Containment of Simple Conjunctive Regular Path Queries

Diego Figueira\textsuperscript{1}, Adwait Godbole\textsuperscript{2}, S. Krishna\textsuperscript{2}, Wim Martens\textsuperscript{2}, Matthias Niewerth\textsuperscript{3}, Tina Trautner\textsuperscript{3}

\textsuperscript{1}Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France
\textsuperscript{2}IIT Bombay, Mumbai, India
\textsuperscript{3}University of Bayreuth, Bayreuth, Germany

Abstract

Testing containment of queries is a fundamental reasoning task in knowledge representation. We study here the containment problem for Conjunctive Regular Path Queries (CRPQs), a navigational query language extensively used in ontology and graph database querying. While it is known that containment of CRPQs is ExpSpace-complete in general, we focus here on severely restricted fragments, which are known to be highly relevant in practice according to several recent studies. We obtain a detailed overview of the complexity of the containment problem, depending on the features used in the regular expressions of the queries, with completeness results for NP, \(\Pi_2^p\), PSPACE or ExpSpace.

1 Introduction

Querying knowledge bases is one of the most important and fundamental tasks in knowledge representation. Although much of the work on querying knowledge bases is focused on conjunctive queries, there is often the need to use a simple form of recursion, such as the one provided by regular path queries (RPQ), which ask for paths defined by a given regular language. Conjunctive RPQs (CRPQs) can then be understood as the generalization of conjunctive queries with this form of recursion. CRPQs are part of SPARQL, the W3C standard for querying RDF data, including well known knowledge bases such as DBpedia and Wikidata. In particular, RPQs are quite popular for querying Wikidata. They are used in over 24% of the queries (and over 38% of the unique queries), according to recent studies (Malyshev et al. 2018; Bonifati, Martens, and Timm 2019). More generally, CRPQs are basic building blocks for querying graph-structured databases (Barceló 2013).

As knowledge bases become larger, reasoning about queries (e.g. for optimization) becomes increasingly important. One of the most basic reasoning tasks is that of query containment: is every result of query \(Q_1\) also returned by \(Q_2\)? This can be a means for query optimization, as it may allow to avoid evaluating parts of a query, or reduce and simplify the query with an equivalent one. Furthermore, query containment has proven useful in knowledge base verification, information integration, integrity checking, and cooperative answering (Calvanese et al. 2000).

The containment problem for CRPQs is ExpSpace-complete, as was shown by (Calvanese et al. 2000) in a now ‘classical’ KR paper, which appeared 20 years ago. However, the lower bound construction of Calvanese et al. makes use of CRPQs which have a simple shape (if seen as a graph of atoms) but contain rather involved regular expressions, which do not correspond to RPQs how they typically occur in practice. Indeed, the analyses of (Bonifati, Martens, and Timm 2019; Bonifati, Martens, and Timm 2020) reveal that a large majority of regular expressions of queries used in practice are of a very simple form. This motivates us to revisit CRPQ containment on queries, focusing on commonly used kinds of regular expressions. Our goal is to identify restricted fragments of CRPQs that are common in practice and which have a reasonable complexity for query containment.

Contribution. According to recent studies on query logs, investigating over 500 million SPARQL queries (Bonifati, Martens, and Timm 2019; Bonifati, Martens, and Timm 2020), it turns out that a large majority of regular expressions that are used for graph navigation are of rather simple forms, like \(a^*\), \(ab^*\), \((a+b)c^*\), \(a(b+c)^*d\), i.e., concatenations of (disjunctions of) single symbols and Kleene stars of (disjunctions of) single symbols. Since CRPQs have concatenations built-in, CRPQs with such expressions are essentially CRPQs in which every atom has a regular expression of the form \((a_1 + \cdots + a_n)\) or \((a_1 + \cdots + a_n)^*\) for \(n \geq 1\). In the remainder of the paper, we often abbreviate the former type of atom with \(A\) and the latter by \(A^*\). If \(n = 1\), we write \(a\) and \(a^*\). Table 1 gives an overview of the frequency of such expressions in the following data sets:

(a) The data set studied by (Bielefeldt, Gonsior, and Krötzsch 2018; Bonifati, Martens, and Timm 2019), which was released by (Malyshev et al. 2018) and contains 208 million parseable Wikidata queries, with over 55 million regular path queries.

(b) The data set of (Bonifati, Martens, and Timm 2020), which contains 339 million parseable queries, mostly from DBpedia, but also from LinkedGeoData, BioPortal, OpenBioMed, Semantic Web Dog Food and the British Museum. These queries contain around 1.5 million regular path queries.\footnote{One sees that regular path queries are much more common in the Wikidata log than in the (mainly) DBpedia log. The reason for this is that the graph structure of DBpedia was designed before}
When we list multiple types of atoms in the table, we allow concatenations of these types. So, \(a(b+c)^*d\) is of type \(A, A^*\) and also of the more general type \(A, A^*\). In contrast to the types listed, RPs that are merely concatenations of single symbols, e.g., \(abc\) or \(aa\), represent only about 25% of the valid Wikidata expressions and 7% of the valid DBpedia expressions in the table.

Another motivation to study CRPQs with atoms of the forms \(a, a^*, A, A^*\) is that these are currently the only expressible atoms in CRPQs in Cypher 9 (Francis et al. 2018, Figure 3), a popular query language for property graphs.

We study the complexity of CRPQ containment for such fragments \(F\) of “simple CRPQs”, that is, CRPQs that only use atoms of some of the types \(a, a^*, A, A^*\). For each fragment \(F\), we provide a complete picture of the complexities of containment problems of the form \(F \subseteq F\), \(F \subseteq \text{CRPQ}\), and \(\text{CRPQ} \subseteq F\) (cf. Table 2, which we discuss in Section 3 in detail). The main take-aways are:

1. Even for such simple CRPQs, containment of the form \(F \subseteq F\) can become EXPSPACE-complete. Moreover, this lower bound already holds for containment of CRPQs using only \(a\)-atoms and \(A^*\)-atoms. This was surprising to us, because such CRPQs seem at first sight to be only mild extensions of conjunctive queries: they extend conjunctive queries only with atoms of the form \((a_1 + \cdots + a_n)^*\), i.e., Kleene closures over sets of symbols. The contrast between NP-completeness of containment for conjunctive queries and EXPSPACE-completeness for CRPQs that additionally allow \((a_1 + \cdots + a_n)^*\) is quite striking.

2. As soon as we disallow disjunction within Kleene closures in \(F\), the complexity of the abovementioned containment problems drops drastically to \(nP^\_\text{pspace}\).

The good news is that such regular expressions are still extremely common in practice, e.g., over 98% of the RPs in the Wikidata query logs (Table 1).

Due to the page limit, we can only provide sketches of some of the proofs. A version with more extensive proofs is available at https://arxiv.org/abs/2003.04411 (Figueira et al. 2020).

**Organization**

In Section 2 we introduce the necessary notation. In Section 3 we present our main results which are then proved in Sections 4–7. We discuss related work in detail in Section 8 and we conclude in Section 9.

## 2 Preliminaries

Let \(\Sigma\) be an infinite set of labels, to which we sometimes also refer as the alphabet. We abstract knowledge bases (or KBs, knowledge graphs, or graph databases) as finite, edge-labeled directed graphs \(K = (V, E)\), where \(V\) is a finite nonempty set of nodes, and \(E\) is a set of labeled directed edges \((u, a, v) \in V \times \Sigma \times V\). A path is a (possibly empty) sequence \(\pi = (v_0, a_1, v_1) \cdots (v_{n-1}, a_n, v_n)\) of edges; we say that \(\pi\) is a path from \(v_0\) to \(v_n\). The length of \(\pi\) is the number \(n \geq 0\) of edges in the sequence. We denote by \(lab(\pi)\) the word \(a_1 \cdots a_n\) of edge labels seen along the path. RPQs (property paths) existed in SPARQL.

If all edges of \(\pi\) have the same label \(a \in \Sigma\), we say \(\pi\) is an a-path. By \(\varepsilon\) we denote the empty word. Regular expressions are defined as usual. We use uppercase letters \(R\) for regular expressions and denote their language by \(L(R)\).

**Conjunctive regular path query (CRPQ)** has the general form \(Q(x_1, \ldots, x_n) \leftarrow A_1 \land \cdots \land A_m\). The atoms \(A_1, \ldots, A_m\) are of the form \(yRz\), where \(y\) and \(z\) are variables and \(R\) is a regular expression. Each distinguished variable \(x_j\) from the left hand side has to occur in some atom on the right hand side. A homomorphism from \(Q\) to \(K\) is a mapping \(\mu\) from the variables of \(Q\) to \(V\). Such a homomorphism satisfies an atom \(xRy\) if there is a path from \(\mu(x)\) to \(\mu(y)\) in \(K\) which is labeled with a word in \(L(R)\). A homomorphism from \(Q\) to \(K\) is called a satisfying homomorphism if it satisfies each atom \(A_i\). For brevity, we also use the term embedding for satisfying homomorphisms. The set of answers \(ans(Q, K)\) of a CRPQ \(Q\) over a knowledge base \(K\) is the set of tuples \((d_1, \ldots, d_n)\) of nodes of \(K\) such that there exists a satisfying homomorphism for \(Q\) on \(K\) that maps \(x_i\) to \(d_i\) for every \(1 \leq i \leq n\).

Given two CRPQs \(Q_1, Q_2\), we say that \(Q_1\) is contained in \(Q_2\), denoted by \(Q_1 \subseteq Q_2\), if \(ans(Q_1, K) \subseteq ans(Q_2, K)\) for every knowledge base \(K\). We say \(Q_1\) is equivalent to \(Q_2\), denoted by \(Q_1 \equiv Q_2\), if \(Q_1 \subseteq Q_2\) and \(Q_2 \subseteq Q_1\). We study the following problem, for various fragments \(F_1, F_2\) of CRPQ:

### Example

To illustrate query containment we consider the following example. Let \(Q_1(x_1, x_2) \leftarrow (x_1 \text{ app jm} \_1) \land (x_2 \text{ app jm} \_1) \land (\text{jm} \_1 \text{ app jm} \_2)\). Query \(Q_1\) returns \((x_1, x_2)\) only if they were both the apprentices of \(\text{jm}_1\) (a Jedi master) who was in turn an apprentice of \(\text{jm}_2\). Now consider \(Q_2(x_1, x_2) \leftarrow (x_1 \text{ app jm} \_1) \land (x_2 \text{ app jm} \_1)\). We see that \(Q_1 \equiv Q_2\). However if we remove the last atom from \(Q_1\), \(Q_1 \subseteq Q_2\) is not necessarily true. The following database provides a counterexample. \(Q_1\) without the last atom returns (LUKE, OBI-WAN) though \(Q_2\) does not.

Let \(Q\) be the CRPQ \(Q(x_1, \ldots, x_n) \leftarrow y_1 R_1 y_2 \land \cdots \land y_{2m-1} R_m y_{2m}\). Let \(K\) be a knowledge base and \(\nu\) a total mapping from the variables \(\{x_1, \ldots, x_n, y_1, \ldots, y_{2m}\}\) of \(Q\) to the nodes of \(K\). Then \(K\) is \(\nu\)-canonical for \(Q\) if

- \(K\) consists of \(m^2\) simple paths, one for each atom of \(Q\), which are node- and edge-disjoint except for the start and end nodes, and
- for each \(i \in \{1, \ldots, m\}\) the simple path \(\pi_i\) associated to the atom \(y_{2i-1} R_i y_{2i}\) connects the node \(\nu(y_{2i-1})\) to the node \(\nu(y_{2i})\) and has \(lab(\pi_i) \in L(R_i)\).

It is easy to see that \(Q_1 \not\subseteq Q_2\) iff there exists a knowledge base \(K\) and a mapping \(\nu\) from the variables of \(Q_1\) to the
But this does not make a difference: in CRPQs such concatenations on all valid queries after duplicate elimination (Unique). We provide Table 2: Complexity of Containment of different fragments $F$ of CRPQs. All results are complete for the class given. We provide Table 1: Percentage of simple RPQs and 2RPQs in the Wikidata query logs in the study (Bonifati, Martens, and Timm 2019) (left) and the diverse query logs of (Bonifati, Martens, and Timm 2020) (right). For every analysis, we show percentages on all valid queries (Valid) and on all valid queries after duplicate elimination (Unique).

$n$ nodes of $K$ such that (i) $K$ is $\nu$-canonical for $Q_1$ and (ii) $\nu(x_1), \ldots, \nu(x_n) \notin \text{ans}(Q_2, K)$. Therefore, to decide Containment, it suffices to study containment on knowledge bases which are $\nu$-canonical for $Q_1$. We call these knowledge bases canonical models of $Q_1$.

It is well-known that there is a natural correspondence between (the bodies of) CRPQs and graphs by viewing their variables as nodes and the atoms as edges. We will therefore sometimes use terminology from graphs for CRPQs (e.g., connected components).

### 3 Main Results

For a class of regular languages $L$ we write CRPQ($L$) to denote the set of CRPQs whose languages (of regular expressions in atoms) are in $L$. We use the same abbreviations for $L$ as discussed in the Introduction: $a$ for regular expressions that are just a single symbol, $a^*$ for Kleene closures of a single symbols, $A$ for disjunctions (or sets) of symbols, and $A^*$ for Kleene closures of disjunctions (or sets) of symbols. A sequence of abbreviations in $L$ represents options: for instance, CRPQ($a, A^*$) is the set of CRPQs in which each atom uses either a single symbol or a transitive closure of a disjunction of symbols.\(^2\)

In this paper, we give a complete overview of the complexity of containment for the fragments $F = \text{CRPQ}(a)$, CRPQ($A$), CRPQ($a, a^*$), CRPQ($A, a^*$), CRPQ($a^*$), and CRPQ($A, A^*$). That is, for each of these fragments we prove that their containment problem is complete for NP, $\Pi^P_2$, or EXPSPACE. Furthermore, for each of these fragments $F$, we give a complete overview of the complexity of the containment problems of the form $F \subseteq \text{CRPQ}$ and $\text{CRPQ} \subseteq F$. An overview of our results can be found in Table 2. All results are completeness results. Some of the results were already obtained in other papers, which we indicate in the table.

Interestingly, our results imply that containment is EXPSPACE-complete only if we allow sets of symbols under the Kleene star both in the left- and right-hand queries. As soon as we further restrict the usage of the Kleene star on one side, the complexity drops to PSPACE or even $\Pi^P_2$. As it turns out, queries having $a^*$ as only means of recursion is still very representative of the queries performed in practice, as evidenced in Table 1, where over 98% of the RPQs in the Wikidata logs are of this form. In the DBpedia\(^+\) logs, this percentage is still around 70% of the total RPQs. Two main reasons why this percentage is lower here are that "wildcards" of the form $a^*$, i.e., follow an edge not labeled $a$, and 2RPQs of the form $(a + a^*)^*$, i.e., undirected reachability over $a$-edges, make up around 15% and 20% respectively of the expressions in unique queries in DBpedia\(^+\). The fact that equivalence testing is $\Pi^P_2$ for these queries, gives hope that optimizations by means of static analysis may be practically feasible for most of the CRPQ used for querying ontologies and RDF data.

Our results apply to both finite and infinite sets of labels, if we do not explicitly say otherwise. The reason is that as long as the query language does not allow for wildcards, we can always restrict to the symbols explicitly used in the queries, which is always a finite set.

If wildcards are allowed, the complexity of query containment can heavily depend on the finiteness of the alphabet of edge labels $\Sigma$. We discovered that our techniques can be used to settle an open question (and correct an error) in the work of Deutsch and Tannen (2002), who have also considered containment of simple CRPQs. Deutsch and Tannen considered CRPQ fragments motivated by the navigational features of XPath and claimed that containment for their $W$-fragment (see Section 7 for a definition), using infinite al-

| Table 1: Percentage of simple RPQs and 2RPQs in the Wikidata query logs in the study (Bonifati, Martens, and Timm 2019) (left) and the diverse query logs of (Bonifati, Martens, and Timm 2020) (right). For every analysis, we show percentages on all valid queries (Valid) and on all valid queries after duplicate elimination (Unique). |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| RPQ Class       | Valid %         | Unique %        | Valid %         | Unique %        |
| $A, A^*$        | 99.02%          | 98.73%          | 99.83%          | 99.83%          |
| $A, a^*$        | 98.40%          | 98.31%          | 99.22%          | 99.44%          |
| $a, A^*$        | 93.50%          | 95.99%          | 94.30%          | 97.10%          |
| $a, a^*$        | 92.88%          | 95.58%          | 93.69%          | 96.69%          |
| Total           | 55,333K         | 14,189K         | 55,333K         | 14,189K         |

| Table 2: Complexity of Containment of different fragments $F$ of CRPQs. All results are complete for the class given. We provide references in round brackets. When there is no bracket, the result follows directly from another cell in the table. $(*)$: (Chandra and Merlin 1977), $(?)$: (Deutsch and Tannen 2002, fragment $(l^*)$), $(?)$: (Calvanese et al. 2000) |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| $F$             | $F \subseteq F$| $F \subseteq \text{CRPQ}$| $\text{CRPQ} \subseteq F$ |
| $a$             | NP $(?)$        | NP (4.2)        | $\Pi^P_2$ (4.4) |
| $A$             | $\Pi^P_2$ (4.3)| $\Pi^P_2$ (5.2)| $\text{PSPACE}$ (5.3) |
| $(a, a^*)$      | $\Pi^P_2$ (7)  | $\Pi^P_2$ (5.2)| $\text{PSPACE}$ (5.5) |
| $(A, a^*)$      | EXPSPACE (6.1) | EXPSPACE (+)    | EXPSPACE (+)    |
| $(A, A^*)$      | EXPSPACE (+*)   | EXPSPACE (+)    | EXPSPACE (+)    |

\(^2\)In some proofs, we also allow concatenations of these forms. But this does not make a difference: in CRPQs such concatenations can always be eliminated at the cost of a few extra variables.

373
phabets, is PSPACE-hard. However, we prove that containment for this fragment is in \( \Pi_2^P \) (Theorem 7.1). The minor error is that Deutsch and Tannen assumed finite alphabets in their hardness proof. In fact, when one indeed assumes a finite set of edge labels in KBs, we prove that the containment problem for the W-fragment is EXPSPACE-complete (Proposition 7.2).

## 4 No Transitive Closure

In this section we study simple CRPQ fragments without transitive closure. We first observe that CRPQ(a) is equivalent to the well-studied class of conjunctive queries (CQ) on binary relations.

**Theorem 4.1** (Chandra and Merlin 1977). Containment of CRPQ(a) in CRPQ is NP-complete.

Even when we allow arbitrary queries on the right, the complexity stays the same. The reason is that the left query has a single canonical model \( K \) of linear size, and thus we can check containment by testing for a satisfying homomorphism from \( Q_2 \) to \( K \) (that preserves the distinguished nodes).

**Theorem 4.2.** Containment of CRPQ(a) in CRPQ is NP-complete.

If we allow more expressive queries on the left, the complexity becomes \( \Pi_2^P \), even if the right-hand queries are CQs.

**Theorem 4.3.** Containment of CRPQ(\( A \)) in CRPQ is \( \Pi_2^P \)-complete, even if the size of the alphabet is fixed.

*Proof sketch.* The upper bound is immediate from Corollary 5.2, which in turn follows from Theorem 7.1. Both these results are proved later. For the lower bound, we reduce from \( \forall \exists \)-QBF (i.e., \( \Pi_2 \)-Quantified Boolean Formulas). Let

\[
\Phi = \forall x_1, \ldots, x_n \exists y_1, \ldots, y_e \varphi(x_1, \ldots, x_n, y_1, \ldots, y_e)
\]

be an instance of \( \forall \exists \)-QBF such that \( \varphi \) is quantifier-free and in 3-CNF. We construct boolean queries \( Q_1 \) and \( Q_2 \) such that \( Q_1 \subseteq Q_2 \) if, and only if, \( \Phi \) is satisfiable.

The query \( Q_1 \) is defined in Figure 2, over the alphabet of labels \( \{a, x_1, \ldots, x_n, y_1, \ldots, y_e, t, f\} \). We now explain how we define \( Q_2 \) over the same alphabet. Every clause of \( \Phi \) is represented by a subquery in \( Q_2 \), as depicted in Figure 3. All nodes with identical label \( (y_{1,t}, y_{1,f}) \) in gadgets \( D, E \) in Figures 2 and 3 are the same node. (So, both queries are DAG-shaped.) Note that for every clause and every existentially quantified literal \( y_i \), wherein we have one node named \( y_{i,t,f} \) in \( Q_2 \). The \( E \)-gadget is designed such that every represented literal can be homomorphically embedded, while exactly one literal has to be embedded in the \( D \)-gadget.

The intuitive idea is that the valuation of the \( x \)-variables is given by the concrete canonical model \( K \) (i.e., whether the corresponding edge is labeled \( t \) or \( f \) in the \( D \) gadget), while the valuation of the \( y \)-variables is given by the embedding of \( Q_2 \) into \( K \) (i.e., whether the corresponding node is embedded into the node \( y_{\_t} \) or \( y_{\_f} \)). The embedding of \( y \)-variables across several clauses has to be consistent, as all clauses share the same nodes \( y_{\_t,f} \), which uniquely get embedded either into \( y_{\_t} \) or \( y_{\_f} \). Hence, when the formula \( \Phi \) is satisfiable, for any assignment to the variables \( \{x_i\} \) (given by the choice of \( t/f \) edges in \( D \)), there is a mapping from \( y_{\_t,f} \) to one of \( y_{\_t} \) or \( y_{\_f} \). This gives \( Q_1 \subseteq Q_2 \). Conversely, if \( Q_2 \) can be embedded in \( K \), then, for a choice of \( t/f \) edges in \( D \), we have an embedding of each clause gadget of \( Q_2 \) in \( K \). In particular, we can always map a literal in each clause of \( Q_2 \) to \( D \), ensuring that \( \varphi \) is satisfied. As this is true for any knowledge base \( K \) obtained for all possible \( t/f \) assignments to \( \{x_i\} \), we obtain \( \Phi \) is satisfiable.

We note that this result can be extended to alphabets of constant size by encoding \( x_i \) as \( x_i = \hat{\delta}^{-1} \hat{\delta}^{j-1} \hat{\delta}^{i-1} \in \{\hat{\delta}, \hat{\delta}^2\}^n \) and \( y_i = \hat{\alpha}^{i-1} \hat{\alpha}^{\delta} \hat{\alpha}^{\delta-j} \in \{\hat{\alpha}, \hat{\alpha}^2\}^f \).

On the other hand, even if we now allow arbitrary CRPQs on the left, containment remains in \( \Pi_2^P \).

**Theorem 4.4.** Containment of CRPQ in CRPQ(a) is \( \Pi_2^P \)-complete.

*Proof.* The lower bound is immediate from Theorem 4.3. For the upper bound, we provide a \( \Sigma_2^P \) algorithm for non-containment, which yields the result. Let \( Q_1 \in \text{CRPQ} \), \( Q_2 \in \text{CRPQ}(a) \), and \( \# \) be a symbol not appearing in \( Q_1 \) or \( Q_2 \). For every atom \( A = xRy \) of \( Q_1 \) we guess words \( u_A \) and \( v_A \) of length \( \leq |Q_2| \) such that \( u_A v_A^\# \cap L(R) \neq \emptyset \) and \( |u_A v_A| < 2|Q_2| \) implies that \( u_A v_A \notin L(R) \). We guess a component \( Q'_2 \) of \( Q_2 \) and we check that
(1) $Q'_2$ cannot be embedded in $Q'_1$, where $Q'_2$ is the KB resulting from replacing each atom $A = xRy$ with the path $u_A \cdot s_# \cdot v_A$, where $s_# = \varepsilon$ if $|u_A v_A| < 2|Q_2|$ and $s_# \neq \#$ otherwise; and

(2) for every atom $A = xRy$ of $Q_1$ such that $|u_A v_A| = 2|Q_2|$ there is $w \in u_A \Sigma^*v_A \cap L(R)$ such that $Q'_2$ cannot be embedded in $w$. This last test amounts to checking that either (i) $Q'_2$ is not homomorphically equivalent to a path or, otherwise, (ii) if $Q'_2$ is homomorphically equivalent to a path with label $w$, we test $u_A \Sigma^*v_A \cap L(R) \cap (\Sigma^* w \Sigma^*)^c \neq \emptyset$.

If tests (1) and (2) succeed, we found a knowledge base into which $Q_1$ can be embedded, but not $Q_2$. Testing whether $Q'_2$ can be homomorphically embedded in $Q'_1$ is in NP as the size of $Q'_1$ is polynomial in $Q_1$ and $Q_2$. Test (2) is in \text{CONP} as we need to check for an embedding of $Q'_2$ for each atom of $Q_1$.

Allowing disjunctions in the right query is rather harmless if we only need to consider polynomial-size canonical models to decide containment correctly. Even if such canonical models may become exponentially large, they can sometimes be encoded using polynomial size, allowing for \text{PI}_2^0 \text{ containment algorithms} (cf. Corollary 5.2, Theorem 7.1). However, if we have arbitrary queries on the left, these techniques do not work anymore, to the extent that the problem becomes \text{PSpace}-complete.

The following theorem can be regarded as a generalization of the result of Björklund, Martens, and Schwentick (2013) [Theorem 9] stating that the inclusion problem between a DFA over an alphabet $\Sigma = \{a, b, c\}$ and a regular expression of the form $\Sigma^*a\Sigma^*b\Sigma^*$ is \text{PSPACE}-complete.

**Theorem 4.5. Containment of CRPQ in CRPQ($A$) is \text{PSPACE}-complete, even if the size of the alphabet is fixed.**

**Proof.** The upper bound follows from Theorem 5.5, which we prove later. For the lower bound we reduce from the corridor tiling problem, a well-known \text{PSPACE}-complete problem (Chlebus 1986). An instance of this problem is a tuple $(T, H, V, \bar{i}, \bar{f}, n)$, where $T$ is the set of tiles, $H, V \subseteq T \times T$ are the horizontal and vertical constraints, encoding which tiles are allowed to occur next to each other and on top of each other, respectively, $\bar{i} = i_1 \ldots i_n \in T^n$ is the initial row, $\bar{f} = f_1 \ldots f_n \in T^n$ is the final row, and $n$ encodes the length of each row in unary. The question is whether there exists a tiling solution, that is, an $N \in \mathbb{N}$ and a function $\tau : \{1, \ldots, N\} \times \{1, \ldots, n\} \rightarrow T$ such that $\tau(1, 1) \cdots \tau(1, n) = i$, $\tau(N, 1) \cdots \tau(N, n) = f$ and all horizontal and vertical constraints are satisfied: $(\tau(i, j), \tau(i, j + 1)) \in H$ and $(\tau(i, j), \tau(i + 1, j)) \in V$ for every $i, j$ in range.

The coding idea is that the query $Q_1$ is a string describing all tilings with correct start and end tiles, with no horizontal errors, and having rows of the correct length. The query $Q_2$ describes vertical errors. Then we have $Q_1 \subseteq Q_2$ if and only if there exists no valid tiling, i.e., every tiling has an error.

Let $(T, H, V, i, \bar{f}, \bar{n})$ be a corridor tiling instance as defined before. From the original proof of Chlebus (1986), it follows that the following restricted version of corridor tiling remains \text{PSPACE}-complete. The set of tiles $T$ is partitioned into $T = T_1 \uplus T_2 \uplus T_3$, such that each row in a solution must belong to $T_1 T_2 T_3 \uplus T_2 T_3 T_1$. The original proof furthermore implies that (i) $(T_1 \times T_1) \cup (T_1 \times T_2) \cup (T_2 \times T_1) \subseteq H$; and (ii) for all $u, v \in T_3$ with $(u, v) \in H$ we have that $T_3 \times \{v\} \subseteq H$ and $\{v\} \times T_1 \subseteq H$. This implies that our horizontal errors can only occur with $T_2$ or $T_3$ involved, so only once per row. Therefore, we construct a new set $H$ defined as follows: $H = H \cap (T_2 \times T_1 \cup T_1 \times T_3 \cup T_3 \times T_2 \cup T_3 \times T_3)$. This set is used in the definition of query $Q_1$.

We encode tiles as follows: each tile $t_i$ has an encoding $t_i$ given by $\Delta^i a_i\Delta^{|t_i|-i-1} e_1 \cdots e_{|t_i|}$, where $e_j = \psi$, if $(t_i, t_j) \in V$ and $e_j = \phi$, otherwise. The second half of the encoding of a tile describes which tiles are allowed to occur above the tile. The query $Q_1$ is

$$\hat{t}_1 \cdots \hat{t}_n \left( \sum_{i=0}^{n-2} \sum_{(v_1, v_2) \in H} (\hat{t}_1)^i \hat{v}_1 \hat{v}_2 (\hat{t}_1)^{n-i-2} \right)^* \hat{f}_1 \cdots \hat{f}_n.$$  

We note that $Q_1$ encodes exactly the tilings without horizontal errors, due to the imposed restrictions.

The query $Q_2$ is $\Delta(\Delta + \psi + \Delta + \phi)(2n-1)|T|-1\phi$ and matches exactly those positions where a vertical error occurs, exploiting the encoding of vertical constraints in the second half of each tile’s encoding.

**5 Simple Transitive Closures**

In this section, we investigate what happens if we consider fragments that only allow singleton transitive closures, that is, transitive closures of single symbols. Our first results imply a number of $\text{PI}_2^0$-results in Table 2.

**Theorem 5.1. Containment of CRPQ($a, a^*$) in CRPQ($a$) is $\text{PI}_2^0$-hard, even if the size of the alphabet is fixed.**

**Proof sketch.** We use a similar reduction as in Theorem 4.3. The only change we make is that we replace the expressions $t \leftarrow f$ in $Q_1$ with $t^* f$-paths. Intuitively, $Q_1$ sets a variable $x_i$ to true if and only if there exists at least one $t$-edge after the $x_i$-edge. The query $Q_2$ is not changed.

**Corollary 5.2. Containment of CRPQ($A, a^*$) in CRPQ($a$) is in $\text{PI}_2^0$.**

**Proof.** This will be a corollary of Theorem 7.1, since CRPQ($A, a^*$) is a fragment of CRPQ($W$).

On the other hand, if we allow arbitrary queries on the left and simple transitive closure on the right-hand query, the problem becomes \text{PSPACE}-hard.

**Theorem 5.3. Containment of CRPQ in CRPQ($a, a^*$) is \text{PSPACE}-complete, even if the size of the alphabet is fixed.**

**Proof sketch.** We adapt the encoding in the proof of Theorem 4.5, by (a) replacing each symbol $\sigma \in \{\phi, \psi, \phi, \psi\}$ with $\sigma \$ where $\$ is a new symbol, and (b) replacing $Q_2$ with $\Delta \$($\Delta^* \phi^* \Delta^* \psi^*$)$^{2K-1}|T|-1 \phi$.

Interestingly, the complexity of containment can drop by adding distinguished variables to the query:
**Proposition 5.4.** The complexity of Containment of (1) CRPQ in CRPQ(A) and (2) CRPQ in CRPQ(A, a*) is in \(\Pi^p_2\) if every component of each query contains at least one distinguished variable.

Finally we show that, as long as the right query only has single symbols under Kleene closures, query containment remains PSPACE-complete.

**Theorem 5.5.** Containment of CRPQ in CRPQ(A, a*) is PSPACE-complete.

**Proof.** The lower bound is immediate from Theorem 4.5. For the upper bound we provide a PSPACE-algorithm for non-containment. Let \(Q_1 \in \text{CRPQ}_1\), \(Q_2 \in \text{CRPQ}(A, a^*)\), and \# be a symbol not appearing in \(Q_1\) and \(Q_2\). We first note that each component of \(Q_2\) can express at most \(|Q_2|\) many label changes on a path. Hence it suffices if the algorithm stores just the part of a path that corresponds to the last \(|Q_2|\) label changes. Furthermore, a standard pumping argument yields that, in a counterexample, the length of segments that only use a single label can be limited to \(|Q_1| + |Q_2|\).

Therefore, for each atom of \(A = xRY\) of \(Q_1\), the PSPACE-algorithm guesses words \(u_A \cdot v_A\) of length at most \(|Q_2| \times (|Q_1| + |Q_2|)\), such that \(u_A \Sigma^* v_A \cap L(R) \neq \emptyset\) and, if \(u_A\) or \(v_A\) has less than \(|Q_2|\) label changes, then \(u_A v_A \in L\). We guess a component of \(Q_2\) and check that

1. \(Q_2\) cannot be embedded in \(Q_1\), where \(Q_1\) is the KB resulting from replacing each atom \(x\) by \(xRY\) with the path \(u_A \cdot s_{\#} \cdot v_A\), where \(s_{\#} = \varepsilon\) if \(u_A\) or \(v_A\) contains less than \(|Q_2|\) label changes and \(s_{\#} = \#\) otherwise; and
2. for every atom \(A = xRY\) of \(Q_1\) such that \(u_A\) and \(v_A\) have \(|Q_2|\) label changes there is \(w \in u_A \Sigma^* v_A \cap L(R)\) such that \(Q_2\) cannot be embedded in \(w\).

If tests (1) and (2) succeed, we found a knowledge base into which \(Q_1\) can be embedded, but \(Q_2\) cannot. Test (1) is in \(\text{coNP}\) as \(Q_1\) has size polynomial in \(Q_1\) and \(Q_2\). Test (2) is in polynomial space, as the restricted language of \(Q_2\) allows us to guess and verify the existence of \(w\) on the fly while only keeping the path corresponding to the last \(|Q_2|\) label changes in memory with length at most \(|Q_2| \times (|Q_1| + |Q_2|)\).

**6 Transitive Closures of Sets**

In this section we show that adding just a little more expressiveness makes containment EXPSPACE-complete. This high complexity may be surprising, considering that it already holds for CRPQ(a, a*) queries, which is a fragment that merely extends ordinary conjunctive queries by adding transitive reflexive closures of simple disjunctions. Our proof is inspired by the hardness proof in (Calvanese et al. 2000) for general CRPQs, but we need to add a number of non-trivial new ideas to make it work for CRPQ(a, a*).

**Disjunction creation.** A significant restriction that is imposed on CRPQ(a, a*) is that the non-transitive atoms are not allowed to have disjunctions in their expressions. We get around this by the following idea that generates disjunctive bad patterns out of conjunctions — we use a similar idea in our next proof.

Consider the following query \(Q_2\) where \(\ell\) is a special helper symbol, \(y_1 \ell \cdot s_3 \cdot \ell \cdot s_2 \ell \cdot y_2\). For query \(Q_1\) given by

\[
\bigland_{\ell \in \Sigma} \exists x_1 \exists x_1 \exists x_1 \exists x_1 \cdot \ell \cdot \bigland_{\ell \in \Sigma} \exists x_2 \exists x_2 \exists x_2 \exists x_2 \cdot \ell \cdot \exists x_2 \bigland_{\ell \in \Sigma} \exists x_2 \exists x_2 \exists x_2 \exists x_2 \cdot \ell
\]

it is clear that \(Q_1\) allows for exactly two \(\ell\), and hence, if \(Q_1\) were to be contained in \(Q_2\), one of the patterns \(s_1\) or \(s_2\) has to be matched to the \((\Sigma \setminus \{\ell\})^+\) fragment in the middle. Essentially, we capture all bad patterns matching either \(s_1\) or \(s_2\), thereby "creating" the result of a disjunction.

**Theorem 6.1.** Containment of CRPQ(a, a*) in CRPQ(a, a*) is EXPSPACE-hard, even if the size of the alphabet is fixed.

**Proof sketch.** We reduce from the exponential width corridor tiling problem. That is, we have

- a finite set \(T = \{t_1, \ldots, t_m\}\) of tiles,
- initial and final tiles \(t_i, t_F \in T\), respectively,
- horizontal and vertical constraints \(H, V \subseteq T \times T\), and
- a number \(n \in \mathbb{N}\) (in unary),

and we want to check if there is a \(k \in \mathbb{N}\) and a tiling function \(\tau: \{1, \ldots, k\} \times \{1, \ldots, 2^n\} \rightarrow T\) such that \(\tau(1, 1) = t_1\), \(\tau(k, 2^n) = t_F\), and all horizontal and vertical constraints are satisfied. In order to have a fixed alphabet, we encode tiles from \(T\) as words from \(\{\emptyset, \bullet\}\)^\(m\). The \(i\)-th tile \(t_i\) is encoded as \(t_i^* = \emptyset^{i-1}\emptyset^{m-1}\). A tiling \(\tau\) is encoded as a string over the alphabet \(\mathbb{B} = \{\$, 0, 1, \emptyset, \bullet, \#\}\), where \$ is the row separator, 0 and 1 are used to encode addresses for each row of the tiling from 0 to \(2^n - 1\) as binary numbers, \# separates the individual bits of an address, and \$ and \# are used to encode the individual tiles. We visualize a tiling as a matrix with \(k\) rows of \(2^n\) tiles each.

An example of a tiling \(\tau\) with \(n = 3\) is below:

\[
\begin{array}{cccc}
\tau(1, 1)000\#00\tau(2, 2)000\#1 \cdots \tau(k, 2^3)1\#1\#1 \\
\vdots & \vdots & \vdots & \vdots \\
\$\tau(1, 1)000\#00\tau(1, 2)000\#1 \cdots \tau(1, 2^3)1\#1\#1 \\
\end{array}
\]

The queries \(Q_1\) and \(Q_2\) use the alphabet \(\mathbb{A} = \mathbb{B} \cup \{\lceil\rceil, \langle\rangle, b, \ast\}\). This new set contains helper symbols \{ and \} which we use for disjunction creation (in a similar way as we explained before the Theorem statement), and \langle and \rangle denote the start and end of the tiling. The \(b\)-symbol is used for a special edge that we use for checking vertical errors. Query \(Q_1\) is given in Figure 4 and query \(Q_2\) is sketched in Figure 5. For convenience we use \(\mathbb{B}_{\emptyset}\) to abbreviate \(\mathbb{B} \cup \{\langle\rangle\}\) and \(\mathbb{B}_{\langle\rangle}\) to abbreviate \(\mathbb{B} \cup \{\lceil\rceil\}\), and \(\mathbb{B}_{\#}\) to abbreviate \(\mathbb{B} \setminus \{\$\}\).

The intuition is that the tiling is encoded in the \(\mathbb{B}_{\ast}\)-edge of \(Q_1\), i.e. the only edge that is labeled by a language that is not a single symbol. The query \(Q_2\) consists of a sequence of bad patterns, one for each possible kind of violation of the described encoding or the horizontal and vertical constraints. The queries are designed in such a way that \(Q_2\) cannot be embedded if a valid tiling is encoded in a canonical model of \(Q_1\). Otherwise, at least one of the bad patterns can be embedded in the encoding of the tiling. The other bad patterns can be embedded at the nodes \(z_2\) and \(z_7\) of \(Q_1\), as these nodes have one self loop for every symbol of the alphabet except \(\ast\).
We can easily design (sets of) patterns, where each pattern is a simple path, to catch the following errors: malformed encoding of a tile, malformed encoding of an address, non-incrementing addresses, missing initial or final $\$, wrong initial or final tile, and an error in the horizontal constraints.

The most difficult condition to test is an error in the vertical constraints, which we encode with the pattern $G_i^{t',v}$ for every $(t',v) \notin V$, given by

$$\bigwedge_{1 \leq i \leq n} G_i^{t',v} \land \bigwedge_{i,j \in \{1,\ldots,n\}, c,d \in \{0,1\} : |i-j| = 1} (x_{i,c}^{t',v} \land y_{i,c}^{t',v} \land L_{y_{j,d}}^{t',v}),$$

where $G_i^{t,v}$ is given in Figure 6 and $L = b^* \emptyset_0^* b^*$. We first explain the intuition behind $G_i^{t,v}$. We assume that the vertical error occurs at tile $t$ having $0$ as $i$-th bit of its address.

In that case, the variable $x_{i,0}^{t,v}$ should be embedded just before the encoding of $t$, while $y_{i,0}^{t,v}$ should be embedded in the next row just after the tile $t$ with the same $i$-th bit. This is enforced as there is one $\$ between $x_{i,0}^{t,v}$ and $y_{i,0}^{t,v}$, ensuring that both variables occur in consecutive rows. The variables $x_{i,0}^{t,v}$ and $y_{i,0}^{t,v}$ are simply embedded at the node corresponding to $z_0$ of $Q_1$.

In the case that the $i$-th bit is $1$, we embed $x_{i,1}^{t,v}$ and $y_{i,1}^{t,v}$ at $z_3$, while $x_{i,1}^{t,v}$ and $y_{i,1}^{t,v}$ are embedded at the tiles violating the vertical constraint, as described in the previous case.

$$\langle \emptyset \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 B^* \rangle \langle \emptyset \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 \hat{e}^0 B^* \rangle$$

Figure 6: Subquery $G_i^{t,v}$ in the proof of Theorem 6.1. Here, $e_i^a = ((0,1)^* \#)^{-1} a (\# (0,1)^*)^{n-1-1}$ is the language enforcing the $i$-th bit to be $a$.

Altogether, $G_i^{t,v}$ verifies that there are positions $v$ and $w$ in consecutive rows of the encoding such that the tiles adjacent to $v$ and $w$ would violate the vertical constraints and the positions agree on the $i$-th bit of the address. To ensure that the positions $v$ and $w$ agree on all $n$ bits of the address we have to ensure that the $n$ patterns $G_1^{t,v}, \ldots, G_n^{t,v}$ all refer to the same two positions in the tiling. This is why we have the additional conjuncts with language $L = b^* \emptyset_0^* b^*$ in $G_i^{t,v}$. The language $L$ is chosen to ensure that there exists exactly one one node $v$ in the tiling such that all the variables $x_{j,0}^{v} \ldots, x_{n,0}^{v}$ for $j \in \{0,1\}$ are either embedded at $v$, at the node corresponding to $z_3$ from $Q_1$, or at the node corresponding to $z_6$ from $Q_1$. If there were two variables $x_{i,c}^{t,v}$ and $x_{i,d}^{t,v}$ embedded at different positions between $z_3$ and $z_6$ then there is a $k$ and $\tilde{\ell}, \tilde{d}$ such that $x_{k,\tilde{\ell}}^{t,v}$ and $x_{k+1,\tilde{d}}^{t,v}$ are embedded at different positions and thus at least one of the conjuncts $x_{k,\tilde{\ell}}^{t,v} \land x_{k+1,\tilde{d}}^{t,v}$ to be violated, as the symbol $b$ can be read only at the beginning or end of a string in $L$ (recall that $b \notin \emptyset_0^* \emptyset_0^* b^*$). The argument for the $y$-variables and the position $w$ is analogous.

To conclude, whenever there exists a valid tiling, we have a canonical knowledge base with the encoding of a tiling occurring between $z_3$ and $z_5$. To embed $Q_2$ into this, we need to span the full length flanked by the *’s in the start and the end. Thanks to (i) the symbols [;] flanking the bad patterns $B_i$ in $Q_2$, and (ii) the presence of these symbols only at edges from nodes $z_2, z_7$ in $Q_1$, at least one of the bad patterns must embed into the part between $z_3$ and $z_6$. If there is no error, we cannot embed $Q_2$, and hence no $B_i$ can be mapped between $z_3$ and $z_6$ and we have $Q_1 \not\subseteq Q_2$. On the other hand, when there is no valid tiling, for each canonical knowledge base with a ‘guessed’ tiling, $Q_2$ maps one of the $B_i$ between $z_3$ and $z_6$, and can hence embed completely from $*$ to $*$, giving $Q_1 \subseteq Q_2$.

\[\Box\]

Remark 6.2. We observe that the queries $Q_1$ and $Q_2$ in Theorem 6.1 have bounded treewidth. Treewidth is a commonly used parameter in parameterized complexity analysis and intuitively, captures how close the graph is to a tree. A tree has treewidth $1$, while $K_n$, the complete graph on $n$ vertices has treewidth $n-1$. It is known that the containment problem of CQs with bounded treewidth (as is the evaluation problem of CQs with bounded treewidth) is in PTIME (Chekuri and Rajaraman 2000). In this light, it is surprising how the complexity of containment increases to EXPSPACE already for CRPQ($\alpha, A^*$), even for queries of bounded treewidth.

7 Deutsch and Tannen’s W-Fragment

The complexity of containment of CRPQs with restricted regular expressions has also been investigated by Deutsch and Tannen (2002). Their work was motivated by the types of restrictions imposed on navigational expressions in the query language XPath. Interestingly, they left some questions open, such as the complexity of containment for CRPQs using expressions from their W-Fragment. The W-

--

\[\text{The nomenclature of this fragment is a mystery to us. Even Deutsch and Tannen say: “The fragments called W and Z have}\]
fragment is defined by the following grammar:

\[
R \rightarrow \sigma \mid \_ \mid S^* \mid R \cdot R \mid (R + R) \\
S \rightarrow \sigma \mid \_ \mid S \cdot S
\]

Here, \(\sigma \in \Sigma\) and \(\_\) is a wildcard, i.e., it matches a single, arbitrary symbol from the infinite set \(\Sigma\). In the RFPs underlying Table 1, wildcards occurred in 0% (40 out of 55M) property paths in Wikidata queries, but in ~4.30% of the property paths in valid and in 15.68% of the property paths in unique DBpedia queries. By CRPQ\((W)\), we denote CRPQs where the regular expressions are from the W-fragment.

Deutsch and Tannen (2002) claimed that containment for CRPQ\((W)\) is PSPACE-hard, but their proof, given in Appendix C of their article, has a minor error: it uses the assumption that \(\Sigma\), the set of edge labels, is finite. In fact, we show that containment of CRPQ\((W)\) queries is in \(\Pi^2_2\). Furthermore, the right query can even be relaxed completely.

**Theorem 7.1.** Containment of CRPQ\((W)\) in CRPQ is in \(\Pi^2_2\).

**Proof.** Let \(Q_1 \in \text{CRPQ}(W)\) and \(Q_2 \in \text{CRPQ}\). We first show a small model property. More precisely, we show that whenever there is a counterexample to the containment, then there also exists a canonical model \(B\) of \(Q_1\) such that \(B \notin Q_2\) and \(B\) can be represented by a polynomial size graph where each edge is either labeled with a single symbol or by \(w^i\), where \(w\) is of size linear in \(Q_1\) and \(i\) is at most \(2|Q_1|^3\).

Assume that \(B\) is the smallest graph that is a canonical model of \(Q_1\) and has no satisfying homomorphism from \(Q_2\). W.l.o.g., we assume that all occurrences of \(\_\) in \(Q_1\) are replaced by the same symbol \(\$\) that does not occur in \(Q_2\). As the W-fragments allows only a fixed string below every star, every path of \(B\) can be written as \(w^n_0 a_1 w^{l_1}_1 a_2 \cdots a_n w^{l_n}_n\), where \(n < |Q_1|\) and \(l_i \in \mathbb{N}\), as all long segments of a path have to result from applying the Kleene star to a fixed string.

It remains to show that for every path, all multiplicities are at most \(2|Q_1|^3\). We assume towards a contradiction that there exists a path \(p\) in \(B\), where for some string \(w\), the multiplicity \(\ell\) is larger than \(2|Q_1|^3\). We assume w.l.o.g. that all NFAs in \(Q_2\) share the same transition function \(\delta\) over the same set of states \(P\), which can be achieved by taking the disjoint union of all sets of states. Let \(M\) be the adjacency matrix of the transition relation for the string \(w\), i.e., \(M\) is a Boolean \(|P| \times |P|\) matrix, that has a 1 on position \((i, i')\), if and only if \(\delta(q_i, w) = q_{i'}\). By the pigeonhole principle, there have to be \(j\) and \(k\) such that \(0 \leq j < k \leq 2|P|^3\) and \(M^j = M^k\). We now shorten \(p\) by \(k - j\) copies of \(w\) and call the resulting graph \(B'\). It is obvious that \(Q_1\) can still embed into \(B'\). We have to show that \(Q_2\) cannot embed into \(B'\). Towards a contradiction we assume that \(h\) is a satisfying homomorphism from \(Q_2\) to \(B'\). Let \(p'\) be a subpath of the path \(p\) that spans at least \(j\) copies of \(w\) such that no node of \(p'\) occurs in the image of \(h\). Such a subpath exists due to the length of \(p\) still being at least \(2|P| + j\) and the fact that the sizes of \(|P|\) and the image of \(h\) are both bounded by \(2|Q_1|^3\). We now insert \(k - j\) copies of \(w\) into \(p'\). By definition of \(M\) and the fact that \(M^j = M^k\), we have that \(h\) is also a satisfying homomorphism from \(Q_2\) to \(B\), the desired contradiction.

We note that the minimal model property implies that the smallest counter examples can be stored using only polynomial space by storing the multiplicities of strings in binary. The \(\Pi^2_2\)-algorithm universally guesses such a polynomial size representation of a canonical model \(B\) of \(Q_1\). Then it tests whether there exists an homomorphism from \(Q_2\) into \(B\) by guessing an embedding. Testing whether a guessed mapping is indeed a satisfying homomorphism can be done in polynomial time using a square-and-multiply algorithm to compute any necessary \(\delta^i(q, w^i)\).

Next we show that, if we assume a finite set of edge labels \(\Gamma\) for knowledge graphs, the containment problem of CRPQ\((W)\) is not just PSPACE-hard (as Deutsch and Tannen showed), but even EXPSPACE-complete. The important technical difference with Theorem 7.1 is that, when the labeling alphabet \(\Gamma\) is finite, it is not always possible to replace occurrences of the wildcard \(\_\) with a fresh symbol that doesn’t appear in either query. Therefore, the counterexamples cannot be stored in a compact way. Even though this is a different setting than all the other results in the paper, we provide a proof, because the problem was left open by Deutsch and Tannen (2002).

**Proposition 7.2.** If edge labels of knowledge bases come from a finite alphabet \(\Gamma\), then containment of CRPQ\((W)\) in CRPQ\((W)\) is EXPSPACE-complete.

**Proof.** To avoid confusion with an infinite alphabet, we write \(\Gamma\) instead of \(\_\). We change the languages used in the proof of Theorem 6.1. We apply the following homomorphism \(h\) to all single label languages of \(Q_1\) and \(Q_2\) (including the languages resulting from the double-self-loops in Figure 4): \# \mapsto \varepsilon, \$ \mapsto \$\Delta, \sigma \mapsto \sigma\Delta\Delta\Delta\Delta\Delta\ for \sigma \in \mathbb{B}\ (\{\}\), and \(\sigma \mapsto \sigma\Delta\Delta \in \mathbb{A} \\mathbb{B}\), where \(\Delta\) and \(\Delta\) are new symbols, i.e., we encode every symbol \(\sigma\) of our original construction by the three symbols \(\sigma_1\sigma_2\), where \(\sigma_1, \sigma_2 \in \{\Delta, \Delta\}\) encode whether \(\sigma\) belongs to \(\mathbb{B}\) and \(\mathbb{A}\), respectively.

We replace every occurrence of \(\mathbb{B}\) with the language \((\text{IT}\Delta\Delta\Delta)^+\) and every occurrence of \(\mathbb{A}\) with the language \((\text{IT}\Delta\Delta\Delta)^+\). We replace \(e_i^j\) as used in Figure 6 with \(\{(0+1)\Delta\Delta\}^{i-1}a\Delta\Delta((0+1)\Delta\Delta)^{j-i-1}\).

The last change is that we add further bad patterns to the construction of \(Q_2\) that detects whenever the language \((\text{IT}\Delta\Delta\Delta)^+\) resulting from the \(\mathbb{B}\) \(\in Q_1\) produces an invalid pattern, i.e., a triple that is not in the image of \(h\).

---

**8 Related Work**

The most relevant work to us is that of Calvanese et al. (2000), who proved that containment for conjunctive regular path queries, with or without inverses, is EXPSPACE-complete, generalizing the EXPSPACE upper bound for CRPQs of Florescu, Levy, and Suciu (1998).

Deutsch and Tannen (2002) have also studied the containment problem for CRPQ with restricted classes of regular expressions. They chose fragments of regular expressions...
Figueira (2020) shows that containment of UC2RPQs is NP-complete if one of the sides is only a path or a DAG. The algorithm also works if both sides are tree-shaped. Their fragments Z and full CRPQs allow unions under Kleene star, but are already EXPSPACE-complete.

Florescu, Levy, andSuciu (1998) studied a fragment of conjunctive regular path queries with wildcards for which the containment problem is NP-complete—thus, it has the same complexity as containment for conjunctive queries. In their fragment, they only allow single symbols, transitive closure over wildcards, and concatenations thereof.

Miklau andSuciu (2004) were the first to investigate containment and satisfiability of tree pattern queries, which are acyclic versions of the CRPQs studied by Florescu, Levy, and Suciu (1998). Tree pattern queries are primarily considered on tree-structured data, but the complexity of their containment remains the same if one allows graph-structured data (Miklau andSuciu 2004; Czerwiński et al. 2018). Containment of tree pattern queries was considered in various forms in (Miklau andSuciu 2004; Neven andSchwentick 2006; Wood 2003; Czerwiński et al. 2015).

Björklund, Martens, andSchwentick (2011) studied containment of conjunctive queries over tree-structured data and proved a trichotomy, classifying the problems as in PTIME, coNP-complete, or \( \Pi^p_2 \)-complete. Their results cannot be lifted to general graphs since they use that, if a child has two direct ancestors, then they must be identical.

Sagiv andYannakakis (1980) studied the equivalence and therefore the containment problem of relational expressions with query optimization in mind. They show that when select, project, join, and union operators are allowed, containment is \( \Pi^p_2 \)-complete.

Chekuri andRajaraman (2000) showed that containment of conjunctive queries is in PTIME when the right-hand side has bounded treewidth. More precisely, they give an algorithm that runs in \( (\sum_{i=1}^{k} |Q_i|)^k \), where \( k \) is the width of \( Q_2 \). Their algorithm especially works for acyclic queries.

Calvanese et al. (2001) provide a PSPACE-algorithm for containment of tree-shaped CRPQs with inverses. The algorithm also works if only the right-hand side is tree-shaped. Figueira (2020) shows that containment of UC2RPQs is in PSPACE if the class of graphs considered has “bounded bridewidth” (= size of minimal edge separator is bounded) and is EXPSPACE-complete otherwise. Barceló, Figueira, andRomero (2019) studied the boundedness problem of UC2RPQs and prove that its EXPSPACE-completeness already holds for CRPQs. (A UC2RPQ is bounded if it is equivalent to a union of conjunctive queries.)

The practical study of (Bonifati, Martens, andTimm 2019) that we mentioned in the beginning of the paper and that was crucial for the motivation of this work would not have been possible without the efforts of the Dresden group on Knowledge-Based Systems (Malyshev et al. 2018), who made sure that anonymized query logs from Wikidata could be released. Bonifati, Martens, andTimm (2019) studied the same log files as Bielefeldt, Gonsior, andKrötzsch (2018).

It should be noted that several extensions and variants of CRPQs have been studied in the literature. Notable examples are nested regular expressions (Pérez, Arenas, andGutierrez 2010), CRPQs with node- and edge-variables (Barceló, Libkin, andReutter 2014), regular queries (Reutter, Romero, andVardi 2015), andGPQ (Libkin, Martens, andVrgoč 2016).

9 Conclusions and Further Work

We have provided an overview of the complexity of CRPQ containment in the case where the regular expressions in queries come from restricted, yet widely used classes in practice. A first main result is that, even when the regular expressions are from the restricted class CRPQ(\( a, A^* \)), the containment problem remains EXPSPACE-hard. Second, in the case that transitive closures are only allowed over single symbols, the complexity of CRPQ containment drops significantly. This is rather good news in practice, because, as we see in Table 2, the types of CRPQs for which containment falls into such lower complexities are extremely common.

Contrary to the EXPSPACE lower bound reduction of Calvanese et al. (2000), the shape of queries (i.e., its underlaying graph) is quite involved, and it crucially involves cycles. This immediately raises a number of questions.

- What is the complexity of Containment of CRPQ(\( a, A^* \)) in CRPQ(\( a, A^* \)) if one of the sides is only a path or a DAG?
- If one takes a careful look at our results, we actually settle the complexity of all forms of containment \( F_1 \subseteq F_2 \) where \( F_i \) is one of our considered classes, except the cases of Containment of CRPQ(\( a, A^* \)) in CRPQ(\( A \)) and Containment of CRPQ(\( a, A^* \)) in CRPQ(\( a, a^* \)). What is the complexity in these cases?

Of course, it would be interesting to understand which of our results can be extended towards C2RPQs, which would slightly increase the coverage of the queries we consider in Table 2. We believe that all our upper bounds can be extended and we plan to incorporate these results in an extended version of the paper.

Another direction could be to combine our fragments with arithmetical constraints. There is a lot of work done considering query containment of conjunctive queries with arithmetical constraints (which is \( \Pi^p_2 \)-complete), see for example Afrati (2019) and the related work mentioned there. We would like to understand to which extent such constraints can be incorporated without increasing the complexity of containment.

It would also be interesting to investigate the problem of boundedness (Barceló, Figueira, andRomero 2019) for the studied classes of CRPQ; understanding whether a query is ‘local’ might be of interest for the graph exploration during its evaluation.
Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — 369116833. The first author acknowledges the support of ANR DeLTA (grant ANR-16-CE40-0007) and ANR QUID (grant ANR-18-CE40-0031).

References


Neven, F., and Schwentick, T. 2006. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Methods in Computer Science 2(3).


