in an exponential blow up of the formula size.

Outline. The remainder of the paper is organized as follows. We provide some background in Sect. 2. The language of EPPs is introduced in Sect. 3. Section 4 describes an encoding of EPPs into the μ-calculus. Section 5 discusses the containment of EPPs. We conclude in Sect. 6.

2 Preliminaries

This section provides some background about the machineries used in this paper. We start with RDF and SPARQL and then give a brief overview about μ-calculus that will be used to tackle the containment of EPPs.

RDF and SPARQL. An RDF triple[1] is a tuple of the form $\langle s, p, o\rangle \in \mathbf{I}\times\mathbf{I}\times\mathbf{I}\cup\mathbf{L}$, where $\mathbf{I}$ (IRIs) and $\mathbf{L}$ (literals) are countably infinite sets. The set of terms of an RDF graph will be $terms(G)$ while $nodes(G)$ will be the set of terms used as a subject or object of a triple. An RDF graph G is a set of triples. To query RDF data, a standard query language, called SPARQL, has been defined. The semantics of a SPARQL query [25] is defined in terms of solution mappings. A (solution) mapping m is a partial function m: $\mathcal{V} \rightarrow \mathbf{I}\cup\mathbf{L}$. The SPARQL semantics uses a function $[\![Q]\!]_G$ that evaluates a query Q on a graph G and gives a multiset (bag) of mappings in the general case. However, when considering SPARQL with patterns using recursive PPs (i.e., using *, +) the standard introduces auxiliary functions (called ALP) that return sets of mapping.

μ-calculus. The μ-calculus ($\mathcal{L}_\mu$) is a logic obtained by adding fixpoint operators to ordinary modal logic [23]. For the purpose of this paper we will make usage of the μ-calculus with nominals and converse programs [31]. The syntax of the μ-calculus includes countable sets of *atomic propositions* AP, a set of *variables* Var, a set of *programs and their respective converses* $\text{Prog} = \{s, p, o, \bar{s}, \bar{p}, \bar{o}\}$ used to allow navigation in a graph. A μ-calculus formula φ is defined inductively as:

$$\varphi ::= \top \mid q \mid X \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle a\rangle\varphi \mid [a]\varphi \mid \mu X\varphi \mid \nu X\varphi$$

where $q \in AP, X \in Var$ and $a \in Prog$ is a transition program or its converse $\bar{a}$. The greatest fixpoint ν and least fixpoint operator μ introduce general and finite recursion in graphs, respectively. The semantics of the μ-calculus is given over a transition system (aka Kripke structure) $K = (S, R, L)$, where S is a non-empty set of nodes, $R : Prog \rightarrow 2^{S\times S}$ is the transition function, and $L : AP \rightarrow 2^S$ assigns a set of nodes to each atomic proposition or nominal where it holds, such that $L(p)$ is a *singleton* for each nominal p. For converse programs, R can be extended as $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$ where $s, s' \in S$.

Definition 1 *(Model of a formula). For a sentence φ and a transition system $K = (S, R, L)$, K is* model *of φ, denoted $K \models \varphi$, if there exists $s \in S$ such that $K, s \models \varphi$ if and only if $s \in [\![\varphi]\!]^K$ – s is an element of the answer to the evaluation of φ over K. If a sentence has a model, then it is called* satisfiable.

If a μ-calculus formula ψ appears under the scope of a least μ or greatest ν fixed point operator over all the programs $\{s, p, o, \bar{s}, \bar{p}, \bar{o}\}$ as, $\mu X.\psi \vee \langle s\rangle X \vee \langle p\rangle X \vee \cdots$ or $\nu X.\psi \wedge \langle s\rangle X \wedge \langle p\rangle X \wedge \cdots$, then, for legibility, we denote the formulae by $lfp(X, \psi)$ and $gfp(X, \psi)$, respectively.

[1] We do not consider bnodes.

RDF Transition System. An RDF transition system [9] is a labeled transition system, $K_G = (S = S' \cup S'', R, L)$, representation of an RDF graph G where two sets of nodes, S' and S'', are introduced: one set S'' for each triple (called triple node) and the other set S' for each subject, predicate, and object of each triple. A triple node (the black node in the figure below) is connected to its subject, predicate, and object nodes. For instance, the RDF graph $\boxed{\text{x}} \xrightarrow{z} \boxed{\text{y}}$ can be turned into the RDF transition system in Fig. 1, where $S' = \{n_1, n_2, n_3\}$, $S'' = \{t\}$, $R(s) = \{(n_1, t)\}, R(p) = \{(t, n_3)\}, R(o) = \{(t, n_2)\}$, and $L(x) = \{n_1\}, L(y) = \{n_2\}, L(z) = \{n_3\}$. Navigation from one node to another, in RDF transition systems, is done by using a set of transition programs $\{s, p, o\}$ and their converses. Since the μ-calculus with converse lacks functionality for number restrictions one cannot impose that each triple node is connected to exactly one node for each subject, predicate, and object node. However, one can impose a lighter restriction to achieve this by taking advantage of the technique introduced in [14] and adopted in [7]. Since it is not possible to ensure that there is only one successor, then we restrict all the successors to bear the same constraints; thus, they become interchangeable. This is achieved by rewriting the formulas using a function func such that all occurrences of $\langle a \rangle \varphi$ (existential formulas) are replaced by $\langle a \rangle \top \wedge [a] \varphi$; in Definition 2, func is defined inductively on the structure of a μ-calculus formula. When checking for query containment, we assume that the formulas are rewritten using the function func.

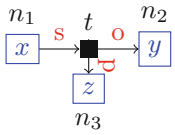

Fig. 1. Encoding of an RDF triple into a transition system.

Definition 2. func *is inductively defined on the structure of a* μ*-calculus formula as follows:*

$$\begin{aligned}
\mathsf{func}(\top) &= \top & \mathsf{func}(\bot) &= \bot \\
\mathsf{func}(q) &= q \quad q \in AP \cup Nom & \mathsf{func}(X) &= X \quad X \in Var \\
\mathsf{func}(\neg\varphi) &= \neg\mathsf{func}(\varphi) & \mathsf{func}(\varphi \wedge \psi) &= \mathsf{func}(\varphi) \wedge \mathsf{func}(\psi) \\
\mathsf{func}(\varphi \vee \psi) &= \mathsf{func}(\varphi) \vee \mathsf{func}(\psi) & \mathsf{func}(\langle a \rangle \varphi) &= \langle a \rangle \top \wedge [a]\mathsf{func}(\varphi) \quad a \in \text{Prog} \\
\mathsf{func}(\langle a \rangle \varphi) &= \langle a \rangle \mathsf{func}(\varphi) & \mathsf{func}([a]\varphi) &= [a]\mathsf{func}(\varphi) \\
\mathsf{func}(\mu X.\varphi) &= \mu X.\mathsf{func}(\varphi) & \mathsf{func}(\nu X.\varphi) &= \nu X.\mathsf{func}(\varphi)
\end{aligned}$$

Figure 2 shows an RDF graph G and an excerpt of its corresponding RDF transition system K_G. Each RDF triple requires the introduction of a transition node (i.e., black nodes in the figure), where: the subject of the RDF triple has an incoming edge to the transition node labeled as s and the predicate and object have outgoing edges labeled as p and o, respectively. At this point, we shall define the model of formula ψ in terms of RDF transition systems.

Definition 3 *(RDF transition system model). An RDF transition system* K_G *is considered as a* model *of formula* ψ *if there exists a node* s *in the transition system where* ψ *holds, i.e.,* $K_G, s \models \psi$*. If a formula has a model, then it is called satisfiable.*

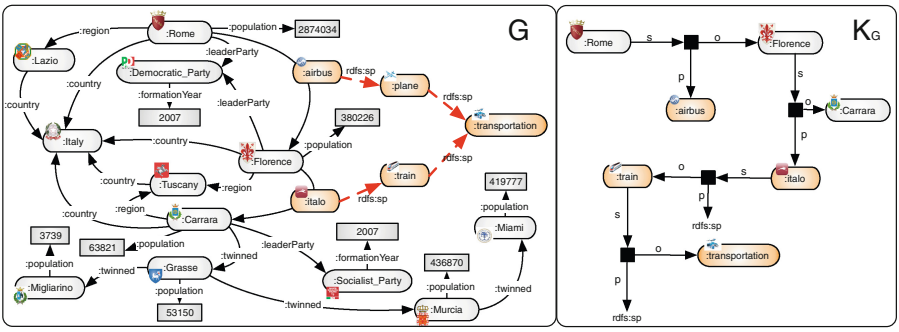

Fig. 2. An graph G and an excerpt of its corresponding RDF transition system K_G

The following lemma links RDF transition systems with transition systems when encoding in μ-calculus a query expressed in some RDF query language.

Lemma 1 (Chekol et al. [7]). *Let φ be a μ-calculus encoding of an EPPs query, φ is satisfied by some RDF transition system K_G if and only if* $\mathsf{func}(\varphi)$ *is satisfied by some transition system K.*

The above lemma serves as a basis for the encoding of EPPs that will be encoded in μ-calculus formulas and then can be interpreted over RDF transition systems.

3 Extended Property Paths

We now introduce our navigational extension of SPARQL called Extended Property Paths (EPPs) [12].

Syntax. EPPs extend PPs and NREs-like languages with path conjunction/negation, repetitions, and more types of tests. The syntax of EPPs is given below:

$$
\begin{aligned}
\mathtt{e} ::= & \text{`}\hat{}\text{'}\ \mathtt{e} \mid \mathtt{e}\ \text{`}+\text{'} \mid \mathtt{e}\ \text{`}?\text{'} \mid \mathtt{e}\text{`}*\text{'} \mid \mathtt{e}\ \text{`}/\text{'}\ \mathtt{e} \mid \mathtt{e}\ \text{`}|\text{'}\ \mathtt{e} \mid \\
& \text{`(' } \mathtt{e} \text{ `)'} \mid [\mathtt{p}]^1\ \mathtt{test}\ [\mathtt{p}]^2 \mid \mathtt{e}\ \text{`}\&\text{'}\ \mathtt{e} \mid \mathtt{e}\ \text{`}\sim\text{'}\ \mathtt{e} \mid \mathtt{e}\text{`}\{\text{'}l, h\text{`}\}\text{'} \\
\mathtt{test} ::= & \text{`!'}\ \mathtt{test} \mid \mathtt{test}\ \text{`\&\&'}\ \mathtt{test} \mid \mathtt{test}\ \text{`||'}\ \mathtt{test} \mid \text{`(' } \mathtt{test} \text{ `)'} \mid \mathtt{base} \\
\mathtt{base} ::= & \mathtt{iri} \mid \text{`}\hat{}\text{'}\mathtt{iri} \mid \text{`TP('}\mathtt{p}\ \text{`,'}\ \mathtt{e}\ \text{`)'} \mid \text{`T('}\mathtt{EExp}\text{`)'} \\
\mathtt{p} ::= & \text{`\_s'} \mid \text{`\_p'} \mid \text{`\_o'}
\end{aligned}
$$

[1]Default is _s; [2]Default is_o. We refer to EPPs without path repetition, path complement as ***c*EPPs**.

EPPs introduce the following new features: path conjunction ($\mathtt{e}_1\&\mathtt{e}_2$), path negation ($\mathtt{e}_1\sim\mathtt{e}_2$), path repetitions between l and h times ($\mathtt{e}\{l,h\}$), and different types of tests (`test`) *within* a path. EPPs allow to specify the *starting* and *ending* position (p) of a test; it is possible to test from each of the subject, predicate

and object positions in RDF triples, mapped in the EPPs syntax to the position symbols _s, _p and _o, respectively. Positions do not need to be always specified; by default a test starts from the subject (_s) and ends on the object (_o) of the triple being traversed. There are different types of tests (test); a test can be a simple check for the existence of a IRI in forward/reverse direction, a *nested* EPP, i.e., TP(P, e), which corresponds to the evaluation of the expression e starting from a position P (of the last triple traversed) and returns true iff there exists at least one node that can be reached via e. A test can also be of type T; here, EExp (not reported here for sake of space) extends the production [110] in the SPARQL grammar[2] which enables to use in EPPs tests available in SPARQL as built-in conditions. Tests can be combined via the logical operators AND (&&), OR (||) and NOT (!).

Positions and Tests. EPPs tests can be coupled with positions. To formally explain the reasoning behind tests and positions, we make usage of a function $\Pi(\mathtt{P}, t)$, which projects the element in position P of a triple t. If we have t= $\langle \mathtt{u}_1,\mathtt{p}_1,\mathtt{u}_2\rangle$ and the test T(_p=$\mathtt{p}_1$) then $\Pi(\_\mathtt{p},\langle \mathtt{u}_1,\mathtt{p}_1,\mathtt{u}_2\rangle)=\mathtt{p}_1$ that checks $\mathtt{p}_1$=$\mathtt{p}_1$, and, in this case, returns true; however, it returns false for T(_o=$\mathtt{p}_3$).

Example 1 (**Path Conjunction, Negation and Tests**). Find pairs of cities located in the same country but not in the same region; such cities must be governed by the same political party, which has been founded before 2010.

NREs-based languages (and PPs) *cannot* express such request due to the lack of path conjunction/negation. With EPPs it can be expressed as shown in Fig. 3.

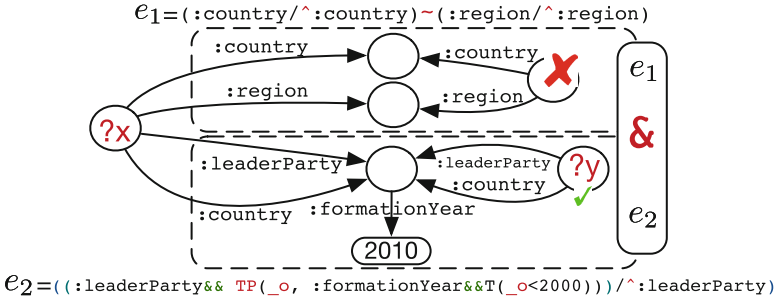

Fig. 3. EPP expression for Example 1.

The expression is the conjunction (&) of the two sub-expressions e_1 and e_2.

In the sub-expression e_1, the symbol ^ denotes backward navigation; from the object to the subject of a triple. Path negation ~ enables to discard from the set of cities in the same country (i.e., :country/^:country) those that are in the same region (i.e., :region/^:region). Path conjunction & enables to keep

[2] http://www.w3.org/TR/sparql11-query/#rExpression This is not considered when dealing with containment of EPPs.

from the set of nodes satisfying the first subexpression (e_1) those that also satisfy the second (e_2), i.e. governed by the same party, which has been founded before 2010. Note that PPs and NREs-based languages lack tests like TP, to check the existence of a path (via `:formationYear`) to nodes having value <2010.

Example 2 (**Positions and Tests**). Consider again the expression in Example 1. The expression including both default positions[3] and using positions to traverse backward edges is shown in Fig. 4.

As an example, the test TP (in Fig. 4) starts from the position _o, that is, the object of the last triple traversed.

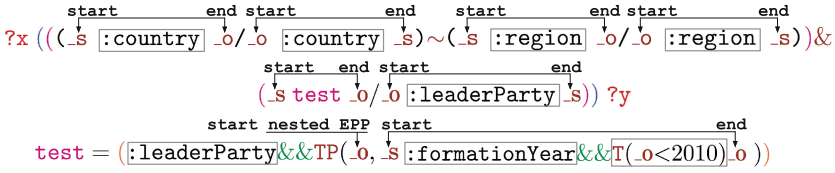

Fig. 4. An EPP expression with positions.

Definition 4. *An EPP query q has the form* $q ::= (\mathcal{V} \cup \boldsymbol{I}, e, \mathcal{V} \cup \boldsymbol{I} \cup \boldsymbol{L})$.

Semantics. For the purpose of this paper we will focus our attention on set semantics. This is in line with recent related work that studied the containment of PPs in SPARQL [20] where PPs were given a set-based semantics. Besides, the containment of conjunctive queries in the relational world when considering bag semantics is still an open issue[4]. Table 1 shows the semantics of EPPs. The semantics makes usage of two evaluation function; the first $\mathbf{E}[\![\cdot]\!]^G$ is used to evaluate all forms of EPPs on an RDF graph G but tests; these latter are handled by the boolean evaluation function $\mathbf{E}_T[\![\cdot]\!]^G_{\mathtt{t}}$. Note that in this latter case tests consider a triple $t \in G$ and have a start and end position. By observing rule R9 one may notice that the final result is a set of pair of nodes in the graph G; these pairs of nodes are obtained via the position mapping function Π that projects the two elements of the triple `t` appearing in position P_1 and P_2 if `t` satisfies the test.

4 Translating *c*EPPs into $\mathcal{L}_\mu$

The goal of this section is to provide the first building block toward tackling the containment of EPP expressions; this concerns the encoding of ***c*EPPs** into $\mathcal{L}_\mu$. We shall start by introducing the notion of path pattern translation.

[3] These are automatically added during the parsing phase.

[4] It is claimed that for basic CQs the complexity of query containment under bag semantics is in Π^p_2 [6].

Table 1. Set-based semantics for EPPs.

R1	$\mathbf{E}[\![\hat{}\,\mathtt{e}]\!]^G := \{(\mathtt{u},\mathtt{v}) : (\mathtt{v},\mathtt{u}) \in \mathbf{E}[\![\mathtt{e}]\!]^G\}$
R2	$\mathbf{E}[\![\mathtt{e}_1/\mathtt{e}_2]\!]^G := \{(\mathtt{u},\mathtt{v}) : \exists \mathtt{w}\ (\mathtt{u},\mathtt{w}) \in \mathbf{E}[\![\mathtt{e}_1]\!]^G \wedge (\mathtt{w},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_2]\!]^G\}$
R3	$\mathbf{E}[\![(\mathtt{e})^*]\!]^G := \{(\mathtt{u},\mathtt{u}) \mid \mathtt{u} \in nodes(G)\} \cup \bigcup_{i=1}^{\infty} \mathbf{E}[\![\mathtt{e}_i]\!]^G \mid \mathtt{e}_1 = \mathtt{e} \wedge \mathtt{e}_i = \mathtt{e}_{i-1}/\mathtt{e}$
R4	$\mathbf{E}[\![(\mathtt{e})^+]\!]^G := \bigcup_{i=1}^{\infty} \mathbf{E}[\![\mathtt{e}_i]\!]^G \mid \mathtt{e}_1 = \mathtt{e} \wedge \mathtt{e}_i = \mathtt{e}_{i-1}/\mathtt{e}$
R5	$\mathbf{E}[\![(\mathtt{e})?]\!]^G := \{(\mathtt{u},\mathtt{u}) \mid \mathtt{u} \in nodes(G)\} \cup \mathbf{E}[\![\mathtt{e}]\!]^G$
R6	$\mathbf{E}[\![(\mathtt{e}_1\vert\mathtt{e}_2)]\!]^G := \{(\mathtt{u},\mathtt{v}) : (\mathtt{u},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_1]\!]^G \vee\ (\mathtt{u},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_2]\!]^G\}$
R7	$\mathbf{E}[\![(\mathtt{e}_1\&\mathtt{e}_2)]\!]^G := \{(\mathtt{u},\mathtt{v}) : (\mathtt{u},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_1]\!]^G \wedge\ (\mathtt{u},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_2]\!]^G\}$
R8	$\mathbf{E}[\![(\mathtt{e}_1 \sim \mathtt{e}_2)]\!]^G := \{(\mathtt{u},\mathtt{v}) : (\mathtt{u},\mathtt{v}) \in \mathbf{E}[\![\mathtt{e}_1]\!]^G \wedge\ (\mathtt{u},\mathtt{v}) \notin \mathbf{E}[\![\mathtt{e}_2]\!]^G\}$
R9	$\mathbf{E}[\![\mathtt{P}_1\ \mathtt{test}\ \mathtt{P}_2]\!]^G := \{(\Pi(\mathtt{P}_1,\mathtt{t}),\Pi(\mathtt{P}_2,\mathtt{t}))) \mid \text{triple } \mathtt{t} \in G \wedge \mathbf{E}_T[\![\mathtt{test}]\!]_t^G\}$
R10	$\mathbf{E}_T[\![\mathtt{u}]\!]_{\mathtt{t}}^G := \Pi(\_\mathtt{p},\mathtt{t}) = \mathtt{u}$
R11	$\mathbf{E}_T[\![\mathtt{T}(\mathtt{EExp})]\!]_{\mathtt{t}}^G := \mathtt{EvalSPARQLBuilt\text{-}in}(\mathtt{EExp},\mathtt{t})$
R12	$\mathbf{E}_T[\![\mathtt{TP}(\mathtt{P},\mathtt{e})\ ]\!]_{\mathtt{t}}^G := \exists v : (\Pi(\mathtt{P},\mathtt{t}),v) \in \mathbf{E}[\![\mathtt{e}]\!]^G$
R13	$\mathbf{E}_T[\![\mathtt{test}_1\&\&\mathtt{test}_2\ ]\!]_{\mathtt{t}}^G := \mathbf{E}_T[\![\mathtt{test}_1\ ]\!]_{\mathtt{t}}^G \wedge \mathbf{E}_T[\![\mathtt{test}_2\ ]\!]_{\mathtt{t}}^G$
R14	$\mathbf{E}_T[\![\mathtt{test}_1\vert\vert\mathtt{test}_2\ ]\!]_{\mathtt{t}}^G := \mathbf{E}_T[\![\mathtt{test}_1\ ]\!]_{\mathtt{t}}^G \vee \mathbf{E}_T[\![\mathtt{test}_2\ ]\!]_{\mathtt{t}}^G$
R15	$\mathbf{E}_T[\![!\mathtt{test}\ ]\!]_{\mathtt{t}}^G := \neg\mathbf{E}_T[\![\mathtt{test}\ ]\!]_{\mathtt{t}}^G$

Definition 5 *(Path pattern translation). Given a* ***cEPPs*** *query* $q{::}=(\alpha, \mathtt{e}, \beta)$ *where* $\alpha \in \mathcal{V} \cup \boldsymbol{I}$ *and* $\beta \in \mathcal{V} \cup \boldsymbol{I} \cup \boldsymbol{L}$*, its translation into* $\mathcal{L}_\mu$ *is given by the following (sub)-formula:* $\mathsf{E}_\mu[q] = (\langle \bar{s} \rangle \alpha \wedge \langle p \rangle \mathsf{RE}_\mu(\mathtt{e}, \beta))$*, where* $\alpha, \beta \in AP \cup \top$*.*

The formula states the existence of the encoding of the path pattern (via $\mathsf{E}_\mu[\mathtt{e}]$) *somewhere* in a transition system and thus it is quantified by μ (least fixed point) so as to propagate the sub-formula to the entire transition system. μ encodes a reflexive transitive closure over all the programs and is denoted by $lfp(X, \mathsf{E}_\mu[q])$ [9]. The variables α and β are encoding as nominals whereas the IRIs in path expressions are encoded as atomic propositions. The encoding, shown in Table 2, makes usage of three functions

- RE_μ: takes an ***c*EPPs** expression $\mathtt{e}$ and a $\mathcal{L}_\mu$ formula φ and builds inductively an encoding based on the structure of the path expression.
- T_μ: takes a path test expression and inductively builds an $\mathcal{L}_\mu$ formula. The test positions are translated into transition programs in $\mathcal{L}_\mu$, i.e., _s becomes the transition program $\bar{s}$ whereas _p and _o become p and o programs, respectively.
- Γ: takes two (complex) ***c*EPPs** expressions and on the basis of the end position of the first and start position of the second returns transition programs expressed in $\mathcal{L}_\mu$ for the RDF transition system.

We now prove the correctness of the translation of ***c*EPPs** into $\mathcal{L}_\mu$ formulas as shown in the following Lemma.

Lemma 2. *Let* q *be a* ***cEPPs*** *query, for every RDF transition system* K_G *whose associated RDF graph is* G*, it holds that* $\mathbf{E}[\![\ q\ ]\!]^G \neq \varnothing$ *iff* $[\![\mathsf{E}_\mu(q)]\!]^{K_G} \neq \varnothing$*.*

Table 2. Encoding of EPPs into $\mathcal{L}_\mu$. The function Γ takes two (complex) EPP expressions and on the basis of the end position of the first and start position of the second returns transition programs expressed in $\mathcal{L}_\mu$ for the RDF transition system. Note that if $\mathtt{P} = \_s$ in line (14), (resp., $\mathtt{P}_1 = \_s$, $\mathtt{P}_2 = \_s$ in line (9)) then the translation into $\mathcal{L}_\mu$ is the transition program $\bar{s}$.

$$
\begin{array}{lll}
(1) & \mathsf{RE}_\mu(iri, \varphi) & = \begin{cases} \langle p\rangle iri \ \wedge \langle o\rangle\varphi & \text{if } \varphi \in AP \\ \langle p\rangle iri \ \wedge \varphi & \text{otherwise} \end{cases} \\
(2) & \mathsf{RE}_\mu(\hat{}\,\mathtt{e}, \varphi) & = \mathsf{RE}_\mu(\mathtt{e}, \langle \bar{s}\rangle\varphi) \\
(3) & \mathsf{RE}_\mu(\mathtt{e}_1 \mid \mathtt{e}_2, \varphi) & = \mathsf{RE}_\mu(\mathtt{e}_1, \varphi) \vee \mathsf{RE}_\mu(\mathtt{e}_2, \varphi)) \\
(4) & \mathsf{RE}_\mu(\mathtt{e}_1/\mathtt{e}_2, \varphi) & = \mathsf{RE}_\mu(\mathtt{e}_1, \Gamma(\mathtt{e}_1, \mathtt{e}_2)\mathsf{RE}_\mu(\mathtt{e}_2, \varphi)) \\
(5) & \mathsf{RE}_\mu((\mathtt{e}_1 \& \mathtt{e}_2), \varphi) & = (\mathsf{RE}_\mu(\mathtt{e}_1, \varphi) \wedge \mathsf{RE}_\mu(\mathtt{e}_2, \varphi)) \\
(6) & \mathsf{RE}_\mu(\mathtt{e}?, \varphi) & = \mathsf{RE}_\mu(\mathtt{e}, \varphi) \vee \langle \bar{s}\rangle\varphi \\
(7) & \mathsf{RE}_\mu(\mathtt{e}^+, \varphi) & = \mu X.\mathsf{RE}_\mu(\mathtt{e}, \varphi) \vee \mathsf{RE}_\mu(\mathtt{e}, \Gamma(\mathtt{e}, \mathtt{e})X) \\
(8) & \mathsf{RE}_\mu(\mathtt{e}^*, \varphi) & = \mathsf{RE}_\mu(\mathtt{e}^+, \varphi) \vee \langle \bar{s}\rangle\varphi \\
(9) & \mathsf{RE}(\mathtt{P}_1 \ \mathtt{test} \ \mathtt{P}_2, \varphi) & = \langle \mathtt{P}_1\rangle\top \wedge \mathsf{T}_\mu(\mathtt{test}) \wedge \langle \mathtt{P}_2\rangle\top \wedge \varphi \\
(10) & \mathsf{T}_\mu(iri) & = \langle p\rangle iri \\
(11) & \mathsf{T}_\mu(\mathtt{test}_1 \ \&\& \ \mathtt{test}_2) & = \mathsf{T}_\mu(\mathtt{test}_1) \wedge \mathsf{T}_\mu(\mathtt{test}_2) \\
(12) & \mathsf{T}_\mu(\mathtt{test}_1 \ || \ \mathtt{test}_2) & = \mathsf{T}_\mu(\mathtt{test}_1) \vee \mathsf{T}_\mu(\mathtt{test}_2) \\
(13) & \mathsf{T}_\mu(!\mathtt{test}) & = \neg\mathsf{T}_\mu(\mathtt{test}) \\
(14) & \mathsf{T}_\mu(\mathtt{TP}(\mathtt{P}, \mathtt{e})) & = \langle \mathtt{P}\rangle\mathsf{RE}_\mu(\mathtt{e}, \top) \\
(15) & \Gamma(\mathtt{e}_1, \mathtt{e}_2) & = \langle o\rangle\langle s\rangle \\
(16) & \Gamma(\mathtt{e}_1, \hat{}\,\mathtt{e}_2) & = \langle o\rangle\langle \bar{o}\rangle \\
(17) & \Gamma(\hat{}\,\mathtt{e}_1, \mathtt{e}_2) & = \langle o\rangle\langle s\rangle \\
(18) & \Gamma(\hat{}\,\mathtt{e}_1, \hat{}\,\mathtt{e}_2) & = \langle o\rangle\langle \bar{o}\rangle
\end{array}
$$

Proof ($\Rightarrow$). Assume that there exists an RDF G such that the evaluation of q over G is nonempty, i.e., $\mathbf{E}\ [\![q]\!]G \neq \varnothing$. We can build such graph from the canonical instance of q and it can be produced using a function θ as shown below:

- if $(\alpha, iri, \beta) \in q$, then $\theta((\alpha, iri, \beta)) = (\alpha, y, \beta) \in G$,
- if $(\alpha, \mathtt{e}, \beta) \in q$, then $\theta((\alpha, \mathtt{e}, \beta)) \in G$,
- if $(\alpha, \hat{}\,\mathtt{e}, \beta) \in q$, then $\theta((\alpha, \hat{}\,\mathtt{e}, \beta)) = (\beta, \mathtt{e}, \alpha) \in G$,
- if $(\alpha, \mathtt{e}_1/\mathtt{e}_2, \beta) \in q$, then $\theta((\alpha, \mathtt{e}_1, y)) \in G$ and $\theta((y, \mathtt{e}_2, \beta)) \in G$,
- if $(\alpha, \mathtt{e}_1|\mathtt{e}_2, \beta) \in q$, then $\theta((\alpha, \mathtt{e}_1, \beta)) \in G$ or $\theta((\alpha, \mathtt{e}_2, \beta)) \in G$,
- if $(\alpha, \mathtt{e}_1 \ \& \ \mathtt{e}_2, \beta) \in q$, then $\theta((\alpha, \mathtt{e}_1, \beta)) \in G$ and $\theta((\alpha, \mathtt{e}_2, \beta)) \in G$,
- if $(\alpha, \mathtt{e}^+, \beta) \in q$, then $\bigcup_{i=1}^{n} \theta((\alpha, \mathtt{e}^i, \beta)) \in G$, where $\mathtt{e}^i$ denotes the composition of $\mathtt{e}$ i times and $n \in \mathbb{N}$,
- if $(\alpha, \mathtt{e}^*, \beta) \in q$, then $(\alpha, u, \alpha) \in G$ or $(\beta, u, \beta) \in G$ or $\theta((\alpha, \mathtt{e}^+, \beta)) \in G$, where u is a fresh IRI,
- if $(\alpha, \mathtt{P}_1 \ iri \ \mathtt{P}_2, \beta) \in q$, then $(\alpha, iri, \beta) \in G$,
- if $(\alpha, \mathtt{P}_1 \ \mathtt{test} \ \mathtt{P}_2, \beta) \in q$, then $\theta((\alpha, \mathtt{P}_1 \ \mathtt{test} \ \mathtt{P}_2, \beta)) \in G$,

- if $(\alpha, \mathrm{P}_1\ \mathtt{test}_1\ \mathrm{P}_2\ \&\&\ \mathrm{P}_3\ \mathtt{test}_2\ \mathrm{P}_4, \beta) \in q$, then $\theta((\alpha, \mathrm{P}_1\ \mathtt{test}_1\ \mathrm{P}_2, \beta)) \in G$ and $\theta((\alpha, \mathrm{P}_3\ \mathtt{test}_2\ \mathrm{P}_4, \beta)) \in G$,
- if $(\alpha, \mathrm{P}_1\ \mathtt{test}_1\ \mathrm{P}_2\ ||\ \mathrm{P}_3\ \mathtt{test}_2\ \mathrm{P}_4, \beta) \in q$, then $\theta((\alpha, \mathrm{P}_1\ \mathtt{test}_1\ \mathrm{P}_2, \beta)) \in G$ or $\theta((\alpha, \mathrm{P}_3\ \mathtt{test}_2\ \mathrm{P}_4, \beta)) \in G$,
- if $(\alpha, \mathrm{P}_1\ !\mathtt{test}\ \mathrm{P}_2, \beta) \in q$, then we introduce a set of fresh IRI's iri_i that do not appear in $\mathtt{test}$ such that $(\alpha, iri_i, \beta) \in G$ and $\theta(\alpha, \mathtt{test}, \beta) \notin G$,
- if $(\alpha, \mathrm{P}_1\ \mathtt{TP}(\mathrm{P}, \mathtt{e})\ \mathrm{P}_2, \beta) \in q$, then $(\alpha, y, \beta) \in G$ and $\theta(y, \mathtt{e}, r) \in G$ or $\theta(\alpha, \mathtt{e}, r) \in G$ or $\theta(\beta, \mathtt{e}, r) \in G$, where y and r are fresh IRIs.

Since G is an instance of q, the evaluation of q over G is not empty. Now, we construct an RDF transition system $K_G = (S, R, L)$ in the same way as it is done in Definition 9 of [7]. It is possible to verify K_G is a model of $\mathsf{E}_\mu(q)$ by working inductively on the construction of $\mathsf{E}_\mu(q)$. This is because atomic propositions encoding the constants and distinguished variables are true in K_G as they exist in G; therefore, $\mathsf{E}_\mu(q)$ is satisfiable in K_G. To elaborate, if l is a distinguished variable (i.e., either x or z) or a constant $\mathtt{e}$ in q, then

- for l a distinguished variable, l is satisfiable in K_G since $[\![l]\!]^{K_G}$ is non empty,
- for $l = \mathtt{e}$ an EPP, its encoding $\mathsf{RE}_\mu(\mathtt{e}, \top)$ is satisfiable in K_G if $\mathtt{e}$ is satisfiable in K_G, this can be proved inductively on the structure of the encoding.

Thus, since K_G is an RDF transition system, we obtain that $[\![\mathsf{E}_\mu(q)]\!]^{K_G} \neq \varnothing$.

($\Leftarrow$) Assume that $[\![\mathsf{E}_\mu(q)]\!]^{K_G} \neq \varnothing$. In order to test if the evaluation of q over an RDF graph obtained from K_G is non empty, we produce an RDF graph G from K_G as done in Lemma 4 of [7]. Thus, since G is a technical construction obtained from an RDF transition system associated to q, it holds that $\mathbf{E}[\![\ q\ ]\!]^G \neq \varnothing$.

Example 3. The μ-calculus encoding of the EPPs in Example 1 is given below. $\Gamma(et1, et2)$ is a shorthand for $\Gamma(:\mathtt{leaderParty}\ \&\&\ \mathtt{TP}(\_o, :\mathtt{formationYear}), \hat{}:\mathtt{leaderParty})$.

$$
\begin{aligned}
\mathsf{E}_\mu(\mathbf{e1}\ \&\ \mathbf{e2}) &= lfp\big(X, \langle \bar{s} \rangle \top \wedge \mathsf{RE}_\mu(\mathbf{e1}\ \&\ \mathbf{e2}, \top)\big)\\
\mathsf{RE}_\mu(\mathbf{e1}\ \&\ \mathbf{e2}, \top) &= \mathsf{RE}_\mu(\mathbf{e1}, \top) \wedge \mathsf{RE}_\mu(\mathbf{e2}, \top)\\
\mathsf{RE}_\mu(\mathbf{e1}, \top) &= \mathsf{RE}_\mu((:\mathtt{country}/\hat{}:\mathtt{country}) \sim (:\mathtt{region}/\hat{}:\mathtt{region}), \top)\\
&= \mathsf{RE}_\mu((:\mathtt{country}/\hat{}:\mathtt{country}), \top) \wedge \neg \mathsf{RE}_\mu((:\mathtt{region}/\hat{}:\mathtt{region}), \top)\\
&= \mathrm{blue}(\mathsf{RE}_\mu(:\mathtt{country}, \Gamma(:\mathtt{country}, \hat{}:\mathtt{country})\mathsf{RE}_\mu(\hat{}:\mathtt{country}, \top))\mathrm{blue}) \wedge\\
&\quad \neg \mathrm{blue}(\mathsf{RE}_\mu(:\mathtt{region}, \Gamma(:\mathtt{region}, \hat{}:\mathtt{region})\mathsf{RE}_\mu(\hat{}:\mathtt{region}, \top))\mathrm{blue})\\
&= \mathrm{blue}(\langle p \rangle :\mathtt{country} \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{country} \wedge \top)\mathrm{blue}) \wedge \neg \mathrm{blue}(\langle p \rangle :\mathtt{region} \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{region} \wedge \langle \bar{s} \rangle \top)\mathrm{blue})\\
\mathsf{RE}_\mu(\mathbf{e2}, \top) &= \mathsf{RE}_\mu(\mathrm{blue}(:\mathtt{leaderParty}\ \&\&\ \mathtt{TP}(\_o, :\mathtt{formationYear})\mathrm{blue})/\hat{}:\mathtt{leaderParty}, \top)\\
&= \mathsf{RE}_\mu(:\mathtt{leaderParty}\ \&\&\ \mathtt{TP}(\_o, :\mathtt{formationYear}), \Gamma(et1, et2)\mathsf{RE}_\mu(\hat{}:\mathtt{leaderParty}, \top))\\
&= \mathsf{RE}_\mu(:\mathtt{leaderParty}, \Gamma(et1, et2)\mathsf{RE}_\mu(\hat{}:\mathtt{leaderParty}, \top)) \wedge\\
&\quad \mathsf{RE}_\mu(\mathtt{TP}(\_o, :\mathtt{formationYear}), \Gamma(et1, et2)\mathsf{RE}_\mu(\hat{}:\mathtt{leaderParty}, \top))\\
&= \mathrm{blue}(\langle p \rangle :\mathtt{leaderParty} \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top)\mathrm{blue}) \wedge\\
&\quad \mathsf{RE}_\mu(\mathtt{TP}(\_o, :\mathtt{formationYear}), \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top))\\
&= \mathrm{blue}(\langle p \rangle :\mathtt{leaderParty} \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top)\mathrm{blue}) \wedge\\
&\quad (\langle \bar{s} \rangle \top \wedge \mathsf{T}_\mu(\mathtt{TP}(\_o, :\mathtt{formationYear})) \wedge \langle o \rangle \top \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top))\\
&= \mathrm{blue}(\langle p \rangle :\mathtt{leaderParty} \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top)\mathrm{blue}) \wedge\\
&\quad (\langle \bar{s} \rangle \top \wedge \langle o \rangle :\mathtt{formationYear} \wedge \langle p \rangle \top \wedge \langle o \rangle \top \wedge \langle o \rangle \langle \bar{o} \rangle (\langle p \rangle :\mathtt{leaderParty} \wedge \top))
\end{aligned}
$$

So far, we have shown that our encoding of ***c*EPPs** queries into the μ-calculus is correct. Next, we formally present the reduction of the containment test of ***c*EPPs** into unsatisfiability test in the μ-calculus.

5 Containment Results

In this section we state our complexity results. We start with ***c*EPPs** and then move to SPARQL with ***c*EPPs**. We start with the containment of ***c*EPPs** which can be stated as follows:

Problem: *c*EPPs Containment	
Input:	***c*EPPs** q_1 and q_2
Output:	Is $q_1 \sqsubseteq q_2$?

Given two ***c*EPPs** queries q_1 and q_2, $q_1 \sqsubseteq q_2$ if and only if for any RDF graph G, the answers of q_1 are included in the answers of q_2, i.e., $\mathbf{E}[\![\ q_1\ ]\!]^G \subseteq \mathbf{E}[\![\ q_2\ ]\!]^G$.

In the previous section, we have presented various functions to produce $\mathcal{L}_\mu$ formulas corresponding to the encodings of ***c*EPPs**. Hence, the problem of containment can be reduced to formula unsatisfiability test in $\mathcal{L}_\mu$ as:

Problem: *c*EPPs Containment via $\mathcal{L}_\mu$	
Input:	$\mathcal{L}_\mu$ encodings $\mathsf{E}_\mu(q_1)$ and $\mathsf{E}_\mu(q_2)$
Output:	Is $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ unsatisfiable?

Therefore, we obtain that $q_1 \sqsubseteq q_2 \Leftrightarrow \mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is unsatisfiable. We now show that our decision procedure is sound and complete.

Theorem 1 (Soundness). *Given two **cEPPs** queries q_1 and q_2 if $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is unsatisfiable, then $q_1 \sqsubseteq q_2$.*

Proof. We show the contrapositive. If $q_1 \not\sqsubseteq q_2$, then $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is satisfiable. One can verify that every RDF graph G in which there is at least one tuple satisfying q_1 but not q_2 can be turned into a RDF transition system model for $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$. To do so, consider an RDF graph G and assume that there is exists a mapping $m \in \mathbf{E}[\![\ q_1\ ]\!]^G$ and $m \notin \mathbf{E}[\![\ q_2\ ]\!]^G$. Let us construct an RDF transition system K_G from G. In fact, we produce G which contains at least a canonical instantiation of q_1, i.e., by replacing the variables in q_1 with their mappings in m. It has been shown that an RDF graph can be turned into an RDF transition system (cf. Lemma 1). At this point, it remains to verify that $[\![\mathsf{E}_\mu(q_1)]\!]^{K_G} \neq \varnothing$ and $[\![\mathsf{E}_\mu(q_2)]\!]^{K_G} = \varnothing$.

Let us construct the formulas $\mathsf{E}_\mu(q_1)$ and $\mathsf{E}_\mu(q_1)$ by first skolemizing the distinguished variables using their mappings in m. Consequently, from Lemma 2 one obtains, $[\![\mathsf{E}_\mu(q_1)]\!]^{K_G} \neq \varnothing$. However, $[\![\mathsf{E}_\mu(q_2)]\!]^{K_G} = \varnothing$, this is because the atomic propositions in the formula corresponding to the constants and variables are not satisfied in K. This implies that $[\![\neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing$. This is justified by the fact that if a formula φ is satisfiable in an RDF transition system, then $[\![\varphi]\!]^{K_G} = S$ thus $[\![\neg\varphi]\!]^{K_G} = \varnothing$. So far we have: $[\![\mathsf{E}_\mu(q_1)]\!]^{K_G} \neq \varnothing$ and $[\![\neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing$. Without loss of generality, $[\![\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing$. Therefore, $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is satisfiable.

Theorem 2 (Completeness). *Given two **cEPPs** queries q_1 and q_2, if $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is satisfiable, then $q_1 \not\sqsubseteq q_2$.*

Proof. $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is satisfiable $\Rightarrow \exists K_G.[\![\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing$. Consequently, K_G is an RDF transition system as shown in Proposition 1 of [7].

Using $K_G = (S' \cup S'', R, L)$, we construct an RDF graph G such that we can show that $\mathbf{E}[\![\ q_1\ ]\!]^G \not\subseteq \mathbf{E}[\![\ q_2\ ]\!]^G$ and hence $q_1 \not\sqsubseteq q_2$ holds:

– for each `iri` or constant u in q_1 and q_2, we create an RDF triple (α, u, β) such that $\{(\alpha,\beta) \mid \exists n \in [\![u]\!]^{K_G} \wedge\ n' \in S'' \wedge\ (\alpha, n') \in R(s) \wedge (n', n) \in R(p) \wedge (n', \beta) \in R(o)\}$.

Thus, it remains to show that $\mathbf{E}[\![\ q_1\ ]\!]^G \not\subseteq \mathbf{E}[\![\ q_2\ ]\!]^G$. From our assumption, we obtain the following:

$$[\![\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing \Rightarrow [\![\mathsf{E}_\mu(q_1)]\!]^{K_G} \neq \varnothing \text{ and } [\![\neg\mathsf{E}_\mu(q_2)]\!]^{K_G} \neq \varnothing.$$
$$\Rightarrow [\![\mathsf{E}_\mu(q_1)]\!]^{K_G} \neq \varnothing \text{ and } [\![\mathsf{E}_\mu(q_2)]\!]^{K_G} = \varnothing.$$

Note here that, if a formula φ is satisfiable in an RDF transition system K, then $[\![\varphi]\!]^{K_G} = S$ as shown in [7]. Consequently, using Lemma 2 and G, we obtain $\mathbf{E}[\![\ q_1\ ]\!]^G \neq \varnothing$ and $\mathbf{E}[\![\ q_2\ ]\!]^G = \varnothing$ because G contains all those triples that satisfy q_1 and not q_2. Therefore, we get $\mathbf{E}[\![\ q_1\ ]\!]^G \not\subseteq \mathbf{E}[\![\ q_2\ ]\!]^G$ and thus $q_1 \not\sqsubseteq q_2$.

From Theorems 1 and 2, we obtain the following result.

Proposition 1 (Complexity of *cEPPs* Containment). *The containment of **cEPPs** queries can be determined in a double exponential amount time.*

Note that the size of the encoding n is exponential. Hence, we obtain a 2EXPTIME upper bound for containment. If we remove path conjunction from the ***c*EPPs** we obtain an EXPTIME upper bound; this is due to the complexity of satisfiability in the μ-calculus which is $2^{\mathcal{O}(n^2 log\ n)}$ where n is the size of the encoding (which is exponential for ***c*EPPs**). To provide a lower bound to the problem of containment in ***c*EPPs**, we utilize the close connection between ***c*EPPs** and PDL (Propositional Dynamic Logic) with path intersection and converse [17]. Like $\mathcal{L}_\mu$, PDL formulas are interpreted over transition systems (aka. Kripke structures). ***c*EPPs** without tests are exactly the same as that of PDL navigational programs (which are regular expressions). PDL formulas can be interpreted over RDF transitions systems by using a fixed set of programs $\{s, p, o, \bar{s}, \bar{p}, \bar{o}\}$ as we have done for $\mathcal{L}_\mu$. Likewise, ***c*EPPs** expressions can be evaluated over RDF transition systems by a simple rewriting. Given a ***c*EPPs** expression $\mathtt{e}$ that can be expressed over a standard RDF graph, it can be turned into an expression $\mathtt{e}'$ that can be expressed over an RDF transition system by a bijective rewriting function ϑ. ϑ is defined inductively:

$$\vartheta(iri) = p/\mathtt{T}(\_o = iri)/\hat{}p/o \qquad \vartheta(\hat{}\mathtt{e}) = p/\mathtt{T}(\_o = iri)/\hat{}p/\hat{}s$$
$$\vartheta(\mathtt{e}_1/\mathtt{e}_2) = \vartheta(\mathtt{e}_1)/pos(\mathtt{e}_1, \mathtt{e}_2)/\vartheta(\mathtt{e}_2) \qquad \vartheta((\mathtt{e})^*) = (\vartheta(\mathtt{e}))^*$$
$$\vartheta((\mathtt{e})^+) = (\vartheta(\mathtt{e}))^+ \qquad \vartheta(\mathtt{e}_1 \mid \mathtt{e}_2) = \vartheta(\mathtt{e}_1) \mid \vartheta(\mathtt{e}_2)$$
$$\vartheta(\mathtt{e}_1 \& \mathtt{e}_2) = \vartheta(\mathtt{e}_1)\ \&\ \vartheta(\mathtt{e}_2) \qquad \vartheta(\mathtt{P}_1\ \mathtt{test}\ \mathtt{P}_2) = \vartheta_\mathtt{T}(\mathtt{P}_1)/\vartheta_\mathtt{T}(\mathtt{test})/\vartheta_\mathtt{T}(\mathtt{P}_2)$$
$$\vartheta_\mathtt{T}(iri) = p/\mathtt{T}(\_o = iri)/\hat{}p \qquad \vartheta_\mathtt{T}(\mathtt{test}_1\ \&\&\ \mathtt{test}_2) = \vartheta_\mathtt{T}(\mathtt{test}_1)\ \&\&\ \vartheta_\mathtt{T}(\mathtt{test}_2)$$
$$\vartheta_\mathtt{T}(\mathtt{test}_1\ ||\ \mathtt{test}_2) = \vartheta_\mathtt{T}(\mathtt{test}_1)\ ||\ \vartheta_\mathtt{T}(\mathtt{test}_2) \qquad \vartheta_\mathtt{T}(\mathtt{TP}(\mathtt{P}, \mathtt{e})) = \vartheta_\mathtt{T}(\mathtt{P})/\vartheta(\mathtt{e})$$
$$\vartheta_\mathtt{T}(!\mathtt{test}) = !\vartheta_\mathtt{T}(\mathtt{test}) \qquad \vartheta_\mathtt{T}(\_s) = s \quad \vartheta_\mathtt{T}(\_p) = p \quad \vartheta_\mathtt{T}(\_o) = o$$
$$pos(\mathtt{e}_1, \mathtt{e}_2) = s \qquad pos(\mathtt{e}_1, \hat{}\mathtt{e}_2) = \hat{}o \qquad pos(\hat{}\mathtt{e}_1, \mathtt{e}_2) = s \qquad pos(\hat{}\mathtt{e}_1, \hat{}\mathtt{e}_2) = \hat{}o$$

where s, p and o are navigational programs of an RDF transition system. In order to show that ***c*EPPs** tests are just a syntactic variant of PDL node formulas (aka XPath node expressions), we prove the following.

Lemma 3. *Satisfiability of* ***cEPPs*** *queries on RDF transition systems can be reduced in polynomial time to satisfiability of* ***cEPPs*** *queries on RDF graphs.*

Proof. We can turn an RDF transition system into a standard RDF graph by using a bijective mapping function similar to the one in [7]*. It is also possible to transform a* ***c*EPPs** *expression* $\mathtt{e}$ *over an RDF transition system* K_G *into a* ***c*EPPs** *expression* $\mathtt{e}'$ *that can be expressed over a standard RDF graph* G *via the function* ϑ*. Thus, it remains to show that,* $\mathtt{e}$ *is satisfiable in* K_G *iff* $\mathtt{e}'$ *is satisfiable in* G*. This can be done inductively on the construction of the expressions.*

Theorem 3. *Testing containment of* ***cEPPs*** *queries is 2EXPTIME-hard.*

The above result is obtained by examining the connection between ***c*EPPs** and Propositional Dynamic Logic with path intersection (ICPDL). ***c*EPPs** can be as seen as a notational variant of ICPDL with the fixed set of atomic programs $\{s, p, o, \bar{s}, \bar{p}, \bar{o}\}$. For example, the ***c*EPPs** expression $\vartheta(a/\mathtt{TP}(\_s, b^+)/c) = s/a/...$ corresponds to the ICPDL formula $a \circ \langle s \rangle \langle b^+ \rangle \circ \langle c \rangle$.

The double exponential upper bound is unavoidable as it has already been shown that path conjunction makes the containment problem very difficult for XPath; it has been proved that the complexity of the containment of CoreXPath($*$, $\cap$) expressions is 2EXPTIME-complete [30]. Consider now the language of ***c*EPPs** without path intersection (***c*EPPs**$_{nn}$). This language is closely related to PDL and CoreXPath [30].

Theorem 4. *The containment problem of* ***cEPPs***$_{nn}$ *is EXPTIME-complete.*

The following result follows from the containment problem for GXPath$_{reg}$ (Graph XPath) [21] and the satisfiability problem for PDL with negation on paths [18].

Theorem 5. *The containment problem of EPPs is undecidable.*

5.1 Containment of SPARQL with EPPs

In this section we briefly discuss the containment problem for SPARQL queries *with EPPs* (from here onwards, we refer to it as just SPARQL). We restrict ourselves to the union of conjunctive fragment of SPARQL, i.e., *AND-UNION fragment of SPARQL, without projection.*

Definition 6. *A SPARQL query* q *is defined inductively as:* $q ::= (\mathcal{V} \cup \boldsymbol{I}, \mathtt{e}, \mathcal{V} \cup \boldsymbol{I} \cup \boldsymbol{L}) \mid q$ **AND** $q' \mid q$ **UNION** q'.

The problem of the containment of SPARQL queries can be reduced into unsatisfiability test in $\mathcal{L}_\mu$ as given below:

Problem: SPARQL Containment via $\mathcal{L}_\mu$	
Input:	$\mathcal{L}_\mu$ encodings $\mathsf{E}(q_1)$ and $\mathsf{E}(q_2)$
Output:	Is $\mathsf{E}(q_1) \wedge \neg\mathsf{E}(q_2)$ unsatisfiable?

In the following, we extend the EPPs encoding function E_μ to translate the containment test of q_1 in q_2 into the μ-calculus.

Definition 7 (SPARQL containment). *The encoding of a SPARQL query q is obtained inductively as follows:*

$$\begin{aligned} \mathsf{E}_\mu((\alpha, e, \beta)) &= \mathit{lfp}(X, \langle \bar{s}\rangle\alpha \wedge \langle p\rangle \mathsf{RE}_\mu(e, \beta)) \\ \mathsf{E}_\mu(q \ \textbf{AND}\ q') &= \mathsf{E}_\mu(q) \wedge \mathsf{E}_\mu(q') \\ \mathsf{E}_\mu(q \ \textbf{UNION}\ q') &= \mathsf{E}_\mu(q) \vee \mathsf{E}_\mu(q') \end{aligned}$$

The variables on the left hand side query are encoded into nominals and the IRIs in the path expressions are encoded into atomic propositions; to encode variables that appear on the right hand side query, we follow two steps: (1) if a variable appears uniquely in the query, it is encoded into $\top$, and (2) if a variable appears multiple times, it is encoded by using the IRIs in the path expression (the triple pattern in which it appears). This technique has been already used in [9]. For instance, if the query is $q = (x, \mathsf{e}_1, y)$ **AND** (y, e_2, z), then the encoding of x (resp. z) is $\top$ and $y \rightarrow \langle \bar{o}\rangle\langle p\rangle \mathsf{e}_1$ or $y \rightarrow \langle s\rangle\langle p\rangle \mathsf{e}_2$. Thus, the encoding becomes:

$$\mathsf{E}_\mu(qp) = \mathsf{E}_\mu((\top, \mathsf{e}_1, \langle \bar{o}\rangle\langle p\rangle \mathsf{e}_1) \ \textbf{AND}\ (\langle \bar{o}\rangle\langle p\rangle \mathsf{e}_1, \mathsf{e}_2, \top)) \vee$$

$$\mathsf{E}_\mu((\top, \mathsf{e}_1, \langle s\rangle\langle p\rangle \mathsf{e}_2) \ \textbf{AND}\ (\langle s\rangle\langle p\rangle \mathsf{e}_2, \mathsf{e}_2, \top))$$

The size of the encoding for the containment problem is polynomial in the number of variables that appear more than once. To be more precise, for a given query, the size of the encoding is: $\Pi_{x \in multiVar}|x|$ where *multiVar* is the set of variables occurring more than once, and $|x|$ is the number of times variable x appears in the query. Note that the size of the encoding is linear if all the variables on the right-hand side query appear uniquely.

Theorem 6 (Soundness and Completeness). *Given two SPARQL queries q_1 and q_2, $\mathsf{E}_\mu(q_1) \wedge \neg\mathsf{E}_\mu(q_2)$ is unsatisfiable if and only if $q_1 \sqsubseteq q_2$.*

From the above theorem, we obtain the following result:

Proposition 2. *Given two SPARQL queries q_1 and q_2, the containment test can be solved in a double exponential amount of time in the size of the encoding $|\mathsf{E}_\mu(q_1)| + |\mathsf{E}_\mu(q_2)|$.*

This result is not surprising, as we have shown in Proposition 1 that the complexity of the containment of EPPs is 2EXPTIME in the worst case. Furthermore, if projection was part of the SPARQL fragment we consider here, then there is a further jump in the complexity of containment, i.e., the complexity increases by one exponential. Thereby, we obtain 3EXPTIME upper bound for the containment of SPARQL queries with projection. This is due to an exponential blow up in the size of the encoding as one needs to take care of multiple occurring non-distinguished variables.

6 Conclusions and Future Work

We have discussed a preliminary study of the problem of query containment for an expressive class of navigational queries captured by the EPPs language. Our study leverages μ-calculus to encode EPPs and then use this encoding to get a 2EXPTIME complexity upper bound. This bound remains the same when considering EPPs in SPARQL without projection. However, if one considers projection, there is an exponential increase in the complexity which results in a 3EXPTIME bound. While the results obtained are of theoretical interest, from a practical point of view an implementation is available (http://sparql-qc-bench.inrialpes.fr/) which can be extended for EPPs. Furthermore, an additional benefit of using μ-calculus is that by exploiting *logical combinators* the size of the encoding can be reduced by upto exponentiation. Thus, the complexity bounds that we obtained are not prohibitive in terms of a practical implementation.

A natural line of future research is to provide a tighter complexity bound for the problem of containment of EPPs. Moreover, the investigation of how the inclusion of constraints in EPPs affects the complexity of query containment is in our research agenda.

References

1. Barcelo, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM Trans. Database Syst. (TODS) **37**, 31 (2012)
2. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Proceedings of 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2000), pp. 176–185 (2000)
3. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Conjunctive query containment and answering under description logic constraints. ACM Trans. Comput. Logic (TOCL) **9**(3), 22 (2008)
4. Calvanese, D., Ortiz, M., Simkus, M.: Containment of regular path queries under description logic constraints. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, p. 805 (2011)
5. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC 1977, pp. 77–90. ACM, New York (1977)
6. Chaudhuri,S., Vardi, M.Y.: Optimization of real conjunctive queries. In: Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1993, pp. 59–70. ACM, New York (1993)
7. Chekol, M.W.: Static analysis of semantic web queries. Ph.D. thesis, Université de Grenoble (2012)
8. Chekol, M.W., Euzenat, J., Genevès, P., Layaïda, N.: PSPARQL query containment. In: Proceedings of DBPL (2011)
9. Chekol, M.W., Euzenat, J., Genevès, P., Layaïda, N.: SPARQL query containment under RDFS entailment regime. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 134–148. Springer, Heidelberg (2012)

10. Chekol, M.W., Euzenat, J., Genevès, P., Layaïda, N.: SPARQL query containment under SHI axioms. In: Proceedings of AAAI, pp. 10–16 (2012)
11. Fionda, V., Pirrò, G.: Querying graphs with preferences. In: 22nd ACM International Conference on Information and Knowledge Management, CIKM 2013, San Francisco, CA, USA, October 27– November 1, pp. 929–938 (2013)
12. Fionda, V., Pirrò, G., Consens, M.P.: Extended property paths: writing more SPARQL queries in a succinct way. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30 January, Austin, Texas, USA, pp. 102–108 (2015)
13. Fionda, V., Pirrò, G., Gutierrez, C.: NautiLOD: a formal language for the web of data graph. Trans. Web **9**(1), 5: 1–5: 43 (2015)
14. Genevès, P., Layaïda, N.: A system for the static analysis of XPath. ACM Trans. Inf. Syst. **24**(4), 475–502 (2006)
15. Genevès, P., Layaïda, N., Schmitt, A.: Efficient static analysis of XML paths and types. In: Proceedings of PLDI, pp. 342–351. ACM (2007)
16. Genevès, P., Schmitt, A. : Expressive logical combinators for free. In: IJCAI, pp. 311–317 (2015)
17. Göller, S., Lohrey, M., Lutz, C.: PDL with intersection and converse: satisfiability and infinite-state model checking. J. Symb. Logic **74**(01), 279–314 (2009)
18. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
19. Ioannidis, Y.E., Ramakrishnan, R.: Containment of conjunctive queries: beyond relations as sets. ACM Trans. Database Syst. **20**(3), 288–324 (1995)
20. Kostylev, E.V., Reutter, J.L., Romero, M., Vrgoč, D.: SPARQL with property paths. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 3–18. Springer, Heidelberg (2015)
21. Kostylev, E.V., Reutter, J.L., Vrgoc, D.: Containment of data graph queries. In: ICDT, pp. 131–142 (2014)
22. Kostylev, E.V., Reutter, J.L., Vrgoč, D.: Static analysis of navigational XPath over graph databases. Inf. Process. Lett. **116**(7), 467–474 (2016)
23. Kozen, D.: Results on the propositional μ-calculus. Theoret. Comput. Sci. **27**(3), 333–354 (1983)
24. Krötzsch, M., Rudolph, S.: Conjunctive queries for EL with composition of roles. In: Proceedings of the 2007 International Workshop on Description Logics (DL 2007), Brixen-Bressanone, near Bozen-Bolzano, 8–10 Italy 2007, June 2007
25. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. (TODS) **34**(3), 16 (2009)
26. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: a navigational language for RDF. Web Semant. Sci. Serv. Agents WWW **8**(4), 255–270 (2010)
27. Pichler, R., Skritek, S.: Containment and Equivalence of Well-designed SPARQL, pp. 39–50 (2014)
28. Reutter, J.L.: Containment of nested regular expressions. CoRR, abs/1304.2637 (2013)
29. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. ACM **27**(4), 633–655 (1980)
30. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of XPath 2.0. J. ACM (JACM) **56**(6), 31 (2009)
31. Vardi, M.Y., Murano, A., Lutz, C., Bonatti, P.A.: The complexity of enriched mu-calculi. Log. Methods Comput. Sci. **4**, 1–27 (2008)