Expressive Languages for Path Queries over Graph-Structured Data

PABLO BARCELÓ, University of Chile
LEONID LIBKIN, University of Edinburgh
ANTHONY W. LIN, Univerity of Oxford
PETER T. WOOD, Birkbeck, University of London

For many problems arising in the setting of graph querying (such as finding semantic associations in RDF graphs, exact and approximate pattern matching, sequence alignment, etc.), the power of standard languages such as the widely studied conjunctive regular path queries (CRPQs) is insufficient in at least two ways. First, they cannot output paths and second, more crucially, they cannot express *relationships* among paths.

We thus propose a class of *extended* CRPQs, called ECRPQs, which add regular relations on tuples of paths, and allow path variables in the heads of queries. We provide several examples of their usefulness in querying graph structured data, and study their properties. We analyze query evaluation and representation of tuples of paths in the output by means of automata. We present a detailed analysis of data and combined complexity of queries, and consider restrictions that lower the complexity of ECRPQs to that of relational conjunctive queries. We study the containment problem, and look at further extensions with first-order features, and with non-regular relations that add arithmetic constraints on the lengths of paths and numbers of occurrences of labels.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*Data Models*; F.1.1 [**Computation by abstract devices**]: Models of Computation—*Automata*

General Terms: Theory, Languages, Algorithms

Additional Key Words and Phrases: Graph databases, conjunctive queries, regular relations, regular path queries

1. INTRODUCTION

For graph-structured data, queries that allow users to specify the types of paths in which they are interested have always played a central role. Most commonly, the specification of such paths has been by means of regular expressions over the alphabet of edge labels [Abiteboul et al. 1997; Calvanese et al. 2000; Consens and Mendelzon 1990; Florescu et al. 1998; Mendelzon and Wood 1995]. The output of a query is typically a set of tuples of nodes that are connected in some way by the paths specified. The canonical class of queries with this functionality are the conjunctive regular path queries (CRPQs). These have been the subject of much investigation, e.g. [Calvanese et al. 2000; Deutsch and Tannen 2001; Florescu et al. 1998].

However, the rapid increase in the size and complexity of graph-structured data (e.g. in the Semantic Web, or in biological applications) has raised the need for addi-

Authors' addresses: P. Barceló, Department of Computer Science, University of Chile, Avda Blanco Encalada 2120, 3er piso, Santiago, Chile; L. Libkin, School of Informatics, University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom; A. W. Lin, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom; P. T. Wood,Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, United Kingdom.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

tional functionality in query languages. Specifically, in many examples, the minimum requirements of sufficiently expressive queries are: (a) the ability to define complex semantic relationships between paths and (b) the ability to include paths in the output of the query. Neither of these is supported by CRPQs.

There are multiple examples of queries that require these new capabilities. For example, Anyanwu and Sheth [2003] introduce a query language for RDF/S in which paths can be compared based on specific semantic associations. Follow-up work in [Anyanwu et al. 2007] proposes the SPARQ2L query language, which extends SPARQL with path variables and path filtering expressions. In handling biological sequences one often needs to compare paths based on similarity (e.g., edit distance) [Gusfield 1997]. Paths can be compared with respect to other parameters, e.g., lengths or numbers of occurrences of labels, which can be useful in route-finding applications [Barrett et al. 2000].

As for the ability to output paths, this has been proposed, for example, as an extension to the SPARQL query language – the standard for retrieving RDF data [Kochut and Janik 2007]. However, that proposal only included a declarative language, and left most basic questions unexplored (e.g., what should an output be if there are infinitely many paths between nodes?). Other applications for this new functionality include determining the provenance of data or artifacts [Holland et al. 2008], finding associations in linked data [Lehmann et al. 2007], biological data [Lee et al. 2007] or social (or criminal) networks [Sheth et al. 2005], as well as performing semantic searches over web-derived knowledge [Weikum et al. 2009].

While the need for the extended functionality of graph query languages is well-documented (and sometimes is even incorporated into a programming syntax), the basic properties of such languages are completely unexplored. We do not know whether queries can be meaningfully evaluated, what their complexity is, whether they can be optimized, etc.

Our main goals, therefore, are to formally define extensions of graph queries that can express complex semantic associations between paths and output paths to the user, and to study them, concentrating on query evaluation and its complexity, as well as some static analysis problems.

We work with the class of *extended conjunctive regular path queries* or (ECRPQs), which generalize CRPQs by allowing them to express the kind of semantic association properties described above. That is, we allow

(1) n-tuples of path labels to be checked for conformity to n-ary path languages, and
(2) paths, rather than simply nodes, to be output.

As an example, consider a graph G with a single edge label, defining the student-advisor relationship. Using CRPQs, one can express many queries, such as finding academic ancestors, or people whose sets of academic parents and grandparents intersect, or checking whether x and y have a common academic ancestor (and if so, who that person is). However, with CRPQs we *cannot* express queries asking for pairs of scientists who have the same-length path to a given advisor, nor can we ask for the precise paths by which two people are related to their common academic ancestor. With ECRPQs, we *can* express such queries.

We now outline a few further examples of problems where the power of ECRPQs is required. They will be fully developed in Section 3, after we have presented the syntax and semantics of ECRPQs.

— *Pattern matching*. Given an alphabet Σ and a set of variables $\mathcal{V}$, a *pattern* is a string over $\Sigma \cup \mathcal{V}$. A pattern defines a pattern language by instantiating variables with strings in Σ^*. For example, the pattern $aXbX$ denotes the set of all strings of the form $a \cdot w \cdot b \cdot w$, where $w \in \Sigma^*$. Pattern languages need not be even context-free: e.g., the language of squared strings (i.e., strings $w \cdot w$ for $w \in \Sigma^*$) can be expressed by the pattern XX, where $X \in \mathcal{V}$. But finding nodes x and y connected by a path whose label is in the language of squared strings can be expressed by the ECRPQ:

$$Ans(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), \pi_1 = \pi_2$$

where x, y and z are node variables and π_1 and π_2 are path variables. Variables z, π_1, and π_2 are meant to be existentially quantified. What makes this different from CRPQs is the binary relation $\pi_1 = \pi_2$ on paths: it states that the path labels between x and an intermediate node z, and between z and y are the same.

— *Semantic web associations*. In RDF/S, properties can be declared to be subproperties of other properties. This is used by Anyanwu and Sheth [2003] to define a notion of semantic association based on ρ-*isomorphic* property sequences: two sequences are ρ-isomorphic if they are of the same length and the properties at the same position in each sequence are subproperties of one another. Such pairs of sequences can be found by a modification of the previous query with a different binary relation expressing the fact that the paths are ρ-isomorphic.

— *Approximate matching*. Approximate string matching [Grahne and Thomo 2004; Kanza and Sagiv 2001] and (biological) sequence alignment [Gusfield 1997] are both based on the notion of edit distance. The relation representing pairs of sequences that have edit distance at most k from one another, for some fixed k, is regular [Frougny and Sakarovitch 1991]. So given a graph representing a pair of sequences, an ECRPQ can determine whether they have *edit* distance at most k, i.e. if one of the sequences can be obtained from the other by applying at most k times the edit operations of insertion, deletion and substitution of a symbol. We show in Section 4 that we can also output the actual gaps and mismatches in the sequences using an ECRPQ.

What kinds of relations can be used to compare paths? Following the idea behind CRPQs, which allow regular conditions on paths, we use *regular relations* for path comparisons. Examples of regular relations (for now, binary) on paths π_1 and π_2 are:

— path equality: $\pi_1 = \pi_2$;
— length comparisons: $|\pi_1| = |\pi_2|$ (and likewise for $<$ and $\leq$);
— prefix: π_1 is a prefix of π_2;
— small edit distance: edit distance between π_1 and π_2 is at most k, for a fixed k;
— synchronous transformation: if $\pi_1 = a_1 \dots a_n$, then $\pi_2 = h(a_1) \dots h(a_n)$ for some map $h : \Sigma \to \Sigma$.

Even though specifying regular relations with regular expressions is probably less natural than specifying regular languages (at least it is more cumbersome), we still believe that our approach is of potential practical impact. In fact, any standard that intends to use many (or some) of the expressive benefits of ECRPQs, without including regular relations, could choose an appropriate set of commonly used regular relations (e.g., equality, length comparisons, or small edit distance), and use those as built-in predicates in a query language.

Note that many common relations are not regular, for example, the suffix relation, the substring or subsequence relation, or arithmetic comparisons of lengths of paths (e.g., $|\pi_1| = 2|\pi_2|$). Nonetheless, we shall show that some of the results can be extended to nonregular relations, in particular to path-length comparisons.

Outline of the results. After we formally define ECRPQs, we present an algorithm for query evaluation. It turns out that the sets of labels of paths satisfying a query are regular, and thus the evaluation algorithm constructs automata to represent such sets.

We then investigate the complexity of query evaluation. As yardsticks, we consider relational languages as well as CRPQs. For conjunctive queries, combined complexity is NP-complete, while it jumps to PSPACE-complete for relational calculus. Hence we cannot hope to get anything below NP for ECRPQs, and we hope not to exceed the complexity of relational queries in a reasonable class. As for data complexity, it is known to be NLOGSPACE-complete for CRPQs, so this will serve as another benchmark.

It turns out that the data complexity of ECRPQs matches that of CRPQs, but combined complexity goes up from NP to PSPACE, matching relational calculus instead. In this case it is natural to look for restrictions. A standard one for CQs is a restriction to acyclicity. This works for CRPQs – combined complexity becomes tractable – but does not work for ECRPQs, as the combined complexity remains PSPACE-complete. However, if our regular relations can only consider lengths of paths, then the complexity of ECRPQs drops to NP, matching the complexity of the usual relational CQs.

We then study extensions of CRPQs and ECRPQs: with negation and universal quantification, and with some non-regular relations. For the former, we get surprisingly reasonable bounds for CRPQs, but the complexity becomes too high when both negation and relations on paths are allowed. For the latter, we consider extensions with linear constraints on path lengths, and prove some good complexity bounds (tractable data complexity and NP combined complexity). We also consider relations that compare numbers of occurrences of labels in paths, and prove some low complexity bounds for queries with such relations.

While query containment is known to be decidable for CRPQs, we show that ECRPQs share more properties with full relational calculus: containment for them becomes undecidable. We recover decidability in one important subcase though.

Additional contributions. An extended abstract of this paper appeared in [Barceló et al. 2010]. The key new contributions are as follows. First, all results (unlike in the conference version) now come with complete detailed proofs. Second, all main questions left open in [Barceló et al. 2010] have been solved. In particular:

- [Barceló et al. 2010] asked whether the data complexity of ECRPQs with negation is elementary. We give a negative answer to this question in Theorem 8.2.
- [Barceló et al. 2010] asked whether decidability results on CRPQs with length comparisons extend to ECRPQs with length comparisons, and whether complexity bounds (in the case of decidability) remain the same. We give positive answers to both questions in Theorem 8.5.

We also asked in [Barceló et al. 2010] whether the containment of a CRPQ in an ECRPQ is decidable. While this paper was in preparation, a negative answer to this question was given by [Freydenberger and Schweikardt 2011]; this is also discussed in the paper.

Organization. In the next section, we present background material on graphs, regular relations and CRPQs. Section 3 introduces ECRPQs, while Section 4 discusses their applications in more detail. In Section 5, we consider the evaluation of ECRPQs. Section 6 deals with the data and combined complexity of ECRPQs. In Section 7 we study query containment, and in Section 8 we consider extensions with negation, and with non-regular features. Finally, Section 9 is devoted to related work while Section 10 summarizes the results obtained in the paper.

2. PRELIMINARIES

Labeled graphs and paths. Queries in our setting will be evaluated over labeled *graph databases*, that naturally model semistructured data. Formally, if Σ is a finite alphabet, then a Σ*-labeled graph database* G (or simply graph database if Σ is clear from the context) is a pair (V, E), such that V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of directed edges labeled in Σ.

A *path* ρ between nodes v_0 and v_m in G is a sequence $v_0 a_0 v_1 a_1 v_2 \cdots v_{m-1} a_{m-1} v_m$, where $m \geq 0$, so that all the v_i's are in V, all the a_j's are letters of Σ, and (v_i, a_i, v_{i+1}) is in E for each $i < m$. The *label* of such a path ρ, denoted by $\lambda(\rho)$, is the string $a_0 \cdots a_{m-1} \in \Sigma^*$. We also define the empty path as (v, ε, v) for each $v \in V$; the label of such a path is the empty string ε.

Note that a Σ-labeled graph database G can be naturally viewed as a nondeterministic finite automaton (NFA) over alphabet Σ without initial and final states. Its states are nodes in V, and its transitions are edges in E. We use this equivalence in several constructions in the paper.

Regular relations. As our plan is to extend the notion of recognizability from string languages to n-ary string relations, we now give the standard definition of *regular* relations over Σ [Elgot and Mezei 1965; Frougny and Sakarovitch 1991; Blumensath and Grädel 2000]. Regular relations are recognized by synchronous n-ary automata (also called *letter-to-letter* automata). These automata have n input tapes onto which the input strings are written, followed by an infinite sequence of $\perp$ symbols. At each step the automaton simultaneously reads the next symbol on each tape, terminating when it reads $\perp$ on each tape.

Formally, let $\perp$ be a symbol not in Σ. We denote the extended alphabet $(\Sigma \cup \{\perp\})$ by $\Sigma_\perp$. Let $\bar{s} = (s_1, \ldots, s_n)$ be an n-tuple of strings over alphabet Σ. We construct a string $[\bar{s}]$ over alphabet $(\Sigma_\perp)^n$, whose length is the maximum of the s_j's, and whose i-th symbol is a tuple $(c_1, \ldots, c_n)$, where each c_k is the i-th symbol of s_k, if the length of s_k is at least i, or $\perp$ otherwise. In other words, we pad shorter strings with the symbol $\perp$, and then view the n strings as one string over the alphabet of n-tuples of letters. For example, if $s_1 = aba$ and $s_2 = babb$, then $[(s_1, s_2)] = \binom{a}{b}\binom{b}{a}\binom{a}{b}\binom{\perp}{b}$, a string over $\Sigma_\perp \times \Sigma_\perp$.

An n-ary relation S on Σ^* is *regular*, if the set $\{[\bar{s}] \mid \bar{s} \in S\}$ of strings over alphabet $(\Sigma_\perp)^n$ is regular (i.e., accepted by an automaton over $(\Sigma_\perp)^n$, or given by a regular expression over $(\Sigma_\perp)^n$). We shall often use the same letter for both a regular expression over $(\Sigma_\perp)^n$ and the relation over Σ^* it denotes, as doing so will not lead to any ambiguity.

As an example, consider a binary relation $s \preceq s'$, saying that s is a prefix of s'. The automaton recognizing this relation accepts if it reads a sequence of letters of the form (a, a), for $a \in \Sigma$, possibly followed by a sequence of letters of the form $(\perp, b)$, for $b \in \Sigma$. As another example, consider a binary relation $\mathsf{el}(s, s')$ (equal length) saying that $|s| = |s'|$. This relation is recognized by an automaton that accepts if it does not see any letters involving the $\perp$ symbol.

To understand which relations on strings are regular, it is often useful to provide a model-theoretic characterization of this class. In the following we assume familiarity with first-order logic (FO). Consider the FO-structure $\mathcal{M}_{\text{univ}} = \langle \Sigma^*, \preceq, \mathsf{el}, (P_a)_{a \in \Sigma} \rangle$ with domain Σ^*, where $\preceq$ and el are as above, and $P_a(s)$ is true iff the last letter for s is a. This is known as a *universal automatic structure* due to the following [Blumensath and Grädel 2000; Bruyère et al. 1994]: an n-ary relation S on Σ^* is regular iff there exists an FO formula $\varphi_S(x_1, \ldots, x_n)$ over $\mathcal{M}_{\text{univ}}$ such that $S = \{\bar{s} \in (\Sigma^*)^n \mid \mathcal{M}_{\text{univ}} \models \varphi_S(\bar{s})\}$.

In particular, regular relations are closed under all Boolean combinations, product, and projection. Furthermore, using the above result it is quite easy to show that an n-

ary relation is regular, by exhibiting FO formulae defining them (see [Blumensath and Grädel 2000; Bruyère et al. 1994; Benedikt et al. 2003] for examples). For example, $|s| < |s'|$ is a regular relation definable by $\varphi(x, y) = \exists y'\ (y' \preceq y \wedge y' \neq y \wedge \mathrm{el}(y', x))$. On the other hand, more elaborate techniques have to be used to prove that an n-ary relation on Σ^* is *not* regular. Examples of this kind include the binary relation $\preceq_{ss}$, that consists of all pairs (s_1, s_2) such that s_1 is a subsequence of s_2, and the ternary relation that contains all tuples (s_1, s_2, s_3) such that $s_1 s_2 = s_3$.

Conjunctive regular path queries. A basic querying mechanism for graph databases is the class of *regular path queries* [Abiteboul et al. 1999; Calvanese et al. 2002] that retrieve all pairs of objects in a graph database connected by a path conforming to some regular expression. However, it has been argued (e.g. [Milo and Suciu 1999]) that in order to make regular path queries useful in practice, they should be extended with several features, one of them being the possibility of using conjunctions of atoms. This extension yields the class of *conjunctive* regular path queries, which we formally define below (see also [Consens and Mendelzon 1990; Mendelzon and Wood 1995; Florescu et al. 1998; Calvanese et al. 2000]). We use a syntax that will be easy to modify later to add the features of ECRPQs.

Fix a countable set of *node* variables (typically denoted by $x, y, z, \ldots$), and a countable set of *path* variables (denoted by $\pi, \omega, \chi, \ldots$). A *conjunctive regular path query* (CRPQ) Q over a finite alphabet Σ is an expression of the form:

$$Ans(\bar{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} L_j(\omega_j), \tag{1}$$

such that

(i). $m > 0$, $t \geq 0$,
(ii). each L_j is a regular expression over Σ,
(iii). $\bar{x} = (x_1, \ldots, x_m)$, $\bar{y} = (y_1, \ldots, y_m)$ and $\bar{z}$ are tuples of node variables,
(iv). $\{\pi_1, \ldots, \pi_m\}$ are distinct path variables,
(v). $\{\omega_1, \ldots, \omega_t\}$ are distinct path variables and each ω_j is among the π_i's, and
(vi). $\bar{z}$ is a tuple of node variables among $\bar{x}$ and $\bar{y}$.

The atom $Ans(\bar{z})$ is the *head* of the query, while the expression on the right of the $\leftarrow$ is its *body*. The query Q is *Boolean* if its head is of the form $Ans()$, i.e. $\bar{z}$ is the empty tuple.

Intuitively, such a query Q selects tuples $\bar{z}$ for which there exist values of the remaining node variables from $\bar{x}$ and $\bar{y}$ and paths π_i between x_i and y_i whose labels satisfy the regular expressions L_1 to L_t. Formally, to define the semantics of CRPQs Q of the form (1), we first introduce a relation $(G, \sigma, \mu) \models Q$, where:

(i). σ is a mapping from $\bar{x}, \bar{y}$ to the set of nodes of a graph database $G = (V, E)$, and
(ii). μ is a mapping from $\{\pi_1, \ldots, \pi_m\}$ to paths in G.

This relation holds iff $\mu(\pi_i)$ is a path in G from $\sigma(x_i)$ to $\sigma(y_i)$, for $1 \leq i \leq m$, and the label of each path $\mu(\omega_j)$ is in the language of L_j, for $1 \leq j \leq t$.

We now define $Q(G)$ to be the set of tuples $\sigma(\bar{z})$ such that $(G, \sigma, \mu) \models Q$. If Q is Boolean, we let $Q(G)$ be true if $(G, \sigma, \mu) \models Q$ for some σ and μ (that is, as usual, the empty tuple models the Boolean constant `true`, and the empty set models the Boolean constant `false`).

Remark: Our syntax differs slightly from the usual CRPQ syntax in the literature (see e.g. [Florescu et al. 1998; Calvanese et al. 2000]). The reason is that we make explicit use of path variables in the queries – to treat CRPQs and ECRPQs in a uniform manner – while the standard approach is to refer to paths only implicitly.

3. EXTENDED CONJUNCTIVE REGULAR PATH QUERIES

Our goal is to extend the class of CRPQs in two ways. First, we want to allow *free path variables* in the heads of queries. Second, we want the bodies of queries to permit checking *relations on sets of paths* rather than just conformance of individual paths to regular languages. This leads to the definition of a class of *extended CRPQs*.

Definition 3.1. An *extended conjunctive regular path query* (ECRPQ) Q over Σ is an expression of the form:

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\bar{\omega}_j), \tag{2}$$

such that

(i). $m > 0, t \geq 0$,
(ii). each R_j is a regular expression that defines a regular relation over Σ,
(iii). $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{y} = (y_1, \ldots, y_m)$ are tuples of node variables,
(iv). $\bar{\pi} = (\pi_1, \ldots, \pi_m)$ is a tuple of distinct path variables,
(v). $\{\bar{\omega}_1, \ldots, \bar{\omega}_t\}$ are distinct tuples of path variables, such that each $\bar{\omega}_j$ is a tuple of variables from $\bar{\pi}$, of the same arity as R_j,
(vi). $\bar{z}$ is a tuple of node variables among $\bar{x}, \bar{y}$, and
(vii). $\bar{\chi}$ is a tuple of path variables among those in $\bar{\pi}$.

□

Note that this is similar to the definition of CRPQs; the main differences between (1) and (2) are:

- ECRPQs can check whether a tuple of path labels belongs to a regular relation, rather than just checking whether a path label belongs to a regular language; and
- outputs of ECRPQs may contain both nodes and paths, while outputs of CRPQs contain only nodes.

The head, the body, and the notion of Boolean ECRPQs are defined in the standard way. The *relational* part of an ECRPQ Q of the form (2) is $\bigwedge_{1 \leq i \leq m}(x_i, \pi_i, y_i)$.

The semantics of ECRPQs is defined by a natural extension of the semantics of CRPQs. For an ECRPQ Q of the form (2), a graph database G and mappings σ from node variables to nodes and μ from path variables to paths, we write $(G, \sigma, \mu) \models Q$ if

- $\mu(\pi_i)$ is a path in G from $\sigma(x_i)$ to $\sigma(y_i)$, for $1 \leq i \leq m$, and
- for each $\bar{\omega}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$, the tuple of strings consisting of labels of $\mu(\pi_{j_1}), \ldots, \mu(\pi_{j_k})$ belongs to the relation R_j.

The output of Q on G (where the head of Q is $Ans(\bar{z}, \bar{\chi})$) is defined as

$$Q(G) = \{(\sigma(\bar{z}), \mu(\bar{\chi})) \mid (G, \sigma, \mu) \models Q\}.$$

Note that the implicit existential quantification over path variables that appear in the body but not in the head is quantification over a potentially infinite set, as there are infinitely many paths in any cyclic graph database.

From now on, we identify the class of CRPQs with the restriction of the class of ECRPQs to queries that do not use regular relations of arity ≥ 2. This is more general than the definition of the previous section, since we now allow CRPQs to output paths.

It is easy to prove that the class of ECRPQs is strictly more expressive than the class of CRPQs.

PROPOSITION 3.2. *There is an* ECRPQ *that is not equivalent to any* CRPQ.

PROOF. Let Q be a CRPQ with the head $Ans(x, y)$. With each string $s = a_0 \dots a_{n-1} \in \Sigma^+$, we associate a graph G_s with vertices $v_0, \dots, v_n$ and edges (v_0, a_0, v_1), (v_1, a_1, v_2), $\dots, (v_{n-1}, a_{n-1}, v_n)$. We let strings$(Q)$ be $\{s \in \Sigma^+ \mid (v_0, v_n) \in Q(G_s)\}$.

We now prove that strings(Q) is regular for every CRPQ Q. To see how the result follows from this, consider an ECRPQ Q'

$$Ans(x, y) \leftarrow (x, \pi, z),\ (z, \pi', y),\ a^+(\pi),\ b^+(\pi'),\ \mathrm{el}(\pi, \pi').$$

Q' selects nodes connected by a path with the label $a^m b^m$ for some $m > 0$. Hence strings$(Q') = \{a^m b^m \mid m > 0\}$ is not regular, and thus Q' is not equivalent to a CRPQ.

To show that strings(Q) is regular for a CRPQ Q, we explain how to construct an NFA for it. Assume that Q uses (existentially quantified) variables $z_1, \dots, z_k$. The automaton first guesses the order in which witnesses to these variables appear in G_s. This imposes an order on variables x, y, and $\bar{z}$, with x always coming first and y always coming last. For each pair of variables z and z' such that z comes before z', the query Q determines the language $L^Q_{zz'}$ that the path from z to z' must belong to as follows. Suppose we have clauses $(z, \pi, z'), L(\pi), (z', \pi', z), L'(\pi')$ (absent clauses can always be added, with the path between nodes belonging to Σ^*). Then the language $L^Q_{zz'}$ is $L \cap \mathrm{rev}(L')$, where $\mathrm{rev}(L')$ is the regular language of reversals of strings in L'.

The automaton for strings(Q), after making the initial guess of the ordering of variables, traverses the string, deciding when the witness for the next variable occurs. For keeping notation simple in the explanation below, assume that the order is $z_1, \dots, z_k$, with none of the variables witnessed by the same element. The automaton starts simulating all automata for the languages $L^Q_{xz_i}$ and L^Q_{xy}. When the automaton for $L^Q_{xz_1}$ is in a final state, it may guess that this is the point where the witness for z_1 occurs. Then it starts simulating all automata for the languages $L^Q_{z_1 z_i}$, for $i > 1$, and $L^Q_{z_1 y}$, now waiting to make a guess for z_2 in a state where the automata for $L^Q_{xz_2}$ and $L^Q_{z_1 z_2}$ are in accepting states. It continues like that; the final state will be those where all the automata for $L^Q_{z_i y}$ and L^Q_{xy} are in final states, confirming that the guesses have been made correctly. It is routine to check that this construction is correct. □

The result continues to hold for Boolean queries as well (see the electronic appendix).

4. APPLICATIONS OF EXTENDED CONJUNCTIVE REGULAR PATH QUERIES

In this section, we show that ECRPQs can express queries found in a wide variety of application areas, including finding associations in semantic web (or linked) data, pattern matching, approximate string matching, and biological sequence alignment.

Finding semantic web associations In a query language for RDF/S introduced in [Anyanwu and Sheth 2003], paths can be compared based on specific *semantic associations*. Edges correspond to RDF properties and paths to property sequences. A property a can be declared to be a subproperty of property b, which we denote by $a \prec b$. Two property sequences u and v are called ρ-isomorphic iff $u = u_1, \dots, u_n$ and $v = v_1, \dots, v_n$, for some n, and $u_i \prec v_i$ or $v_i \prec u_i$ for every $i \leq n$. Nodes x and y are called ρ-isoAssociated iff x and y are the origins of two ρ-isomorphic property sequences.

Finding nodes which are ρ-isoAssociated cannot be done in a query language supporting only conventional regular expressions, not least because doing so requires checking that two paths are of equal length. However, pairs of ρ-isomorphic sequences can be expressed using the regular relation R given by the following regular expression:

$$\left(\bigcup_{a,b \in \Sigma:\ (a \prec b \vee b \prec a)} (a, b) \right)^*$$

Then an ECRPQ returning pairs of nodes x and y that are ρ-isoAssociated can be written as follows:

$$Ans(x, y) \leftarrow (x, \pi_1, z_1), (y, \pi_2, z_2), R(\pi_1, \pi_2)$$

Path variables in an ECRPQ can also be used to return the actual paths found by the query, a mechanism found in the query languages proposed in [Abiteboul et al. 1997; Anyanwu and Sheth 2003; Holland et al. 2008; Kochut and Janik 2007]. For example, in [Anyanwu and Sheth 2003] a ρ-query can take a pair of nodes u, v and return the property sequences relating them. This too can be expressed by an ECRPQ:

$$Ans(\pi_1, \pi_2) \leftarrow (u, \pi_1, z_1), (v, \pi_2, z_2), R(\pi_1, \pi_2)$$

where the regular relation R is defined as above.

Pattern matching Let Σ be a finite alphabet and $\mathcal{V}$ be a countable set of variables such that $\Sigma \cap \mathcal{V} = \emptyset$. A *pattern* α is a string over $\Sigma \cup \mathcal{V}$. It denotes the language $L_\Sigma(\alpha)$ obtained by applying substitutions $\sigma : \mathcal{V} \to \Sigma^*$ to α. As we remarked already, such languages need not even be context-free.

However, for each pattern $\alpha = \alpha_1 \cdots \alpha_n$, where every $\alpha_i \in \Sigma \cup \mathcal{V}$, we can define an ECRPQ $Q_\alpha(x, y)$ which finds pairs of nodes connected by a path in $L_\Sigma(\alpha)$ (note that this property is not definable by a CRPQ). Indeed, the relational part of Q_α is $(x_0, \pi_1, x_1), \ldots, (x_{n-1}, \pi_n, x_n)$. If α_i is a letter $a \in \Sigma$, then Q_α contains the atom $a(\pi_i)$, while if α_i is a variable, then Q_α contains the atom $\Sigma^*(\pi_i)$. Finally, to ensure equality of variables, for every pair α_i, α_j which are the same variable, the query Q_α contains a conjunct $\pi_i = \pi_j$. It is clear that Q_α indeed finds nodes connected by paths whose labels are in $L_\Sigma(\alpha)$.

In fact, ECRPQs can express queries corresponding to a larger class of languages than the pattern languages. For instance, the language $a^n b^n c^n$, where $a, b, c \in \Sigma$ and $n \in \mathbb{N}$, cannot be denoted by patterns, but can easily be denoted by an ECRPQ with the help of the equal length predicate el (recall that $\text{el}(\pi, \pi')$ is a shorthand for $(\bigcup_{a,b \in \Sigma}(a, b))^*(\pi, \pi')$):

$$Ans(x, y) \leftarrow (x, \pi_1, z_1), (z_1, \pi_2, z_2), (z_2, \pi_3, y), a^*(\pi_1), b^*(\pi_2), c^*(\pi_3), \text{el}(\pi_1, \pi_2), \text{el}(\pi_2, \pi_3).$$

Approximate matching and sequence alignment We treat approximate string matching and (biological) sequence alignment together because both are based on the notion of edit distance between strings. We consider the three edit operations of insertion, deletion and substitution, defined as follows. Let $s, s' \in \Sigma^*$. Applying an edit operation to s yielding s' can be modeled as a binary relation $\rightsquigarrow$ over Σ^* such that $x \rightsquigarrow y$ holds iff there exist $u, v \in \Sigma^*$, $a, b \in \Sigma$, with $a \neq b$, such that one of the following is satisfied:

$$\begin{array}{lll} x = uav, & y = ubv & \text{(substitution)} \\ x = uav, & y = uv & \text{(deletion)} \\ x = uv, & y = ubv & \text{(insertion)} \end{array}$$

Let $\overset{k}{\rightsquigarrow}$ stand for the composition of $\rightsquigarrow$ with itself k times. The *edit distance* $d_e(x, y)$ between x and y is the minimum number k of edit operations such that $x \overset{k}{\rightsquigarrow} y$.

We define a relation $D^{\leq k}$ between strings as follows: $(x, y) \in D^{\leq k}$ iff $d_e(x, y) \leq k$. This relation is regular (indeed, it is easy to see that it is accepted by a two-tape transducer, and the difference between the lengths of x and y is bounded by k; then it follows from the fact that rational relations of such bounded distance are regular [Frougny and Sakarovitch 1991]).

We now consider the use of edit distance in finding string (or sequence) alignments. We can view an *alignment* of strings x and y over Σ at distance k as follows:

$$\begin{array}{rcl} x & = & x_0 \;\; a_1 \;\; x_1 \;\; \cdots \;\; a_k \;\; x_k \\ y & = & y_0 \;\; b_1 \;\; y_1 \;\; \cdots \;\; b_k \;\; y_k \end{array} \tag{3}$$

such that (i) $x_i, y_i \in \Sigma^*$ and $x_i = y_i$ for $i \in [0, k]$, and (ii) $a_i, b_i \in \Sigma \cup \{\epsilon\}$ and $a_i \neq b_i$, for $i \in [1, k]$. There is an alignment of x and y at distance k iff $(x, y) \in D^{\leq k}$. We call each pair (x_i, y_i) a *match* and each pair (a_i, b_i) a *mismatch* if $a_i, b_i \in \Sigma$ or a *gap* if a_i or b_i is ϵ. (If we allow that $a_i = b_i$, then we align the strings with distance *at most* k).

We have shown above that we can use an ECRPQ to determine whether there *exists* an alignment at distance k between two strings. However, we may also wish to return the actual gaps and mismatches to the user. For that, we assume that each node has an ϵ-labeled loop, and use an ECRPQ whose body is as follows

$$\bigwedge_{0\leq i\leq 2k} (x_i, \pi_i, x_{i+1}), \quad \bigwedge_{0\leq i\leq 2k} (y_i, \rho_i, y_{i+1}), \quad \bigwedge_{0\leq i\leq k} \pi_{2i} = \rho_{2i}, \quad \bigwedge_{1\leq i\leq k} R(\pi_{2i-1}, \rho_{2i-1}),$$

where R is a finite language containing all pairs (a, b) in $\Sigma \cup \{\epsilon\}$ with $a \neq b$. The head of the query contains the variables π_{2i-1}, ρ_{2i-1}, for $1 \leq i \leq k$.

With the same approach, we can use ECRPQs to align not only pairs but arbitrary tuples of sequences. Multiple sequence alignment is used to find the shared evolutionary origins of biological sequences.

In Section 8, we consider adding linear constraints on lengths of paths to ECRPQs. Such functionality would give us the ability to compare various properties of alignments. Given an alignment between sequences x and y as shown in (3) above, the number of mismatches between x and y is given by

$$m = \Sigma_{i=1}^{k} |a_i| \cdot |b_i|$$

while the total number of gaps in x and y is given by $k - m$. If $\Sigma_{i=1}^{k-1} |x_i| = 0$, for example, then all mismatches and gaps in x are consecutive. The number of non-gaps in x, for example, is given by $\Sigma_{i=1}^{k} |a_i|$. Overall we have the relationship

$$k = \Sigma_{i=1}^{k} |a_i| + \Sigma_{i=1}^{k} |b_i| - \Sigma_{i=1}^{k} |a_i| \cdot |b_i|$$

that is, k equals the number of non-gaps in x plus the number of non-gaps in y minus the number of mismatches.

5. QUERY EVALUATION

We now describe how ECRPQs can be evaluated. We need to take care of two aspects that distinguish ECRPQs from CRPQs: relations on paths, and path variables in the output. To deal with the former, we define a notion of *convolutions* of graph databases and queries, that reduces the evaluation of ECRPQs to the evaluation of CRPQs. To deal with the latter, we produce an automaton construction that can represent both nodes and paths in the output.

Convolutions of graphs and queries. We now present a construction that transforms a graph database G and an ECRPQ Q into a graph database G' and a CRPQ Q' with a single relational atom so that the evaluation of Q' over G' “coincides” (modulo a simple translation) with the evaluation of Q over G.

Let G be a Σ-labeled graph database. By $G_\perp$ we denote the $\Sigma_\perp$-labeled graph database obtained from G by adding a $\perp$-labeled loop to each node of G. We iteratively define G^m, the *m'th convolution* of G, as follows:

$$G^1 := G_\perp \;\text{ and }\; G^{m+1} = G_\perp \otimes G^m,$$

where $\otimes$ denotes the *product* of two graph databases. We use the symbol $\otimes$ rather than $\times$ to indicate that this is not the standard product viewed as a graph/automaton over the same alphabet, but rather a graph over the product of alphabets. Formally, given a Σ_1-labeled graph database $G_1 = (V_1, E_1)$ and a Σ_2-labeled graph database $G_2 = (V_2, E_2)$, their *product* $G_1 \otimes G_2$ is the $(\Sigma_1 \times \Sigma_2)$-labeled graph database $G = (V_1 \times V_2, E)$, where E contains edges $((v_1, v_2), (a, b), (v_1', v_2'))$, such that $(v_1, a, v_1') \in E_1$ and $(v_2, b, v_2') \in E_2$. Note that this makes G^m a $(\Sigma_\perp)^m$-labeled graph database.

Consider an ECRPQ Q of the form:

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_{2i-1}, \pi_i, x_{2i}), \bigwedge_{1 \leq j \leq t} R_j(\bar{\pi}^j). \tag{4}$$

Note that the variables $x_1, \ldots, x_{2m}$ are not necessarily distinct. Let S_j $(1 \leq j \leq t)$ be the n_j-ary regular relation defined by R_j. We let L_Q be the regular expression over $(\Sigma_\perp)^m$ that represents the m-ary regular relation $S_Q = S_1(\bar{\pi}^1) \bowtie \ldots \bowtie S_t(\bar{\pi}^t)$. Note that S_Q is indeed regular since the class of regular relations is closed under intersection, projection, and product, and that relations of the form $\{\bar{s} \mid s_i = s_j\}$, which are necessary for defining joins, are regular as well.

The *convolution* of ECRPQ Q (4) is the CRPQ query Q_c defined as

$$Ans(y, y', \pi) \leftarrow (y, \pi, y'), L_Q(\pi). \tag{5}$$

Note that this is indeed a CRPQ over $(\Sigma_\perp)^m$-labeled graph databases. Moreover, $Q_c(G^m)$, which consists of sets of tuples each comprising two m-tuples of nodes and a path in G^m, contains all the information we need to extract $Q(G)$; below, we show how to do this.

Let

$$\bar{\rho} = \bar{v}_0 \bar{a}_0 \bar{v}_1 \bar{a}_1 \bar{v}_2 \cdots \bar{v}_{p-1} \bar{a}_{p-1} \bar{v}_p$$

be a path in G^m, where $\bar{v}_i = (v_i^1, \ldots, v_i^m)$ for each $i \leq p$ is a node in G^m, and $\bar{a}_i = (a_i^1, \ldots, a_i^m)$ for each $i \leq p-1$ is an element of $(\Sigma_\perp)^m$. Then, for each $j \leq m$, we let

$$\bar{\rho}(j) = v_0^j a_0^j v_1^j \cdots v_{p-1}^j a_{p-1}^j v_p^j$$

be a path in $G_\perp$. Notice that this is indeed a path in $G_\perp$ but not necessarily in G, as it may contain $\perp$-labeled loops. We then let $\bar{\rho}_s(j)$ stand for the path obtained from $\bar{\rho}(j)$ by eliminating all such loops $v \perp v$; this is now a path in G.

The output of $Q_c(G^m)$ consists of tuples of the form $(\bar{u}, \bar{u}', \bar{\rho})$, where $\bar{u} = (u_1, u_3, \ldots, u_{2m-1})$ and $\bar{u}' = (u_2, u_4, \ldots, u_{2m})$ are nodes in G^m and $\bar{\rho}$ is a path in G^m. We say that $(\bar{u}, \bar{u}')$ are Q-compatible if, whenever $x_i = x_j$ in Q, we have $u_i = u_j$, for all $i, j \leq 2m$. We now define the *Q-compatible output of Q_c on G^m* as the projection of the set

$$\left\{ (\bar{u}, \bar{u}', \bar{\rho}_s(1), \ldots, \bar{\rho}_s(m)) \;\middle|\; \begin{array}{l} (\bar{u}, \bar{u}') \text{ is } Q\text{-compatible} \\ \text{and } (\bar{u}, \bar{u}', \bar{\rho}) \in Q_c(G^m) \end{array} \right\}$$

onto the attributes that appear in the head of Q in (4). That is, if x_i is among $\bar{z}$, we project onto u_i, and if π_j is among $\bar{\chi}$, we project onto $\bar{\rho}_s(j)$.

From all the previous remarks we immediately obtain the following:

THEOREM 5.1. *Let Q be an* ECRPQ *of the form (4) and G a graph database. Then the Q-compatible output of the convolution CRPQ Q_c on G^m coincides with $Q(G)$.*

Representing paths in the answer. Since ECRPQs can return paths, the answer to a query may be infinite (for example, if there is a cycle in the input graph, then we have infinitely many paths). In such cases we need to return a compact representation of the set of answers to an ECRPQ Q . It turns out that for each tuple of nodes $\bar{v}$, the set $\{\bar{\chi} \mid (\bar{v}, \bar{\chi}) \in Q(G)\}$ is a regular relation, and an automaton defining this relation can be constructed in time polynomial in the size of the input graph. We now present this construction.

Consider an ECRPQ Q of the form (2), i.e.,

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq p} R_j(\bar{\pi}^j),$$

a graph database $G = (V, E)$, and a tuple $\bar{v}$ of nodes such that $|\bar{v}| = |\bar{z}|$. We let $Q(G, \bar{v})$ stand for the set $\{\bar{\rho} \mid (\bar{v}, \bar{\rho}) \in Q(G)\}$.

Let $|\bar{\chi}| = k$. We say that a path $\bar{\pi}$ in G^k *represents* a k-tuple of paths $(\rho_1, \ldots, \rho_k)$ in $Q(G, \bar{v})$ if $\bar{\pi}_s(j) = \rho_j$ for each $j \leq k$ and the label of $\bar{\pi}$ is precisely $[\lambda(\rho_1), \ldots, \lambda(\rho_k)]$. Recall that $\lambda(\cdot)$ stands for the label of a path; in particular, each $\lambda(\rho_j)$ is a string in Σ^*. Recall also that $[\cdot]$ transforms a tuple of k strings into a string of k-tuples. Notice that such a path $\bar{\pi}$ is unique for the tuple $(\rho_1, \ldots, \rho_k)$, and in turn determines the tuple $(\rho_1, \ldots, \rho_k)$ uniquely.

PROPOSITION 5.2. *For each* ECRPQ Q *with the head* $Ans(z_1, \ldots, z_\ell, \chi_1, \ldots, \chi_k)$, *graph database* $G = (V, E)$ *and tuple* $\bar{v} \in V^\ell$, *one can construct, in polynomial time in* $|E|$, *an automaton* $\mathcal{A}_Q^{(G,\bar{v})}$ *over the alphabet* $V^k \cup (\Sigma_\perp)^k$ *that accepts precisely the representations of all the tuples of paths in* $Q(G, \bar{v})$.

In order to prove this proposition we use techniques similar to those in the proof of Theorem 6.3. Therefore, we postpone its proof until we prove Theorem 6.3.

6. COMPLEXITY OF QUERY EVALUATION

The reduction from ECRPQs to CRPQs gives us fairly easy upper bounds: one has to compute the convolution and evaluate a CRPQ over it. Using NLOGSPACE and NP bounds on the data and combined complexity of CRPQs, we conclude that the data complexity of ECRPQs is in PTIME, and their combined complexity is in EXPTIME. But can we do better?

It turns out that we can. For data complexity, we can lower the bound to NLOGSPACE: that is, the data complexity of CRPQs and ECRPQs is the same. For combined complexity, however, relations do make a difference: we show PSPACE-completeness of combined complexity. In the relational world, there are many techniques for lowering the NP combined complexity of conjunctive queries, typically by considering acyclic queries. This approach works for CRPQs, for which we show that acyclic queries can be evaluated in PTIME. However, when we move to ECRPQs, acyclicity does not lower the complexity. We then show that the techniques inspired by modeling infinite-state systems for verifying their temporal properties give us NP-completeness of combined complexity of classes of ECRPQs, matching the combined complexity of relational CQs.

6.1. Data complexity

If we fix a query Q over Σ, the problem we look at is the following:

PROBLEM:	ECRPQ-EVAL(Q)
INPUT:	A Σ-labeled graph database G, a tuple $\bar{v}$ of nodes in G and a tuple $\bar{\rho}$ of paths in G.
QUESTION:	Does $(\bar{v}, \bar{\rho})$ belong to $Q(G)$?

The convolution technique, if applied carefully, gives us an NLogspace upper bound. To evaluate the convolution query Q_c over G^m, we use an "on the fly" evaluation of the emptiness algorithm for the cross product of the automaton G^m, with a guessed assignment for the initial and final states, and the automaton $\mathcal{A}_Q$ that accepts the language L_Q of the convolution query. In the proof we still have to deal with some technical details (for instance, the presence of paths in the output for non-Boolean queries).

Theorem 6.1. *For each* ECRPQ *Q, the problem* ECRPQ-eval*(Q) is in* NLogspace.

For reasons that will become clear later, we postpone the proof of this theorem until immediately after the proof of Theorem 6.3.

Since the problem can be NLogspace-hard even for regular path queries that do not make use of path variables in the head [Consens and Mendelzon 1990], we also have a matching lower bound. Also note that when query Q is fixed, Proposition 5.2 tells us that there is a polynomial-size family of automata that represents the whole space of answers for Q over G.

6.2. Combined complexity

We now turn to the combined complexity, that is, query evaluation that takes both the graph database and the query as input:

Problem:	ECRPQ-eval
Input:	A finite alphabet Σ, a Σ-labeled graph database G, an ECRPQ Q over Σ, a tuple $\bar{v}$ of nodes in G and a tuple $\bar{\rho}$ of paths in G.
Question:	Does $(\bar{v}, \bar{\rho})$ belong to $Q(G)$?

The problem CRPQ-eval is the restriction to when the query Q in the input is a CRPQ.

We start with the easier problem CRPQ-eval. It appears to be a folklore result (although we could not find it stated explicitly in the literature) that, without path variables in the head (i.e., the empty tuple $\bar{\rho}$) this problem is NP-complete. For the sake of completeness we present a proof (in the appendix) of a slightly more general result that handles free path variables as well.

Proposition 6.2. CRPQ-eval *is* NP*-complete.*

However, adding regular relations to queries makes the query evaluation problem harder (at least under widely-held complexity theoretical assumptions). Notice that this is in stark contrast with what happens in the same case to the data complexity of the problem, where relations on paths do not increase the complexity.

Theorem 6.3. ECRPQ-eval *is* Pspace*-complete. It remains* Pspace*-hard even when restricted to Boolean* ECRPQ*s.*

Proof. We first prove membership. Let Q be an ECRPQ of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\bar{\omega}_j),$$

$\bar{v}$ a tuple of nodes in G such that $|\bar{v}| = |\bar{z}|$ and $\bar{\rho}$ a tuple of paths in G such that $|\bar{\rho}| = |\bar{\chi}|$. From Proposition 5.1, checking whether $(\bar{v}, \bar{\rho}) \in Q(G)$ is equivalent to checking whether $(\bar{v}, \bar{\rho})$ belongs to the Q-compatible output of Q_c over G^m, where Q_c is the convolution $Ans(y, y', \pi) \leftarrow (y, \pi, y'), L_Q(\pi)$ of Q as defined in Section 3. We show that the latter can be done in nondeterministic polynomial space in the size of Q and G. Then membership will follow from Savitch's Theorem, that shows that PSPACE equals NPSPACE. First, we notice the following:

LEMMA 6.4. *It is possible to construct from Q an automaton $\mathcal{A}_Q$, of size at most exponential in the size of Q, such that $\mathcal{A}_Q$ accepts exactly the language defined by L_Q.*

PROOF. Recall that L_Q is defined over $(\Sigma_\perp)^m$. Further, assume that R_j is defined over $(\Sigma_\perp)^{n_j}$, $n_j > 0$, and that $\bar{\omega}_j = (\pi_{j_1}, \ldots, \pi_{j_{n_j}})$. Each regular expression R_j $(1 \leq j \leq t)$ can be transformed in polynomial time in the size of Q into an equivalent NFA $\mathcal{A}_j$. Assume that $p_j \geq 0$ is the number of states of $\mathcal{A}_j$, for each $1 \leq j \leq t$. Then we can define the NFA $\mathcal{A}_Q$ as the one that on string $\bar{w} = (a_1^1, \ldots, a_m^1) \cdots (a_1^\ell, \ldots, a_m^\ell)$ over alphabet $(\Sigma_\perp)^m$ does the following: It checks that for each $1 \leq j \leq t$ the string

$$(a_{j_1}^1, \ldots, a_{j_{n_j}}^1) \cdots (a_{j_1}^\ell, \ldots, a_{j_{n_j}}^\ell)$$

over alphabet $(\Sigma_\perp)^{n_j}$ is accepted by $\mathcal{A}_j$. If that is the case, $\mathcal{A}_Q$ accepts $\bar{w}$; otherwise, it rejects.

Clearly, $\mathcal{A}_Q$ can be constructed by taking the cross product of all the $\mathcal{A}_j$'s and then projecting each coordinate on the corresponding indexes from $\{1, \ldots, m\}$. That is, assume that $(r_1, \ldots, r_m)$ and $(r'_1, \ldots, r'_m)$ are two states of $\mathcal{A}_Q$, where r_j is a state of $\mathcal{A}_j$, for each $1 \leq j \leq t$. Then there is a transition in $\mathcal{A}_Q$ labeled $(a_1, \ldots, a_m) \in (\Sigma_\perp)^m$ if and only if there is a transition labeled $(a_{j_1}, \ldots, a_{n_j})$ from r_j to r'_j in $\mathcal{A}_j$, for each $1 \leq j \leq t$.

Thus, $\mathcal{A}_Q$ has at most $\max\{p_j \mid 1 \leq j \leq t\}^m$ states. Since the alphabet of $\mathcal{A}_Q$ is also of exponential size with respect to Q, we get that $\mathcal{A}_Q$ is of size at most exponential in the size of Q. □

Notice that, since $\mathcal{A}_Q$ is of exponential size in the size of Q, we can assume without loss of generality that each state of $\mathcal{A}_Q$ is of polynomial size in Q. We can then define an algorithm that checks in nondeterministic polynomial space whether $(\bar{v}, \bar{\rho})$ belongs to the Q-compatible output of Q_c on G_m as follows. The algorithm first guesses a polynomial space assignment σ from $\{y, y'\}$ to the nodes of G^m. Then the algorithm checks that σ "respects" $\bar{v}$, i.e. for each z in $\bar{z}$, if the value in $\bar{v}$ that corresponds to the variable z is v, then $\sigma(z) = v$. Afterwards, the algorithm checks whether $(\sigma(y), \sigma(y'))$ is Q-compatible. All this can be done in polynomial space.

Recall that $G^m(\sigma(y), \sigma(y'))$ is the NFA that is obtained from G^m by fixing $\sigma(\bar{y})$ and $\sigma(\bar{y}')$ as initial and final states, respectively. It is not hard to see that $G^m(\sigma(y), \sigma(y'))$ is of exponential size in G and Q, but the size of each one of its states is polynomial in the size of G and Q. Further, $(\bar{v}, \bar{\rho})$ belongs to the Q-compatible output of Q_c over G^m if and only if there is a string $\bar{a}_1 \cdots \bar{a}_p$ over alphabet $(\Sigma_\perp)^m$ that is accepted by $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ and that "respects" $\bar{\rho}$; this means that there is a run

$$(q_0, \bar{u}_0), \ldots, (q_p, \bar{u}_p)$$

of $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ over $\bar{a}_1 \cdots \bar{a}_p$, such that (1) each q_i $(0 \leq i \leq p)$ is a state of $\mathcal{A}_Q$, (2) each $\bar{u}_i$ $(0 \leq i \leq p)$ is a state of $G^m(\sigma(y), \sigma(y'))$ (i.e. a node of G^m), (3) q_0 and q_p are an initial and final state of $\mathcal{A}_Q$, respectively, (4) $\bar{u}_0$ and $\bar{u}_p$ are the initial and final state of $G^m(\sigma(y), \sigma(y'))$, respectively (i.e. $\bar{u}_0 = \sigma(y)$ and $\bar{u}_p = \sigma(y')$), and (5) if $\bar{\mu}$ is the path $\bar{u}_0\bar{a}_1\bar{u}_1\bar{a}_2\bar{u}_2 \cdots \bar{u}_{p-1}\bar{a}_{p-1}\bar{u}_p$ in G^m, then for each $1 \leq j \leq m$ and variable χ in $\bar{\chi}$ such that $\chi = \pi_j$ it is the case that $\bar{\mu}_s(j) = \rho$, assuming that ρ is the value that corresponds to χ in the tuple $\bar{\rho}$.

As we have mentioned, both $\mathcal{A}_Q$ and $G^m(\sigma(y), \sigma(y'))$ are of exponential size. However, each state in $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ is of polynomial size. Thus, checking whether there is a string in $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ that "respects" $\bar{\rho}$ can be done in nondeterministic PSPACE by using a standard "on-the-fly" construction of $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ as follows: Whenever the emptiness algorithm wants to move from a state r_1 of $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ to a state r_2, it guesses r_2 and checks whether there is a transition from r_1 to r_2. Once this is done, the algorithm can discard r_1 and follow from r_2. Thus, at each step, the algorithm needs to keep track of at most two states, each one of polynomial size. Further, it is not hard to see that the extra constraint for the emptiness algorithm of finding a string that respects $\bar{\rho}$ does not increase the complexity.

For hardness we use a polynomial-time reduction from the problem of regular expression intersection (REI), which is known to be PSPACE-complete [Kozen 1977]. This problem is defined as follows: Given m regular expressions $R_1, \dots, R_m$ over a common alphabet $\Sigma = \{a_1, \dots, a_n\}$, is there a string $w \in \Sigma^*$ that belongs to R_i for each $i \in [1, m]$? It follows from [Kozen 1977] that REI remains PSPACE-complete even for a fixed Σ.

Let $\mathcal{R} = R_1, \dots, R_m$ be an instance of the REI problem, and assume that the common alphabet of the R_i's is $\Sigma = \{a_1, \dots, a_n\}$. We construct (in polynomial time) an instance $(\Sigma, G_{\mathcal{R}}, Q_{\mathcal{R}})$ of the ECRPQ evaluation problem as follows:

— $Q_{\mathcal{R}}$ is the Boolean query

$$Ans() \leftarrow \bigwedge_{i \in [1,m]} (x_i, \pi_i, y_i),\ R_i(\pi_i),\ \bigwedge_{i,j \in [1,m]} \pi_i = \pi_j,$$

such that the elements in the set $\{x_1, \dots, x_m, y_1, \dots, y_m\}$ are pairwise distinct. That is, $Q_{\mathcal{R}}$ asks whether there exist nodes $u_1, \dots, u_m, v_1, \dots, v_m$ and paths $\rho_1, \dots, \rho_m$ in the graph, such that ρ_i ($i \in [1, m]$) is path from u_i to v_i and $\lambda(\rho_i)$ belongs to R_i, and for each $i, j \in [1, m]$, $\lambda(\rho_i) = \lambda(\rho_j)$.

— $G^{\Sigma}_{\mathcal{R}} = (V, E)$ is the Σ-labeled graph database such that $V = \{v_1, \dots, v_{n+1}\}$ and E contains all tuples of the form (v_i, a, v_j), such that (1) $i, j \in [1, n+1]$ and $i \neq j$, and (2) $a = a_{j-1}$ if $i < j$, and $a = a_j$ otherwise. Notice that for each node v in $G^{\Sigma}_{\mathcal{R}}$ and string $w \in \Sigma^*$, there is a path ρ in $G^{\Sigma}_{\mathcal{R}}$ that starts in v and such that $\lambda(\rho) = w$.

We prove next that there is a string $w \in \Sigma^*$ that belongs to R_i for each $i \in [1, m]$ iff $Q_{\mathcal{R}}(G^{\Sigma}_{\mathcal{R}}) = \texttt{true}$. Assume first that $Q_{\mathcal{R}}(G^{\Sigma}_{\mathcal{R}}) = \texttt{true}$. Then there exist nodes $u_1, \dots, u_m, v_1, \dots, v_m$ and paths $\rho_1, \dots, \rho_m$ in $G^{\Sigma}_{\mathcal{R}}$, such that (1) ρ_i ($1 \leq i \leq m$) is a path from u_i to v_i, (2) $\lambda(\rho_i)$ belongs to R_i, and (3) for each $1 \leq i, j \leq m$ it is the case that $\lambda(\rho_i) = \lambda(\rho_j)$. In particular, the string $\lambda(\rho_i)$ belongs to R_i, for each $i \in [1, m]$. On the other hand, assume that $w \in \Sigma^*$ belongs to each R_i. Let ρ be an arbitrary path that starts in v_1 and such that $\lambda(\rho) = w$. Assume that ρ goes from v_1 to v. Then clearly there exist nodes $u_1, \dots, u_m, v_1, \dots, v_m$ (namely, $u_i = v_1$ and $v_i = v$ for each $1 \leq i \leq m$) and paths $\rho_1, \dots, \rho_m$ in $G^{\Sigma}_{\mathcal{R}}$ (namely, $\rho_i = \rho$ for each $1 \leq i \leq m$), such that (1) ρ_i ($1 \leq i \leq m$) is path from u_i to v_i, (2) $\lambda(\rho_i)$ belongs to R_i, and (3) for each $1 \leq i, j \leq m$ it is the case that $\lambda(\rho_i) = \lambda(\rho_j)$. This implies that $Q_{\mathcal{R}}(G^{\Sigma}_{\mathcal{R}}) = \texttt{true}$. □

By adapting the previous machinery we can now give a proof of Proposition 5.2 and Theorem 6.1.

PROOF OF THEOREM 6.1. The same "on-the-fly" algorithm used in the proof of Theorem 6.3 works. Notice, however, that in this case (that is, with Q assumed to be fixed) the size of $\mathcal{A}_Q \times G^m(\sigma(y), \sigma(y'))$ is polynomial. Further, each one of its states can be represented with $O(\log |G|)$ bits. This implies that the nondeterministic algorithm of the previous proof works in NLOGSPACE. □

PROOF OF PROPOSITION 5.2. Let Q be a fixed ECRPQ of the form

$$Ans(z_1, \ldots, z_\ell, \chi_1, \ldots, \chi_k) \leftarrow \bigwedge_{1\leq i\leq m} (x_i, \pi_i, y_i), \bigwedge_{1\leq j\leq t} R_j(\bar{\omega}_j),$$

$G = (V, E)$ be a Σ-labeled graph database, $\bar{v} = (v_1, \ldots, v_\ell)$ be a tuple of nodes in G, and Q_c be the convolution $Ans(y, y', \pi) \leftarrow (y, \pi, y'), L_Q(\pi)$ of Q as defined in Section 3.

First, define Θ as the set of all those mappings σ from $\{y, \bar{y}'\}$ to G^k that "respect" $\bar{v}$ and that satisfy that $(\sigma(y), \sigma(y'))$ is Q-compatible. Then construct, in polynomial time in $|E|$, the NFA $\mathcal{A}$ that accepts the intersection of $\bigcup_{\sigma\in\Theta} G^k(\sigma(y), \sigma(y'))$ and the NFA $\mathcal{A}_Q$ that recognizes L_Q. It is not hard to see that from $\mathcal{A}$ it is possible to construct, in polynomial time, an automaton $\mathcal{B}$ over alphabet $V^k \cup (\Sigma_\perp)^k$, that accepts exactly those strings $\bar{v}_0\bar{a}_1\bar{v}_1\bar{a}_2 \cdots \bar{v}_{p-1}\bar{a}_p\bar{v}_p$ such that (1) each $\bar{v}_i$ ($0 \leq i \leq p$) is a node in V^k, (2) each $\bar{a}_i$ ($1 \leq i \leq p$) is an element of $(\Sigma_\perp)^k$, and (3) $\bar{v}_0, \ldots, \bar{v}_p$ is a successful run of $\mathcal{A}$ over $\bar{a}_1 \cdots \bar{a}_p$ (that is, in particular $\bar{v}_0$ and $\bar{v}_p$ are initial and final states of $\mathcal{A}$, respectively). Finally, by projecting on the corresponding path variables, it is possible to construct in polynomial time in $|\mathcal{B}|$ the NFA $\mathcal{A}_Q^{(G,\bar{v})}$ that accepts exactly the restriction of the paths accepted by $\mathcal{B}$ to the variables in $\bar{\chi}$. Thus, the whole process takes polynomial time and the size of $\mathcal{A}_Q^{(G,\bar{v})}$ is polynomial in $|E|$. □

6.3. Restrictions

We now look at various approaches to lowering the complexity of query evaluation.

Acyclic queries. It is, of course, a classical result of relational theory that acyclic conjunctive queries are tractable with respect to combined complexity. What if we require that the relational part of an (E)CRPQ be acyclic? Formally, we say that an ECRPQ or a CRPQ Q is acyclic if the graph H_Q of its relational part $\bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i)$, containing precisely the edges (x_i, y_i) for $i \leq m$, is acyclic.

The following result shows that the situation is similar for CRPQs but drastically different for ECRPQs: the restriction works for the former but not for the latter. In fact, allowing only unary regular relations is precisely the boundary of tractability for the query evaluation problem restricted to acyclic ECRPQs.

THEOREM 6.5.

— *The problem* CRPQ-EVAL *is in* PTIME, *if restricted to the class of acyclic CRPQs.*
— *The problem* ECRPQ-EVAL *is* PSPACE-*complete, even if restricted to the class of acyclic Boolean* ECRPQ*s, over a fixed alphabet* Σ, *that make use of regular relations of arity at most* 2.

PROOF. The second result uses the reduction of Theorem 6.3, which requires Boolean acyclic queries and binary relations over a fixed alphabet. Next we prove the first result.

Let Σ be a finite alphabet, $G = (V, E)$ a Σ-labeled graph database, Q a CRPQ over Σ of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m} (x_i, \pi_i, y_i), \bigwedge_{1\leq i\leq t} L_j(\omega_j),$$

$\bar{v}$ a tuple of nodes in G and $\bar{\rho}$ a tuple of paths in G. We assume without loss of generality that $|\bar{v}| = |\bar{z}|$ and $|\bar{\chi}| = |\bar{\rho}|$.

We start by transforming Q into a Boolean query Q' that makes use of constants for nodes and paths as follows. First, instantiate the node variables in the body of Q which appear in $\bar{z}$ with the corresponding nodes from $\bar{v}$. In the same way, instantiate the path variables in the body of Q which appear in $\bar{\chi}$ with the corresponding paths from $\bar{\rho}$.

Now, for each $i \in [1, m]$, the combination of (x_i, π_i, y_i) and the $L_j(\omega_j)$'s can be evaluated in PTIME to yield a binary relation r_i over pairs of nodes v_1 and v_2 such that $(v_1, v_2) \in r_i$ iff there is a path from v_1 to v_2 in G satisfying R_j, for each $1 \leq j \leq t$ such that $\omega_j = \pi_i$. This can be done using the standard technique of forming an NFA to recognize the intersection of the languages denoted by R_i and G, except that G is now viewed as an automaton in which every node is both an initial and final state (unless x_i and/or y_i has been instantiated to a node in G, in which case the initial and/or final states are restricted accordingly; and unless π_i has been instantiated to a path in G, in which case we have to restrict the initial an/or final states accordingly, as well as checking whether the label of such a path satisfies R_j, for each $1 \leq j \leq t$ such that $\omega_j = \pi_i$).

It is not hard to see then that the problem of checking whether $(\bar{v}, \bar{\rho}) \in Q(G)$ is equivalent to the problem of evaluating an acyclic conjunctive query Q', whose size is linear in the size of Q, over the relational database that contains the relations $r_1, \ldots, r_m$. Since each r_i can be constructed in polynomial time, and the evaluation of acyclic conjunctive queries over relational databases is known to be in PTIME, we conclude that the problem of evaluating acyclic CRPQs over graph databases is in PTIME. □

Numerical representations of regular relations. The idea here comes from the field of verification of infinite-state systems, where regular languages are used to represents possible states of such systems (e.g., strings of states of an unbounded number of components of a system) and regular relations represent transitions between them [Abulla et al. 2003]. While many problems related to verifying such systems are computationally hard or even undecidable, they become easier if an abstraction of a regular language presentation is used. Often such abstractions are in the form of definability in linear integer arithmetic, see, e.g., [Verma et al. 2005; Ibarra et al. 2002]. We shall now look at a similar idea of abstracting regular relations in connection with the ECRPQ evaluation problem.

We first look at relations that consider only the *lengths* of strings. More precisely, with each n-ary regular relation R, we associate a regular relation R_{len} defined as

$$\{(s_1, \ldots, s_n) \mid \exists (s'_1, \ldots, s'_n) \in R : \ |s_i| = |s'_i| \text{ for all } i\}.$$

Now, given an ECRPQ Q, we define Q_{len} as Q in which each relation R is replaced by R_{len}. This is still an ECRPQ due to the following:

LEMMA 6.6. *If R is a regular relation, then so is R_{len}.*

It turns out that with this abstraction, we can lower the combined complexity to that of ordinary relational conjunctive queries.

THEOREM 6.7. *The problem of checking, for a graph database G, a tuple $\bar{a}$, and an* ECRPQ *Q, whether $\bar{a} \in Q_{\text{len}}(G)$, is* NP*-complete.*

We now prove both Lemma 6.6 and Theorem 6.7.

PROOF. We start with a few observations about relations of the form R_{len}. Let R be an m-ary relation over Σ. By $\mathcal{P}$ we denote an ordered partition of $\{1, \ldots, m\}$, i.e., a partition of $\{1, \ldots, m\}$ into blocks $B_1, \ldots, B_k$ for some $k \leq m$ with an order $B_1 < \ldots < B_k$ on them. Given an m-tuple $\bar{s} = (s_1, \ldots, s_m)$ of strings over Σ, we say that it *length-conforms* to $\mathcal{P}$ if two conditions hold:

— if i, j are in the same block of the partition $\mathcal{P}$, then $|s_i| = |s_j|$; and
— if $i \in B_p$ and $j \in B_q$ with $p < q$, then $|s_i| < |s_j|$.

By $\ell(\bar{s})$ we denote the tuple $(|s_1|, \ldots, |s_m|) \in \mathbb{N}^m$, and by $\ell_{\mathcal{P}}(\bar{s})$, if $\bar{s}$ length-conforms to $\mathcal{P}$, the following tuple $(n_1, \ldots, n_k)$:

— $n_1 = |s_j|$ for an arbitrary $j \in B_1$;
— $n_{p+1} = |s_j| - |s_i|$ for arbitrary $s_j \in B_{p+1}$ and $s_i \in B_p$.

Since $\bar{s}$ length-conforms to $\mathcal{P}$, this is unambiguous.

Recall that a unary automaton is an automaton over a one-letter alphabet; its transitions therefore can be viewed as a binary relation on the set of states. Given such an automaton $\mathcal{A}$, it accepts strings of the form a^n for the single-letter a, which we identify with numbers, writing instead that $\mathcal{A}$ accepts $n \in \mathbb{N}$.

We now need the following claim. Assume that R is given by a letter-to-letter NFA $\mathcal{A}_R$. Let $\mathcal{A}$ be an arbitrary automaton; by $\mathcal{A}(I, F)$ we mean an automaton with the same transitions as $\mathcal{A}$, but initial states I and final states F. Now if we have a sequence $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of unary automata, we say that it *accepts* $\bar{n} = (n_1, \ldots, n_k)$ *via a tuple of states* $(q_1, \ldots, q_{k-1})$ if there is an initial state q_0 of $\mathcal{A}_1$ and a final state q_k of $\mathcal{A}_k$ so that n_j is accepted by $\mathcal{A}_j(q_{j-1}, q_j)$, for all $j \leq k$. Note that each state q_j, for $1 \leq j < k$, must be shared by $\mathcal{A}_j$ and $\mathcal{A}_{j+1}$.

CLAIM 6.7.1. *Given a regular relation R over m-ary tuples of strings, and an ordered partition $\mathcal{P}$ of $\{1, \ldots, m\}$ with k blocks, there exist unary automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and sets of states $S_1, \ldots, S_k$, which can be effectively constructed from $\mathcal{A}_R$ and $\mathcal{P}$, such that, for each $\bar{s}$ that length-conforms to $\mathcal{P}$,*

$$\bar{s} \in R_{\text{len}} \quad \Leftrightarrow \quad \ell_{\mathcal{P}}(\bar{s}) \textit{ is accepted by } \mathcal{A}_1, \ldots, \mathcal{A}_k \textit{ via some tuple } (q_1, \ldots, q_{k-1}),$$

where each q_j is in S_j, for $j \leq k$.

We now prove the claim. First note that membership in R_{len} for tuples that length-conform to $\mathcal{P}$ depends only on $\ell_{\mathcal{P}}(\bar{s})$: if both $\bar{s}$ and $\bar{s}'$ length-conform to $\mathcal{P}$ and $\ell_{\mathcal{P}}(\bar{s}) = \ell_{\mathcal{P}}(\bar{s}')$, then $\bar{s} \in R_{\text{len}}$ iff $\bar{s}' \in R_{\text{len}}$.

We also need the following observation. We say that from a state q of $\mathcal{A}_R$, a final state is *reachable by* $(\mathcal{P}, j)$ if it is reachable by reading a string that starts with letters that have $\perp$ in positions corresponding to B_i for $i < j$ and letters from Σ in positions corresponding to B_p for $p \geq j$, then continues with letters that have $\perp$ in positions corresponding to B_i for $i < j + 1$ and letters from Σ in positions corresponding to B_p for $p \geq j+1$, and so on, until it finally reads a sequence of letters having $\perp$ in positions corresponding to $B_1, \ldots, B_{k-1}$ and letters from Σ in positions on B_k, before reaching a final state. By the standard reachability analysis of the automaton graph we can see that the set of such states can be computed in polynomial time in the size of $\mathcal{A}_R$.

We now construct the sequence of unary automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ inductively as follows. The states of all the automata will be the states of $\mathcal{A}_R$. The transitions will be subsets of transitions of $\mathcal{A}_R$.

— Automaton $\mathcal{A}_1$, while reading its input, guesses at each step a letter in Σ^m (i.e., no $\perp$-symbols) and simulates a transition of $\mathcal{A}_R$. Its accepting states are those (reachable) states from which a final state of $\mathcal{A}_R$ is reachable by $(\mathcal{P}, 2)$.
— The automaton $\mathcal{A}_{p+1}$ has as its set of initial states the final states of $\mathcal{A}_p$. It simulates $\mathcal{A}_R$ on letters that have $\perp$ in positions corresponding to $B_1, \ldots, B_p$, and symbols in Σ in positions corresponding to $B_{p+1}, \ldots, B_k$ (i.e., it guesses such a letter for each input symbol). Its accepting states are those from which a final state of $\mathcal{A}_R$ is reachable by $(\mathcal{P}, p+2)$. If $p + 1 = k$, its accepting states are the reachable final states of $\mathcal{A}_R$.

Suppose now $\bar{s} \in R_{\text{len}}$. Then $\ell_{\mathcal{P}}(\bar{s})$ is accepted by $\mathcal{A}_1, \ldots, \mathcal{A}_k$ via some tuples of states: simulating $\mathcal{A}_R$ gives us an accepting run. Conversely, if $\ell_{\mathcal{P}}(\bar{s})$ is accepted by $\mathcal{A}_1, \ldots, \mathcal{A}_k$ via a tuple of states $(q_1, \ldots, q_{k-1})$, from the description of the $\mathcal{A}_i$'s we get a tuple of strings $\bar{s}'$ (the guessed strings) so that $\bar{s}'$ is accepted by $\mathcal{A}_R$ and $\ell_{\mathcal{P}}(\bar{s}) = \ell_{\mathcal{P}}(\bar{s}')$; thus $\bar{s}$ is accepted by R_{len}. This proves the claim.

We now use the fact that the language accepted by a unary NFA $\mathcal{A}$ is a union of at most quadratically many arithmetic progressions (i.e., sets $\theta = a + b\mathbb{N} = \{a + bn \mid n \in \mathbb{N}\}$). This was first claimed in [Chrobak 1986], and a mistake in that paper was recently fixed in [To 2009], which also showed that such a set of arithmetic progressions can be constructed in polynomial time. This, together with Claim 6.7.1, gives us the following.

CLAIM 6.7.2. *Given a regular relation R over m-ary tuples of strings, and an ordered partition $\mathcal{P}$ of $\{1, \ldots, m\}$ with k blocks, there exist sets $\Theta_1, \ldots, \Theta_k$ of arithmetic progressions, which can be constructed in polynomial time, such that there is a set $J \subseteq \Theta_1 \times \ldots \Theta_k$ satisfying the following:*

$$\bar{s} \in R_{\text{len}} \quad \Leftrightarrow \quad \exists\, (\theta_1, \ldots, \theta_k) \in J : \; n_i \in \theta_i \textit{ for all } i \leq k,$$

where $\ell_{\mathcal{P}}(\bar{s}) = (n_1, \ldots, n_k)$. Moreover, given a tuple $(\theta_1, \ldots, \theta_k) \in \Theta_1, \ldots, \Theta_k$, one can check in polynomial time whether it belongs to J.

Note that each of the claims implies Lemma 6.6. We now prove Theorem 6.7 using these claims.

Let $G = (V, E)$ be a Σ-labeled graph database, and Q an ECRPQ in which all relations R have been replaced by R_{len}, i.e., a query of the form $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j R^j_{\text{len}}(\bar{\pi}_j)$. Given a tuple of nodes and a tuple of paths, we need to check whether they belong to the output of the query. The first step of an NP (combined complexity) algorithm is of course to guess the witness nodes. Thus, we need to show how to check, for tuples $\bar{a}, \bar{b}, \bar{a}', \bar{b}'$ of nodes with $|\bar{a}| = |\bar{b}|$ and $|\bar{a}'| = |\bar{b}'|$ as well as a tuple of paths $\bar{\rho}$ with $|\bar{\rho}| = |\bar{a}'|$ whether $\bigwedge_i (a_i, \pi_i, b_i), \bigwedge_{i'} (a'_i, \rho_i, b'_i), \bigwedge_j R^j_{\text{len}}(\bar{\pi}_j)$ holds for some tuple of paths $\bar{\pi}$. Since the second conjunct is trivially verifiable in polynomial time, our goal is thus to show how to check the existence of a tuple of paths $\bar{\pi}$ so that

$$\bigwedge_{i \leq p} (a_i, \pi_i, b_i), \bigwedge_{j \leq t} R^j_{\text{len}}(\bar{\pi}_j)$$

holds. Here each $\bar{\pi}_j$ is a tuple of paths among the π_i's and those in $\bar{\rho}$.

For this, we proceed as follows. We guess an ordered partition $\mathcal{P}$ of the set of indices of $\bar{\pi}$ such that $\bar{\rho}$ length-conforms to $\mathcal{P}$. Next, for each of the relations R^j_{len}, we compute, as in Claim 6.7.2, the sets Θ_i's of arithmetic progressions and guess the set J that will witness membership in R^j_{len} for the given ordered partition. We then verify in polynomial time that each guessed tuple of arithmetic progressions belongs to J. Finally, for each of the paths we use, we guess the points on that path whose distance from the starting point of the path are the same as the length of the paths of smaller length. For example, if we have 4 paths $\pi_1, \ldots, \pi_4$ and an ordered partition $\Pi = \{\{1, 2\}, \{3\}, \{4\}\}$, we would guess, for a subgoal (a, π_4, b), nodes a' and a'' so that the distance from a to a' will be the same as the lengths of π_1 and π_2, and the distance from a to a'' will be the same as the length of π_3. Now it remains to check that we can find paths $\bar{\pi}$ that satisfy these guesses.

But now it follows from Claim 6.7.2 that the latter can be solved if we can solve the following problem: given nodes $a_1, \ldots, a_n$ in a graph database, arithmetic progressions θ_{ij} for $i, j \leq n$, and a family $\mathcal{S}$ of subsets of $\{1, \ldots, n\} \times \{1, \ldots, n\}$, do there exist paths π_{ij} between a_i and a_j for $i, j \leq n$ so that $|\pi_{ij}| \in \theta_{ij}$ and whenever (i, j) and (i', j') are in the same set of $\mathcal{S}$, we have $|\pi_{ij}| = |\pi_{i'j'}|$? Indeed, by guessing intermediate points of paths, we eliminated the need for inequality comparisons between the length of paths.

To solve this problem, we view the graph database as unary automata $\mathcal{A}_{ij}$, for every $i, j \leq n$, where a_i is the initial state and a_j is the final state, and the transitions are the graph edges themselves. Each such automaton can be converted in polynomial time into a union of arithmetic progressions $c_k^{ij} + d_k^{ij}\mathbb{N}$ so that the lengths of paths between a_i and a_j are precisely the numbers that belong to one of the progressions. Then the problem above is solvable iff the following Presburger formula

$$\exists x_{11} \ldots \exists x_{nn} \left(\bigwedge_{i,j \le n} \big(\bigvee_k \exists n\, (c_k^{ij} + d^{ij} kn = x_{ij}) \wedge \theta_{ij}(x_{ij}) \big) \wedge \bigwedge_{S \in \mathcal{S}} \bigwedge_{(i,j),(i',j') \in S} x_{ij} = x_{i'j'} \right) \quad (6)$$

is satisfiable. Indeed, the witnesses x_{ij}'s then are the lengths of the witness paths between a_i and a_j, for $i, j \le n$. In (6), for an arithmetic progression $\theta = a + b\mathbb{N}$, where a and b are integer constants, by $\theta(x)$ we mean the formula $\exists n(x = a + bn)$.

As (6) is an existential Presburger formula, its satisfaction can be checked in NP. In fact one can guess, for each i, j, a particular arithmetic progression $c_k^{ij} + d_k^{ij}\mathbb{N}$ that will witness the satisfaction of the formula, and then (6) is transformed into an instance of integer linear programming, for which witnesses of polynomial size can be guessed according to [Papadimitriou 1981].

It is straightforward to observe that all the guesses can be combined and the entire procedure runs in NP, thus completing the proof. □

It also follows from Theorem 6.7 that the combined complexity of queries Q_{len} (i.e., when the input contains Q_{len} rather than Q) is NP-complete as well.

Another standard way of creating an abstraction of R is to count the numbers of occurrences of symbols; this, however, leads to non-regular languages and relations (e.g., if we only count numbers of occurrences of letters, strings from $(ab)^*$ will be transformed into strings in which the number of a's equals the number of b's). Nonetheless, we can prove an NP upper bound for such nonregular relations; they will be considered in the next section when we look at extensions of ECRPQs.

Repetition of path variables In the definitions of CRPQs and ECRPQs, repetition of path variables is allowed in neither the relational parts nor the regular languages/relations. For instance, we cannot write $(x, \pi, y), (x', \pi, y')$, nor can we write $R_1(\bar{\omega}), R_2(\bar{\omega})$. We refer to these as relational repetitions and regular repetitions, respectively. What happens if these are allowed? It turns out that the complexity of ECRPQs is not affected, but the combined complexity of CRPQs jumps to that of ECRPQs, no matter what kind of repetition is allowed.

PROPOSITION 6.8. *The problem* ECRPQ-EVAL *remains in* PSPACE *for* ECRPQ*s with any kind of repetition, while the problem* CRPQ-EVAL *becomes* PSPACE*-complete even for Boolean acyclic CRPQs with either relational or regular repetitions.*

PROOF. The proof of Theorem 6.3 can be easily adapted to show that the problem ECRPQ-EVAL remains in PSPACE for ECRPQs with any kind of repetition. For the lower bound we use the same reduction from REI used in the proof of Theorem 6.3, but this time we construct the CRPQ

$$Ans() \leftarrow \bigwedge_{1 \le i \le m} (x_i, \pi, y_i), R_i(\pi),$$

that repeats path variables. □

7. QUERY CONTAINMENT

The task of checking query containment is crucial for problems such as query optimization and data integration. The problem of query containment for CRPQs was first introduced in [Florescu et al. 1998] which showed an EXPSPACE upper bound. A matching lower bound was then shown in [Calvanese et al. 2000]. Here we study the problem of query containment for ECRPQs, i.e., checking for two ECRPQs Q and Q' over Σ, whether $Q(G) \subseteq Q'(G)$ for every Σ-labeled graph database G.

It is well-known that the containment and equivalence problem are undecidable for rational relations on words, i.e., relations defined by *asynchronous* transducers [Berstel 1979]. However, in the context of ECRPQs we cannot use this fact to establish the undecidability of containment because regular relations used in ECRPQs are synchronous. In fact, for regular relations, containment is decidable. Hence in order to prove Theorem 7.1 below, which states that containment for ECRPQs is undecidable, we have to use different techniques. In fact, we have indicated in Section 4 how string patterns (with variables) can be coded by ECRPQs. Combining this with a recent result on the undecidability of pattern containment [Freydenberger and Reidenbach 2010], we obtain the following.

THEOREM 7.1. *There exists a fixed finite alphabet Σ, such that the containment problem for* ECRPQ*s over Σ is undecidable.*

PROOF. It has been recently shown in [Freydenberger and Reidenbach 2010] that there is a finite alphabet Σ such that the problem of checking whether $L_\Sigma(\alpha) \subseteq L_\Sigma(\beta)$, for arbitrary patterns α and β over Σ, is undecidable. We reduce this problem to the problem of query containment for ECRPQs over a fixed alphabet.

Let $\alpha = \alpha_1 \cdots \alpha_n$ and $\beta = \beta_1 \cdots \beta_m$ be patterns over Σ; that is, each α_i and β_j ($1 \leq i \leq n$, $1 \leq j \leq m$) is an element of $\Sigma \cup \mathcal{V}$ (where $\mathcal{V}$ is a set of variables). Let Q_α and Q_β be ECRPQs for α and β as constructed in Section 4. Assume without loss of generality that the relational parts of α and β are of the form $\bigwedge_{1\leq i\leq n}(x_{i-1}, \pi_i, x_i)$ and $\bigwedge_{1\leq j\leq m}(y_{j-1}, \pi'_j, y_i)$, respectively.

Let p and p' be symbols not in $\Sigma \cup \mathcal{V}$. Define Σ' as $\Sigma \cup \{p, p'\}$. Further, define Q'_α as the ECRPQ over Σ' that is obtained from Q_α by (1) extending its relational part with the atoms (x_{init}, π_0, x_0) and $(x_n, \pi_{n+1}, x_{end})$, and (2) extending its body with the atoms $p(\pi_0)$ and $p'(\pi_{n+1})$. In the same way, we define Q'_β as the ECRPQ over Σ' that is obtained from Q_β by (1) extending its relational part with the atoms (y_{init}, π'_0, y_0) and $(y_m, \pi'_{m+1}, y_{end})$, and (2) extending its body with the atoms $p(\pi'_0)$ and $p'(\pi'_{m+1})$. We prove next that $\alpha \subseteq \beta$ iff $Q'_\alpha \subseteq Q'_\beta$.

Indeed, assume first that $Q'_\alpha \subseteq Q'_\beta$. Let $\bar{w} = a_1 \cdots a_q$ be a string over Σ that belongs to $L_\Sigma(\alpha)$. Let G be the Σ'-labeled graph database such that (1) its set of nodes is $\{v_{init}, v_0, v_1, \ldots, v_q, v_{end}\}$, (2) there is an edge from v_{init} to v_1 labeled p, (3) there is an edge from v_p to v_{end} labeled p', (4) there is an edge from v_{i-1} to v_i ($1 \leq i \leq q$) labeled a_i, and (5) those are the only edges in G. Notice that there is a unique path in G from v_0 to v_q, and the label of that path is precisely $\bar{w}$.

Clearly, $Q'_\alpha(G) = \texttt{true}$, and thus, $Q'_\beta(G) = \texttt{true}$. Notice that for every assignment σ from the node variables of Q'_β to the nodes of G and for every assignment μ from the path variables in Q'_β to the paths in G such that (σ, μ) satisfies Q'_β, it must be the case that (1) $\sigma(y_{init}) = v_{init}$, $\sigma(y_0) = v_0$, $\sigma(y_m) = v_q$, and $\sigma(y_{end}) = v_{end}$, and (2) if $\rho = \mu(\pi'_1) \cdots \mu(\pi'_m)$ then $\rho = v_0 a_1 v_1 \cdots v_{p-1} a_q v_q$ and $\lambda(\rho) = \bar{w}$. This shows that $\bar{w} \in L_\Sigma(\beta)$. We conclude that $\alpha \subseteq \beta$.

Assume, on the other hand, that $\alpha \subseteq \beta$. Let G be an arbitrary Σ'-labeled graph database such that $Q'_\alpha(G) = \texttt{true}$. Assume that σ is an assignment from the node variables of Q'_α to the nodes of G and μ is an assignment from the path variables in Q'_α to the paths in G such that (σ, μ) satisfies Q'_α. Let $v = \sigma(x_0)$ and $v' = \sigma(x_n)$. Then there exists a path ρ in G from v to v' such that $\lambda(\rho) \in L_\Sigma(\alpha)$. Then $\lambda(\rho) \in L_\Sigma(\beta)$. We conclude that $Q'_\beta(G) = \texttt{true}$, and thus, that $Q'_\alpha \subseteq Q'_\beta$. This concludes the proof of the theorem. □

It follows from [Freydenberger and Schweikardt 2011] that the problem of containment of a CRPQ in an ECRPQ remains undecidable. We can recover decidability in the opposite case. The problem of containment of an ECRPQ in a CRPQ is the problem of checking whether $Q(G) \subseteq Q'(G)$ for every Σ-labeled graph database G, where Q is an ECRPQ and Q' is a CRPQ over Σ. We can adapt proof techniques from [Calvanese et al. 2000] to show that the above problem has the same complexity as CRPQ containment.

THEOREM 7.2. *The problem of checking containment of an ECRPQ in a CRPQ is* EXPSPACE-*complete.*

PROOF. Hardness follows from [Calvanese et al. 2000], since the problem of query containment is EXPSPACE-hard already for CRPQs. Next we prove membership. In order to prove that Q is not contained in Q' we look for a counterexample along the lines of how it is done in [Calvanese et al. 2000; Florescu et al. 1998]. That is, we look for a labeled graph database G such that $Q(G) \not\subseteq Q'(G)$. But as opposed to [Calvanese et al. 2000], which coded counterexamples as strings accepted by usual automata, we code counterexamples as strings accepted by automata that accept k-ary regular relations, where k depends on Q only. This is because the query Q may contain regular relations of arity bigger than 1 in its body, and, thus, in most of the cases no standard automaton will be capable of keeping track of the relationships between paths that are required to satisfy Q.

As in the case of [Calvanese et al. 2000; Florescu et al. 1998], it is useful first to give a semantic characterization of containment between ECRPQs in terms of the notion of *canonical* graphs. Let Q be an ECRPQ of the form: $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, \pi_i, y_i), \bigwedge_{1 \leq j \leq t} R_j(\bar{\omega}_j)$, and assume that each R_j ($1 \leq j \leq t$) is a regular language that represents an n_j-ary regular relation. Let G be a graph database and σ an assignment of the node variables in Q into G. Then G is σ-*canonical* for Q if:

— G consists of m simple paths, one for each conjunct in the relational part of Q, which are node and edge disjoint, i.e. only the start and end nodes can be shared between different paths;
— for each $1 \leq j \leq m$, if ρ_j is the path associated with the atom (x_j, π_j, y_j) then ρ_j starts at the node $\sigma(x_j)$ and ends at the node $\sigma(y_j)$; and
— for each $1 \leq j \leq t$, if $\bar{\omega}_j = \pi_{j_1}, \ldots, \pi_{j_{n_j}}$ ($1 \leq j_\ell \leq m$, for each $1 \leq \ell \leq n_j$) then $[\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_{n_j}})]$ satisfies R_j.

Clearly, if G is σ-canonical for Q for some assignment σ, and $\bar{\chi} = (\pi_{\ell_1}, \ldots, \pi_{\ell_p})$, where each π_{ℓ_i} belongs to $\{\pi_1, \ldots, \pi_m\}$, then $(\sigma(\bar{z}), \rho_{\ell_1}, \ldots, \rho_{\ell_p})$ belongs to $Q(G)$.

Let Q' be an ECRPQ of the form $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq m'} (u_i, \pi'_i, v_i), \bigwedge_{1 \leq j \leq t'} S_j(\bar{\omega}'_j)$ over the same alphabet as Q, and assume that each S_j ($1 \leq j \leq t'$) is a regular language that represents a p_j-ary regular relation, $p_j > 0$. Notice that we assume that Q' has the same node and path variables in the head as Q (i.e. it has the same *free* variables as Q). Further, we assume without loss of generality that the set of non-free variables of Q and Q' respectively are disjoint.

Let G be a graph database that is σ-canonical for Q, for some σ. Further, let σ' be a mapping from the node variables of Q' into the nodes of G and μ be a mapping from the path variables of Q' into the paths of G. Then (σ', μ) is a (Q', G, σ)-*mapping*, if the following hold:

— For each $z \in \bar{z}$, we have that $\sigma(z) = \sigma'(z)$;
— for each $\chi \in \bar{\chi}$, if $\chi = \pi_i$ ($1 \leq i \leq m$) then $\mu(\chi) = \rho_i$;
— for each $1 \leq j \leq m'$, $\mu(\pi'_j)$ is a path from $\sigma'(u_j)$ to $\sigma'(v_j)$; and

— for each $1 \leq j \leq t'$, if $\bar{\omega}'_j = \pi'_{j_1}, \ldots, \pi'_{j_{p_j}}$ ($1 \leq j_\ell \leq m$, for each $1 \leq \ell \leq p_j$), then $[\lambda(\mu(\pi'_{j_1})), \ldots, \lambda(\mu(\pi'_{j_{p_j}}))]$ satisfies S_j.

Using essentially the same techniques as in [Calvanese et al. 2000; Florescu et al. 1998] it is possible to give the following semantic characterization of ECRPQ containment:

CLAIM 7.2.1. *Let Q and Q' be two ECRPQs over the same alphabet Σ. Then Q is not contained in Q' if and only if there exists a Σ-labeled graph database G and an assignment σ from the variables of Q to the nodes of G, such that G is σ-canonical for Q and no (Q', G, σ)-mapping exists.*

The key idea used in [Calvanese et al. 2000] to prove that containment of CRPQs is in EXPSPACE is as follows. Given CRPQs Q and Q', first one codes each graph database G that is σ-canonical for Q (for some σ) as a string over some extended alphabet of exponential size, and then constructs an NFA $\mathcal{A}$ that accepts exactly those strings over this extended alphabet that correspond to those codifications of σ-canonical graphs G for Q for which no (Q', G, σ)-mapping exists. The NFA $\mathcal{A}$ is double exponential in the size of the input, i.e. the pair (Q, Q'). Thus, checking whether Q is not contained in Q' is equivalent to checking whether the language accepted by $\mathcal{A}$ is nonempty, which can be done in EXPSPACE by a typical “on-the fly” simulation of $\mathcal{A}$.

In more detail, in order to generate candidate counterexamples to the fact that the CRPQ $Q := Ans(\bar{z}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i, \pi_i, y_i), \bigwedge_{1\leq j\leq m} L_j(\omega_j)$ is contained in the CRPQ Q', Calvanese et al. [2000] construct an NFA $\mathcal{A}_1$ that accepts strings of the form $\$d_1w_1e_1\$d_2w_2e_2\$\cdots\$d_mw_me_m\$$, where m is the number of path variables in Q, that represent graph databases that are σ-canonical for Q, for some assignment σ. Each d_i and e_i ($1 \leq i \leq m$) is a set of node variables from Q. To construct $\mathcal{A}_1$ it is necessary to define:

— An NFA that accepts all strings of the form above, such that for each $i \in [1, m]$, $x_i \in d_i$, $y_i \in e_i$, and for each $1 \leq j \leq m$ it is the case that w_j belongs to L_j.
— An NFA that checks that the distinct symbols d_i and e_i appearing in the string form a partition of the variables of Q.

Then $\mathcal{A}_1$ is defined as the intersection of these two NFAs.

Finally, $\mathcal{A}_1$ is extended to an NFA $\mathcal{A}$ that accepts those strings w of the form above, that “code” a graph G_w that is σ_w-canonical for Q and such that no (Q', G_w, σ_w)-mapping exists. The size of $\mathcal{A}$ is double exponential in $|Q| + |Q'|$. It is shown that Q is not contained in Q' iff $\mathcal{A}$ accepts at least some string.

With essentially the same techniques we can prove Theorem 7.2. Let Q be an ECRPQ and Q' be a CRPQ over the same alphabet. Assume that m is the number of relational atoms in Q. Let Q_1 be the restriction of Q to the regular relation atoms of arity at most 1. We construct, using the techniques in [Calvanese et al. 2000] mentioned above, a double exponential NFA $\mathcal{A}$ that accepts at least some string iff Q_1 is not contained in Q'. As we explained above, the NFA $\mathcal{A}$ accepts strings of the form $\$d_1w_1e_1\$d_2w_2e_2\$\cdots\$d_mw_me_m\$$. It is then easy to construct an m-ary letter-to-letter NFA $\mathcal{A}_m$ of double exponential size, that accepts exactly the strings $(d_1w_1e_1, d_2w_2e_2, \ldots, d_mw_me_m)$ such that $\mathcal{A}$ accepts $\$d_1w_1e_1\$d_2w_2e_2\$\cdots\$d_mw_me_m\$$. We then compute the intersection $\mathcal{A}_2$ of $\mathcal{A}$ with an NFA $\mathcal{A}_Q$ that checks that, for each regular relation atom in Q of the form $R(\pi_{i_1}, \ldots, \pi_{i_n})$, such that R is of arity > 1, the tuple $(w_{i_1}, \ldots, w_{i_n})$ belongs to R. The size of $\mathcal{A}_Q$ is exponential, and thus, the size of $\mathcal{A}_2$ is double exponential. Further, it can easily be proved that Q is not contained in Q'

iff $\mathcal{A}_2$ accepts at least some string. This can be done in EXPSPACE by an "on-the-fly" construction of $\mathcal{A}_2$. □

8. EXTENSIONS

We now look at various ways of going beyond the class of ECRPQs. First, we consider an analog of relational calculus by adding negation and arbitrary quantification to the language. After that, we look at conjunctive queries with non-regular relations; we handle linear constraints on lengths of paths, and Parikh-image constraints.

8.1. Adding negation

We now investigate the query evaluation problem for the extension of ECRPQs with negation. Formally, we define the language $\mathrm{ECRPQ}^{\neg}$ over alphabet Σ as the set of formulas described by the following grammar:

$$\begin{aligned} \mathrm{atom} &:= \pi_1 = \pi_2 \mid x = y \mid (x, \pi, y) \mid R(\pi_1, \dots, \pi_n) \\ \varphi, \psi &:= \mathrm{atom} \mid \neg\varphi \mid \varphi \wedge \psi \mid \exists x \varphi \ \mid \ \exists \pi \varphi \end{aligned}$$

Here x, y range over the set of node variables, $\pi, \pi_1, \dots$ range over the set of path variables, and R ranges over the set of regular expressions over alphabets $(\Sigma_\perp)^n$ ($n > 0$) that represent n-ary regular relations over Σ. The language $\mathrm{CRPQ}^{\neg}$ is defined as the restriction of $\mathrm{ECRPQ}^{\neg}$ to formulas that only make use of regular languages. The notions of free and bound variables are standard; we write $\varphi(\bar{x}, \bar{\pi})$ to list free node and path variables explicitly.

The semantics of $\mathrm{ECRPQ}^{\neg}$ is defined in the standard way. Given a graph database $G = (V, E)$, a mapping σ from the set of free node variables of φ into V, and a mapping μ from the set of free path variables of φ into the set of paths in G, the notion $(G, \sigma, \mu) \models \varphi$ is defined just as for ECRPQs with the following additional rules:

- the Boolean connectives $\neg$ and $\vee$ have the standard semantics;
- $(G, \sigma, \mu) \models \exists x \varphi$ iff there exists $v \in V$ such that $(G, \sigma_{x \to v}, \mu) \models \varphi$, where $\sigma_{x \to v}$ extends the assignment σ by letting $\sigma(x) = v$;
- $(G, \sigma, \mu) \models \exists \pi \varphi$ iff there exists a path ρ in G such that $(G, \sigma, \mu_{\pi \to \rho}) \models \varphi$, where $\mu_{\pi \to \rho}$ extends the assignment μ by letting $\mu(\pi) = \rho$.

Given a graph database G and an $\mathrm{ECRPQ}^{\neg}$ formula $\varphi(\bar{x}, \bar{\pi})$, we let $\varphi(G)$ be the set of tuples $(\bar{v}, \bar{\rho})$ such that $(G, \sigma, \mu) \models \varphi$, where $\sigma(\bar{x}) = \bar{v}$ and $\mu(\bar{\pi}) = \bar{\rho}$.

Notice that $\mathrm{ECRPQ}^{\neg}$ and $\mathrm{CRPQ}^{\neg}$ express nontrivial properties of graph databases that are not expressible by means of ECRPQs. For instance, the query $\neg\exists\pi\,((x, \pi, y) \wedge L(\pi))$ defines the set of all pairs (a, b) of nodes such that no path between them is labeled by a string from language L.

Combined complexity. The problems $\mathrm{ECRPQ}^{\neg}$-EVAL and $\mathrm{CRPQ}^{\neg}$-EVAL are defined exactly as the query evaluation problems in Section 6.2, except that the input query is from the extended language. Again we see a significant difference between regular languages and regular relations in queries. The complexity jumps in both cases, but while $\mathrm{CRPQ}^{\neg}$-EVAL can be solved in single-exponential time, $\mathrm{ECRPQ}^{\neg}$ queries cannot be evaluated in time bounded by a fixed stack of exponents.

THEOREM 8.1.

- *The problem* $\mathrm{CRPQ}^{\neg}$-EVAL *is* PSPACE-*complete.*
- *The problem* $\mathrm{ECRPQ}^{\neg}$-EVAL *is decidable, but non-elementary.*

PROOF. We start by proving the first part of the theorem. Hardness follows from Proposition 6.8 since the language $\mathrm{CRPQ}^{\neg}$ allows for path variable repetition. Next we prove membership.

Let $G = (V, E)$ be a Σ-labeled graph database and $\varphi(\bar{x}, \bar{\pi})$ be a $\mathrm{CRPQ}^{\neg}$ formula. Further, let $\bar{v}$ be a tuple of nodes in G such that $|\bar{v}| = |\bar{x}|$ and $\bar{\rho}$ be a tuple of paths in G such that $|\bar{\rho}| = |\bar{\pi}|$. Further, assume that $L_1, \dots, L_m$ are all the regular expressions mentioned in φ. Our goal is to define a PSPACE procedure that checks whether $(\bar{v}, \bar{\rho}) \in \varphi(G)$. In order to do that, we first have to introduce some new terminology.

Let τ be a first-order (FO) vocabulary $\langle Nodes, Paths, Endpoints, L_1, \dots, L_m \rangle$, where $Nodes$, $Paths$, and L_i ($1 \leq i \leq m$) are unary relation symbols, and $Endpoints$ is a ternary relation symbol. We define, from G, an FO structure $\mathcal{M}_G$ over τ as follows: The domain of $\mathcal{M}_G$ is the disjoint union of V and all the paths that belong to G. (Notice that each node in V is also a path in G, but here we consider them to be different objects. That is, each $v \in V$ appears separately as a node and a path in the domain of $\mathcal{M}_G$). The interpretation of $Nodes$ in $\mathcal{M}_G$ contains all those elements of the domain that are nodes. The interpretation of $Paths$ in $\mathcal{M}_G$ contains all those elements of the domain that are paths. The interpretation of the ternary relation $Endpoints$ contains all tuples (v, ρ, v') such that ρ is a path in G from node v to node v'. Finally, the interpretation of the symbol L_i ($1 \leq i \leq m$) contains all those paths in G whose label satisfies the regular expression L_i.

Let φ_τ be the FO formula over vocabulary τ obtained from φ by simultaneously replacing (1) each subformula of the form $\exists x \theta$ (for x a node variable) with $\exists x (Nodes(x) \wedge \theta)$, (2) each subformula of the form $\exists \pi \theta$ (for π a path variable) with $\exists \pi (Paths(\pi) \wedge \theta)$, and (3) each atomic formula of the form (x, π, y) with $(Nodes(x) \wedge Nodes(y) \wedge Paths(\pi) \wedge Endpoints(x, \pi, y))$.

Clearly, $(\bar{v}, \bar{\rho}) \in \varphi(G)$ iff $(\bar{v}, \bar{\rho})$ belongs to the evaluation of φ_τ over $\mathcal{M}_G$.

Of course, $\mathcal{M}_G$ cannot be effectively constructed from G since the set of paths in G is potentially infinite, and, thus, $\mathcal{M}_G$ is also potentially infinite. However, it is possible to prove that there exists a finite substructure $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ of $\mathcal{M}_G$ such that $(\bar{v}, \bar{\rho})$ belongs to the evaluation of φ_τ over $\mathcal{M}_G$ iff it belongs to the evaluation of φ_τ over $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$. We show how to define $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ next.

Assume that the *quantifier rank* of φ_τ is $k \geq 0$, where as usual the quantifier rank of an FO formula θ is the depth of nested quantification in θ. Let $\mathcal{L} \subseteq \{L_1, \dots, L_m\}$. We say that a path in G *satisfies* $\mathcal{L}$ if the label of π satisfies $\mathcal{L}$, for each $L \in \mathcal{L}$, and does not satisfy L', for each $L' \in \{L_1, \dots, L_m\} \setminus \mathcal{L}$. (Notice that for each path in G there is one, and only one, subset $\mathcal{L}$ of $\{L_1, \dots, L_m\}$ that it satisfies.) For each pair (v, v') of nodes in V, and for every $\mathcal{L} \subseteq \{L_1, \dots, L_m\}$, let $c_{\mathcal{L},v,v'} \geq 0$ be the minimum between $k + |\bar{\rho}|$ and the number of paths in G that go from v to v' and satisfy $\mathcal{L}$. We arbitrarily pick, for each pair (v, v') of nodes in V and for each $\mathcal{L} \subseteq \{L_1, \dots, L_m\}$, $c_{\mathcal{L},v,v'}$ distinct paths $\rho^1_{\mathcal{L},v,v'}, \dots, \rho^{c_{\mathcal{L},v,v'}}_{\mathcal{L},v,v'}$ from v to v' that satisfy $\mathcal{L}$.

We define the structure $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ as follows: Its domain contains all the nodes of V, each path ρ that belongs to the tuple $\bar{\rho}$, and every path of the form $\rho^i_{\mathcal{L},v,v'}$, where $\mathcal{L} \subseteq \{L_1, \dots, L_m\}$, $v, v' \in V$ and $1 \leq i \leq c_{\mathcal{L},v,v'}$. The interpretation of $Nodes$ in $\mathcal{M}'_{G,\bar{t},\bar{\rho}}$ contains all nodes in the domain. The interpretation of $Paths$ in $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ contains all those elements of the domain that are paths. The interpretation of the ternary relation $Endpoints$ contain all tuples of the form (v, ρ, v'), where $v, v' \in V$ and ρ is a path in the domain that goes from v to v' in G. Finally, the interpretation of L_i ($1 \leq i \leq m$) in $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ contains all those paths in the domain of $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ for which its label satisfies L_i.

By using a standard Ehrenfeucht-Fraïssé argument it is possible to prove the following:

CLAIM 8.1.1. *The structures* $(\mathcal{M}_G, \bar{v}, \bar{\rho})$ *and* $(\mathcal{M}'_{G,\bar{v},\bar{\rho}}, \bar{v}, \bar{\rho})$ *are indistinguishable by* FO *sentences of quantifier rank* $\leq k$.

Proof (Sketch): We show that the duplicator has a winning strategy in the k-round Ehrenfeucht-Fraïssé game played on $(\mathcal{M}_G, \bar{v}, \bar{\rho})$ and $(\mathcal{M}'_{G,\bar{v},\bar{\rho}}, \bar{v}, \bar{\rho})$. The duplicator's response to a spoiler move in round $i \leq k$ is (inductively) defined as follows (we assume without loss of generality that the spoiler never repeats moves, i.e. in no round does the spoiler choose an element that has already been chosen by either player in previous rounds):

— If the spoiler's move in round i is a node in either of the two structures, then the duplicator responds by mimicking the spoiler's move on the other structure;

— if the spoiler's move in round i is a path ρ in $\bar{\rho}$ in either of the two structures, then again the duplicator responds by mimicking the spoiler's move on the other structure;

— if the spoiler plays a path ρ from node v to v', in either of the two structures, such that ρ satisfies $\mathcal{L} \subseteq \{L_1, \ldots, L_m\}$ and ρ is not a path in $\bar{\rho}$, then the duplicator responds with any path from v to v' in the other structure that (1) satisfies $\mathcal{L}$, (2) does not belong to $\bar{\rho}$, and (3) has not been previously chosen in the game. Notice that it is always possible for the duplicator to choose such a path, since for each pair of nodes $v, v' \in V$ and for each $\mathcal{L} \subseteq \{L_1, \ldots, L_m\}$, the number of paths from v to v' that satisfy $\mathcal{L}$ and that do not belong to $\bar{\rho}$ is the same up to k.

It is not hard to see that duplicator's response defined in this way always preserves a partial isomorphism between the two structures. This implies that the duplicator has a winning strategy in the k-round Ehrenfeucht-Fraïssé game played on $(\mathcal{M}_G, \bar{v}, \bar{\rho})$ and $(\mathcal{M}'_{G,\bar{v},\bar{\rho}}, \bar{v}, \bar{\rho})$, and, thus, by well-known results, that the structures are indistinguishable by FO sentences of quantifier rank $\leq k$. □

The previous claim shows that $(\bar{v}, \bar{\rho}) \in \varphi(G)$ if and only if $(\bar{v}, \bar{\rho})$ belongs to the evaluation of φ_τ over $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$. Thus, a straightforward approach to check whether $(\bar{v}, \bar{\rho}) \in \varphi(G)$ would be to construct $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ and then evaluate φ_τ over it. The problem with this approach is that $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ could be of exponential size, and, thus, impossible to construct in polynomial space. It will be necessary to follow a different approach then.

Assume that φ_τ is given in prenex normal form, i.e. φ_τ is of the form $Q_1 y_1 \cdots Q_m y_m \, \alpha(\bar{x}, \bar{\pi}, y_1, \ldots, y_m)$, where each Q_i is either $\exists$ or $\forall$ and α is quantifier-free (if φ_τ is not in prenex normal form, we can convert it in polynomial time into an equivalent formula in prenex normal form). We follow a usual PSPACE argument to evaluate FO formulas on structures. The main problem with this is that some of the elements in $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ are paths and have to be treated as such. Thus, it is necessary to define a way of coding paths in polynomial space.

In our case, each path will be coded with an *address*, that is a string over a finite alphabet. The address of a path ρ can be intuitively explained as follows:

— It starts with a new symbol p, that states that this is the address of a path;

— the address continues with the encodings of the two endpoints v and v' of the path (separated with some delimiter); this part of the address uses $O(\log_2 |V|)$ space;

— afterwards, the address contains an encoding of the subset $\mathcal{L}$ of $\{L_1, \ldots, L_m\}$ that ρ satisfies; this encoding is a string of length m over alphabet $\{0, 1\}$ such that its i-th symbol is 1 iff $L_i \in \mathcal{L}$;

— then the address contains an encoding of the integer $i \leq k + |\bar{\rho}|$ such that $\rho = \rho^i_{\mathcal{L},v,v'}$; this encoding uses $O(\log_2 (|\varphi| + |\bar{\rho}|))$ space.

Clearly, the address of a path defined in this way can be specified using at most polynomial space.

We show next how the problem of checking whether $(\bar{v}, \bar{\rho})$ belongs to the evaluation of φ_τ over $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$ can be solved in polynomial time by an alternating Turing machine. This will finish the proof of the theorem, since the class of problems that can be solved in polynomial time by alternating Turing machines coincides with the class of problems that can be solved in PSPACE.

The alternating machine proceeds as follows. It first replaces in φ_τ each variable x in $\bar{x}$ with the encoding of the corresponding node v of $\bar{v}$. It then replaces each variable π in $\bar{\pi}$ with the encoding (address) of the corresponding path ρ in $\bar{\rho}$. Then the machine reads the formula φ_τ from left-to-right. Each time it encounters an existential quantifier $\exists y_i$ it enters an existential state, and each time it encounters a universal quantifier $\forall y_i$ it enters a universal state. In each case, the machine "guesses" the interpretation of y_i as the encoding of a node or a path $c(y_i)$ in the domain. Finally, the machine verifies that $\alpha(\bar{v}, \bar{\rho}, c(y_1), \ldots, c(y_m))$ holds, and if that is the case it accepts. We show next that the latter can be done in polynomial time. Notice that this implies that the whole process can be performed in polynomial time.

We start with the case of the atomic formulas in α. In order to check whether the element assigned to a variable belongs to the interpretation of $Nodes$ in $\mathcal{M}'_{G,\bar{v},\bar{\rho}}$, we only have to check that the encoding of this element does not start with a p. In order to check whether the element belongs to the interpretation of $Paths$, it is sufficient to check that its encoding starts with a p. In order to check whether the elements a, b, c assigned to variables x, π, y, respectively, are such that (a, b, c) belongs to the interpretation of $Endpoints$, we only have to check that b is the encoding of a path, a and c are encodings of nodes, and that b is a path from a to c. Finally, in order to check whether the element a assigned to a variable belongs to the interpretation of L_i $(1 \leq i \leq m)$, we only have to check that a is a path (i.e. its encoding starts with p) and the bit that corresponds to L_i in the encoding of the subset $\mathcal{L}$ of $\{L_1, \ldots, L_m\}$ that satisfies a is set to 1.

Thus, the value of the atomic formulas involved in $\alpha(\bar{v}, \bar{\rho}, c(y_1), \ldots, c(y_m))$ can be computed in polynomial time. But α is a polynomial size Boolean combination of atomic formulas, and, thus, the value of $\alpha(\bar{v}, \bar{\rho}, c(y_1), \ldots, c(y_m))$ can be computed in polynomial time from the values of the atomic formulas. We conclude that computing the value of $\alpha(\bar{v}, \bar{\rho}, c(y_1), \ldots, c(y_m))$ can be done in polynomial time.

There is, however, one small issue that requires explanation in order for the previous procedure to work properly. Assume that the procedure "guesses" the interpretation of a variable y_i in φ_τ to be the encoding of a path in G from v to v' that satisfies $\mathcal{L} \subseteq \{L_1, \ldots, L_m\}$. Then it is necessary to check that, if the encoding implies that this path is $\rho^i_{\mathcal{L},v,v'}$, then $i \leq c_{\mathcal{L},v,v'}$. In order to do so, the procedure needs to check, in a subroutine, whether there exist i different paths from v to v' that satisfy $\mathcal{L}$. The next claim shows that this can be done in polynomial space, which finishes the proof of the theorem.

CLAIM 8.1.2. *For each pair $v, v' \in V$, subset $\mathcal{L}$ of $\{L_1, \ldots, L_m\}$ and $i \leq k + |\bar{\rho}|$, one can check in* PSPACE *whether there exist i distinct paths in $G = (V, E)$ from v to v' that satisfy $\mathcal{L}$.*

Proof (Sketch): First, let $\mathcal{A}_{v,v'}$ be the automaton over alphabet $\{v\} \cup (\Sigma_\perp \times V)$ defined as follows. The set of states is the disjoint union of V with a new state s. The initial state of $\mathcal{A}$ is s and the final state is v'. Further, the transition relation of $\mathcal{A}$ is defined as follows: (1) For every edge $(v_1, a, v_2) \in E$ there is a transition in $\mathcal{A}$ from v_1 to v_2 labeled

(a, v_2), (2) for every node $v_1 \in V$ there is a transition from v_1 to v_1 in $\mathcal{A}$ labeled $(\perp, v_1)$, and (3) there is a transition in $\mathcal{A}$ from s to v labeled v. Intuitively, $\mathcal{A}_{v,v'}$ accepts exactly those strings of the form $v(a_1, v_1)(a_2, v_2) \cdots (a, v')$ such that $va_1v_1a_2v_2 \cdots av'$ is a path in G from v to v', when we allow paths to loop arbitrarily many times on $\perp$-labeled nodes.

Let $\mathcal{A}^i_{v,v'}$ be the automaton over alphabet $\{v^i\} \cup (\Sigma_\perp \times V)^i$ defined as follows: The set of states is $V^i \cup \{s^i\}$, the initial state is s^i and the final state is $(v')^i$. There is a transition in $\mathcal{A}^i$ from $\bar{u} = (u_1, \dots, u_i)$ to $\bar{w} = (w_1, \dots, w_i)$ labeled $\bar{t} = (t_1, \dots, t_i)$ iff there is a transition labeled t_ℓ from u_ℓ to w_ℓ in $\mathcal{A}_{v,v'}$, for each $1 \leq \ell \leq i$. Clearly, $\mathcal{A}^i_{v,v'}$ is of exponential size but the size of each one of its states is polynomial. Further, it is decidable in polynomial time whether there exists a transition labeled $\bar{t}$ from state $\bar{u}$ to $\bar{w}$ in $\mathcal{A}^i_{v,v'}$.

Define now an automaton $\mathcal{A}'_{v,v'}$ that is the restriction of $\mathcal{A}^i_{v,v'}$ to those strings $v^i(w^1_1, \dots, w^1_i) \cdots (w^m_1, \dots, w^m_i)$ over alphabet $\{v^i\} \cup (\Sigma_\perp \times V)^i$ that satisfy the following:

- For each $1 \leq \ell \leq i$, if for some $1 \leq j < m$ it is the case that $w^j_\ell = (\perp, v_1)$, for some $v_1 \in V$, then for each $j < k \leq m$ it is the case that $w^k_\ell = (\perp, v_1)$.
- For each $1 \leq \ell, p \leq i$, if $\ell \neq p$ then the strings $vw^1_\ell \cdots w^m_\ell$ and $vw^1_p \cdots w^m_p$ over alphabet $\{v\} \cup (\Sigma_\perp \times V)$ are different.

The first condition says that each projection of a string accepted by $\mathcal{A}'_{v,v'}$ represents a path in G from v to v' that loops only on v' and only at the end of the path. The second condition ensures that any two distinct projections of a path accepted by $\mathcal{A}'_{v,v'}$ represent different strings.

It is not hard to prove that the language accepted by $\mathcal{A}'_{v,v'}$ is nonempty iff there exist i distinct paths in G from v to v'. Further, it is also not hard to see that $\mathcal{A}'_{v,v'}$ is of exponential size but the size of each one of its states is polynomial; and it is decidable in polynomial time whether there exists a transition labeled $\bar{t}$ from state $\bar{q}$ to state $\bar{q}'$ in $\mathcal{A}'_{v,v'}$.

Let $\mathcal{A}_\mathcal{L}$ be the automaton that accepts all those strings $\bar{w}$ over Σ such that $\bar{w}$ satisfies L, for each $L \in \mathcal{L}$, and does not satisfy L', for each $L' \in \{L_1, \dots, L_m\} \setminus \mathcal{L}$. Notice that the size of $\mathcal{A}_\mathcal{L}$ is exponential in the size of φ, but each one of the states of $\mathcal{A}_\mathcal{L}$ is polynomial in the size of φ. Further, it is decidable in polynomial time whether there is a transition labeled $\bar{a}$ from a state $\bar{r}$ to a state $\bar{r}'$ of $\mathcal{A}_\mathcal{L}$.

Let Q be the set of states of $\mathcal{A}'_{v,v'}$ and R be the set of states of $\mathcal{A}_\mathcal{L}$. We define a new automaton $\mathcal{A}^f_{v,v'}$ over alphabet $\{v^i\} \cup (\Sigma_\perp \times V)^i$ as follows: The set of states of $\mathcal{A}^f_{v,v'}$ is $Q \times R^i$, the initial state is $(q_0, r_0, \dots, r_0)$, where q_0 and r_0 are the initial states of $\mathcal{A}'_{v,v'}$ and $\mathcal{A}_\mathcal{L}$, respectively, and the final states are those of the form $(q_f, r_f, \dots, r_f)$, where q_f and r_f are final states of $\mathcal{A}'_{v,v'}$ and $\mathcal{A}_\mathcal{L}$, respectively. Further, the automaton $\mathcal{A}^f_{v,v'}$ has a transition from state $(q, r^1_1, \dots, r^1_i)$ to $(q', r^2_1, \dots, r^2_i)$ ($q, q' \in Q$ and $\{r^1_1, r^2_1, \dots, r^1_i, r^2_i\} \subseteq R$) labeled $\bar{w} = (w_1, \dots, w_i)$ iff (1) there is a transition from q to q' in $\mathcal{A}'_{v,v'}$ labeled $\bar{w}$, (2) if $\bar{w} = v^i$ then $r^1_j = r^2_j$, for each $1 \leq j \leq i$, and (3) if $\bar{w} = ((a_1, v_1), \dots, (a_i, v_i)) \in (\Sigma_\perp \times V)^i$, then the following holds: (a) if $a_j = \perp$ ($1 \leq j \leq i$) then $r^1_j = r^2_j$, and (b) if $a_j \neq \perp$ ($1 \leq j \leq i$) then there is a transition labeled a_j from r^1_j to r^2_j in $\mathcal{A}_\mathcal{L}$. Notice again that the size $\mathcal{A}^f_{v,v'}$ is exponential in the size of φ, but each one of the states of $\mathcal{A}^f_{v,v'}$ is polynomial in the size of φ. Further, it is decidable in polynomial time whether there is a transition labeled $\bar{w}$ from a state $(q, \bar{r})$ to a state $(q', \bar{r}')$ of $\mathcal{A}^f_{v,v'}$.

It is not hard to prove that the language accepted by $\mathcal{A}^f_{v,v'}$ is nonempty iff there exist i distinct paths in G from v to v' that satisfy $\mathcal{L}$. The former can be easily checked in polynomial space using a standard “on-the-fly” verification procedure. This finishes the proof of the claim. □

This completes the proof of the first part of the theorem.

Now we prove the second part of the theorem. We only prove decidability since the non-elementary lower bound will follow from the second part of Theorem 8.2. The argument uses essentially the same kind of proof techniques applied in Section 5 to construct a compact representation of the set of answers to an ECRPQ. That is, we show that for every $\mathrm{ECRPQ}^\neg$ formula one can construct an automaton that represents the set of “answers” to the formula for a given interpretation of the free node variables.

CLAIM 8.1.3. *Let $\varphi(\bar{x}, \bar{\pi})$ be an $\mathrm{ECRPQ}^\neg$ formula, $G = (V, E)$ a Σ-labeled graph database, and $\bar{v}$ be tuple of nodes in G such that $|\bar{v}| = |\bar{x}|$. It is possible to effectively construct an automaton $\mathcal{A}_\varphi^{(G,\bar{v})}$ over alphabet $V^{|\bar{\pi}|} \cup (\Sigma_\perp)^{|\bar{\pi}|}$ that accepts precisely the set $\varphi(G, \bar{v}) = \{\bar{\rho} \mid (\bar{v}, \bar{\rho}) \in \varphi(G)\}$.*

It is not hard to see that Claim 8.1.3 implies decidability of the $\mathrm{ECRPQ}^\neg$ evaluation problem. Indeed, an algorithm that checks whether $(\bar{v}, \bar{\rho})$ belongs to $\varphi(G)$ can proceed as follows: It first constructs $\mathcal{A}_\varphi^{(G,\bar{v})}$ from $G = (V, E)$, $\bar{v}$ and φ. Then it constructs the string $\bar{\omega}$ over alphabet $V^{|\bar{\pi}|} \cup (\Sigma_\perp)^{|\bar{\pi}|}$ that represents the tuple $\bar{\rho}$ (this can be done in polynomial time). Finally, the algorithm checks whether $\bar{\omega}$ is accepted by $\mathcal{A}_\varphi^{(G,\bar{v})}$. If the latter is the case, the procedure accepts.

Now we prove Claim 8.1.3. Let Σ be a finite alphabet, $G = (V, E)$ be a Σ-labeled graph database, $\varphi(\bar{x}, \bar{\pi})$ an $\mathrm{ECRPQ}^\neg$ formula over alphabet Σ and $\bar{v}$ a tuple of nodes in G such that $|\bar{v}| = |\bar{x}|$. We first show how to construct the automaton $\mathcal{A}_\varphi^{(G,\bar{v})}$ for the case when φ is an atom, i.e. an $\mathrm{ECRPQ}^\neg$ formula over Σ that is of the form $x = y$, $\pi = \pi'$, (x, π, y) or $R(\pi_1, \dots, \pi_n)$, where R is a regular relation over $(\Sigma_\perp)^n$ ($n \geq 0$) that represents an n-ary regular relation over Σ.

- Assume first that φ is $x = y$. Then $\bar{v}$ is of the form $(v, v') \in V \times V$. If $v = v'$, we set $\mathcal{A}_\varphi^{(G,\bar{v})}$ to be the automaton that accepts all strings over alphabet $V \cup \Sigma$. If $v \neq v'$ we set $\mathcal{A}_\varphi^{(G,\bar{v})}$ to be the automaton that accepts no string over the alphabet $V \cup \Sigma$.
- Assume second that φ is (x, π, y). Then $\bar{v}$ is of the form $(v, v') \in V \times V$. Let $V = \{v_1, \dots, v_m\}$, and let $U = \{u_1, \dots, u_m\}$ be a disjoint copy of V. The automaton $\mathcal{A}_\varphi^{(G,\bar{v})}$ is defined as follows: (1) $V \cup U \cup \{f\}$ is the set of states, where f is a node neither in U nor in V; (2) for every edge $(v_i, a, v_j) \in E$, the automaton has a transition labeled a from state v_i to state u_i and a transition labeled v_j from u_i to v_j; (3) the automaton has a transition from f to v labeled v; (4) the initial state is f; and (5) the final state is v'.
- Assume third that φ is $R(\pi_1, \dots, \pi_n)$, where R is a regular relation over $(\Sigma_\perp)^n$ ($n \geq 0$) that represents an n-ary regular relation over Σ. Then $\bar{v}$ is the empty tuple.
  Let $\mathcal{A}$ be the automaton that recognizes R. Then take $\mathcal{A}'$ to be the NFA $\mathcal{A} \times \bigcup_{(\bar{u},\bar{v}) \in G^n \times G^n} G^n(\bar{u}, \bar{v})$ over alphabet $(\Sigma_\perp)^n$. The states of $\mathcal{A}'$ are of the form $(q, \bar{u})$, where q is a state of $\mathcal{A}$ and $\bar{u}$ is a node in G^n. It is not hard to construct (in particular, using techniques that are similar to the ones in the previous item) from $\mathcal{A}'$ an automaton $\mathcal{A}''$ over alphabet $V^n \cup (\Sigma_\perp)^n$ that accepts exactly those strings $\bar{u}_0 \bar{a}_1 \bar{u}_1 \cdots \bar{a}_m \bar{u}_m$ such that for some sequence $q_0 \cdots q_m$ of states in $\mathcal{A}$, it is the case that $(q_0, \bar{u}_0) \cdots (q_m, \bar{u}_m)$ is an accepting run of $\mathcal{A}'$ over $\bar{a}_1 \cdots \bar{a}_m$.
  In this case we set $\mathcal{A}_\varphi^{(G,\bar{t})}$ to be $\mathcal{A}''$.

- Assume finally that φ is $\pi_1 = \pi_2$. Then $\bar{v}$ is the empty tuple.
  We construct $\mathcal{A}_\varphi^{(G,\bar{v})}$ as follows. Take $\bigcup_{(\bar{u},\bar{u}')\in G^2\times G^2} G^2(\bar{u},\bar{u}')$ and construct from it (using techniques similar to the ones in the previous items) an automaton $\mathcal{A}$ over alphabet $V^2 \cup (\Sigma_\perp)^2$ that accepts exactly those strings $\bar{v}_0\bar{a}_1\bar{v}_1\cdots\bar{a}_m\bar{v}_m$ such that $\bar{v}_0\cdots\bar{v}_m$ is an accepting run of $\bigcup_{(\bar{u},\bar{u}')\in G^2\times G^2} G^2(\bar{u},\bar{u}')$, i.e. $\bar{v}_0\bar{a}_1\bar{v}_1\cdots\bar{a}_m\bar{v}_m$ is a path in G^2. Finally, let $\mathcal{A}''$ be the restriction of $\mathcal{A}'$ to the alphabet $\{(s,s) \mid s \in V \cup \Sigma_\perp\}$.
  In this case we set $\mathcal{A}_\varphi^{(G,\bar{v})}$ to be $\mathcal{A}''$.

It is straightforward to see that the following is the case: Let $\varphi(\bar{x},\bar{\pi})$ be an atomic $\text{ECRPQ}^\neg$ formula, $G = (V,E)$ a graph database, and $\bar{v}$ be tuple of nodes in G such that $|\bar{v}| = |\bar{x}|$. The automaton $\mathcal{A}_\varphi^{(G,\bar{v})}$, as defined above, can be effectively constructed from φ, G and $\bar{v}$. Further, $\mathcal{A}_\varphi^{(G,\bar{v})}$ accepts precisely the set $\varphi(G,\bar{v})$.

Now we show the construction of $\mathcal{A}_\varphi^{(G,\bar{v})}$ for the rest of the cases. In all these cases, the construction is given recursively:

- If φ is $\neg\psi(\bar{x},\bar{\pi})$ then $\mathcal{A}_\varphi^{(G,\bar{v})}$ is defined as the automaton that accepts the complement of $\mathcal{A}_\psi^{(G,\bar{v})}$.
- If φ is $\psi(\bar{x},\bar{\pi}) \wedge \theta(\bar{x},\bar{\pi})$ then $\mathcal{A}_\varphi^{(G,\bar{v})}$ is the intersection of $\mathcal{A}_\psi^{(G,\bar{v})}$ and $\mathcal{A}_\theta^{(G,\bar{v})}$.
- If φ is $\exists y\psi(\bar{x},y,\bar{\pi})$ then $\mathcal{A}_\varphi^{(G,\bar{v})}$ is defined as $\bigcup_{v'\in V} \mathcal{A}_\psi^{(G,\bar{v},v')}$.
- If φ is $\exists\pi'\psi(\bar{x},\bar{\pi},\pi')$ then $\mathcal{A}_\varphi^{(G,\bar{v})}$ is defined as the projection of $\mathcal{A}_\psi^{(G,\bar{v})}$ over the component that represents π.

From here on the proof of the claim is almost straightforward, using the atomic case and induction. □

Data complexity. We now turn to data complexity, with the graph database as the sole input. That is, we look at problems $\text{CRPQ}^\neg$-EVAL(φ) and $\text{ECRPQ}^\neg$-EVAL(φ), for each query φ. Again we see a significant gap between allowing regular languages and relations. In the former case, the complexity matches that of the CRPQs, while in the latter case the problem is non-elementary. That is, neither the combined nor the data complexity of the problem of evaluating queries in $\text{ECRPQ}^\neg$ can be bounded by a stack of exponentials.

THEOREM 8.2.

- *For each* $\text{CRPQ}^\neg$ *formula* φ*, the problem* $\text{CRPQ}^\neg$-EVAL*(*φ*) is in* NLOGSPACE.
- *There is a finite alphabet* Σ *and a family* $\{\varphi_k\}_{k\in\mathbb{N}}$ *of Boolean ECRPQ*$^\neg$ *formulas over* Σ *such that, for all* $k \in \mathbb{N}$*, checking whether a* Σ*-labeled graph database* G *satisfies* φ_k *necessarily requires* k*-fold exponential time.*

PROOF. It is not hard to see that the PSPACE algorithm used in the first part of the proof of Theorem 8.1 (to show an upper bound on the complexity of $\text{CRPQ}^\neg$-EVAL) can be performed in NLOGSPACE if we assume the formula to be fixed. This shows that $\text{CRPQ}^\neg$-EVAL(φ) is in NLOGSPACE, for each $\text{CRPQ}^\neg$ formula φ.

Now we prove the second part. We only need to prove the following lemma.

LEMMA 8.3. *Fix the* FO *vocabulary* $\sigma = \langle <, U\rangle$*, where* $<$ *is binary and* U *is unary. Let* $\mathcal{C}_k$ *stand for the class of all* FO *formulas over* σ *in prenex-normal form with quantifier type* $(\forall^+\exists^+)^k$*. Then there exists a fixed ECRPQ*$^\neg$ *formula* φ_k *such that satisfiability of* $\mathcal{C}_k$ *formulas over* σ*-structures is exponential-time reducible to checking whether an input graph database* G *over* Σ *satisfies* φ_k*.*

In fact, since there exists a small constant $\varepsilon \in \mathbb{N}$ such that satisfiability of $\mathcal{C}_k$ formulas over σ-structures is not solvable in $(k-\varepsilon)$-EXPTIME [Robertson 1974; Stockmeyer 1974], this lemma immediately gives the second part of Theorem 8.2.

We proceed with the proof of the above lemma. Each candidate σ-structure $M = \langle\{1,\ldots,n\};<,U\rangle$ for $\mathcal{C}_k$ formulas can be represented as a string $\bar{w} = w_1 \ldots w_n$ over alphabet $\{0,1\}$, where $w_i = 1$ iff $i \in U$. However, since we are only allowed to construct a *fixed* ECRPQ$^\neg$ formula φ_k independent of the input $\mathcal{C}_k$ formulas, we can only embed the information on quantifier type in the input formula in φ_k; we will have to embed the remaining information on the input $\mathcal{C}_k$ formula within the graph database G. To this end, since the quantifiers in $\mathcal{C}_k$ formulas quantify over string positions, we will "annotate" binary strings with this piece of information, which will then be checked by a fixed regular relation in the formula in φ_k with the help of the non-fixed graph database G. Annotated strings will be represented over the alphabet $\{0,1,\square,\boxtimes\}$.

We illustrate our annotation schemes by an example. Suppose that the input is a $\mathcal{C}_1$ formula $\forall x_1 \forall x_2 \exists y_1 \exists y_2 \psi$, where ψ is quantifier-free, and the input string is $\bar{w} = 01000101$. We think of the input formula as a game with three rounds: (1) player $\forall$ annotates two (not necessarily distinct) positions of $\bar{w}$ with x_1 and x_2, (2) player $\exists$ annotates two (not necessarily distinct) positions of $\bar{w}$ with y_1 and y_2, and (3) check that the annotated string satisfies ψ. At the beginning of round 1, the annotated string is

$$0\square\square\square\square 1\square\square\square\square 0\square\square\square\square 0\square\square\square\square 0\square\square\square\square 1\square\square\square\square 0\square\square\square\square 1\square\square\square\square.$$

Notice that the number of $\square$ next to a bit 0 or 1 is 4, which corresponds to the number of variables in the input formula. Suppose that player $\forall$ assigns x_1 (resp. x_2) to position 1 (resp. 4). Then we cross the first box next to the bit at position 1 and the second box next to the bit at position 4, resulting in the following annotated string:

$$0\boxtimes\square\square\square 1\square\square\square\square 0\square\square\square\square 0\square\boxtimes\square\square 0\square\square\square\square 1\square\square\square\square 0\square\square\square\square 1\square\square\square\square.$$

Suppose that player $\exists$ assigns y_1 (resp. y_2) to position 6 (resp. 1). Then we cross the box again appropriately, resulting in the following annotated string:

$$0\boxtimes\square\square\boxtimes 1\square\square\square\square 0\square\square\square\square 0\square\boxtimes\square\square 0\square\square\square\square 1\square\square\boxtimes\square 0\square\square\square\square 1\square\square\square\square.$$

From the above example, we see that the number of rounds of the game depends only on the number of quantifier alternations (i.e. the value of k in $\mathcal{C}_k$), *not* on the number of variables in the input $\mathcal{C}_k$ formula. In the general case of a formula in $\mathcal{C}_k$ of the form

$$\forall \mathbf{x}_1 \exists \mathbf{y}_1 \ldots \forall \mathbf{x}_k \exists \mathbf{y}_k \psi,$$

where ψ is quantifier-free, the $\mathcal{C}_k$-game consists of $2k+1$ rounds, with rounds $2i-1$ and $2i$ ($1 \leq i \leq k$) corresponding to the annotation of i-th block $\forall \mathbf{x}_i \exists \mathbf{y}_i$ of quantifiers in the formula, and round $2k+1$ for checking whether the annotation satisfies ψ.

Therefore, we proceed with the proof of Lemma 8.3 as follows. The alphabet Σ of edge labels is defined as:

$$\Sigma := \{0,1,\square,\boxtimes,\ell\},$$

where ℓ is a fresh distinguished symbol. The formula φ_k that we construct uses path variables $\{\pi_i\}_{i=1}^{2k+1}$ such that π_1 captures the annotated string at the beginning of round 1, and π_i ($2 \leq i \leq 2k+1$) captures the annotated string at the end of round $i-1$. The way in which π_i is quantified in the formula φ_k will depend on the parity of i. More precisely, we define φ_k as follows:

$$\begin{aligned}
\theta_{k+1}(y,\pi) &:= (y=y) \wedge (\pi=\pi)\\
\theta_i(y,\pi_{2i-1}) &:= \exists x \Big(\ell(y,x) \wedge \forall z \forall \pi_{2i} \big((x,\pi_{2i},z) \wedge R(\pi_{2i-1},\pi_{2i}) \rightarrow\\
&\qquad \exists u \exists y \exists \pi_{2i+1} \big(\ell(z,u) \wedge (u,\pi_{2i+1},y) \wedge R(\pi_{2i},\pi_{2i+1}) \wedge \theta_{i+1}(y,\pi_{2i+1}) \big)\big)\Big) \qquad (1 \leq i \leq k)\\
\varphi_k &:= \exists x \exists y \exists \pi_1 \big((x,\pi_1,y) \wedge R'(\pi_1) \wedge \theta_1(y,\pi_1) \big),
\end{aligned}$$

where $R(\pi,\pi')$ is a regular relation checking "consistencies" between π and π' in the following sense (we denote by $\pi[i]$ the i-th letter of $\lambda(\pi)$):

(1) $|\pi| = |\pi'|$,
(2) for each $i \in \mathbb{N}$, if $\pi[i] = 0$, then $\pi'[i] = 0$
(3) for each $i \in \mathbb{N}$, if $\pi[i] = 1$, then $\pi'[i] = 1$,
(4) for each $i \in \mathbb{N}$ if $\pi[i] = \Box$, then $\pi'[i] \in \{\Box, \boxtimes\}$, and
(5) for each $i \in \mathbb{N}$, if $\pi[i] = \boxtimes$, then $\pi'[i] = \boxtimes$,

and $R'(\pi)$ is the regular language $\big((0+1) \cdot \Box^*\big)^*$ that checks that the label of π represents a potential initial annotation of a string.

For example, whenever $k=1$, we obtain the formula φ_1 defined as:

$$\begin{aligned}
&\exists x \exists y \exists \pi_1 \Big((x,\pi_1,y) \wedge R'(\pi_1) \wedge\\
&\exists x \big(\ell(y,x) \wedge \forall z \forall \pi_2 \big((x,\pi_2,z) \wedge R(\pi_1,\pi_2) \rightarrow \exists u \exists y \exists \pi_3 (\ell(z,u) \wedge (u,\pi_3,y) \wedge R(\pi_2,\pi_3)) \big) \big) \Big).
\end{aligned}$$

At this point, observe that φ_k is independent of the input $\mathcal{C}_k$ formula. We now need to take into account the input formula in the graph database G that we construct. Suppose that the input formula $\Psi \in \mathcal{C}_k$ is

$$\forall \mathbf{x}_1 \exists \mathbf{y}_1 \ldots \forall \mathbf{x}_k \exists \mathbf{y}_k \psi$$

where ψ is quantifier-free. Let $r_i := |\mathbf{x}_i|$ and $t_i := |\mathbf{y}_i|$. Let $r := \sum_{i=1}^{k} r_i$ and $t := \sum_{i=1}^{k} t_i$.

We construct the graph database G from the disjoint $2k+1$ graph databases $G_1, \ldots, G_{2k+1}$ defined below. Each graph database G_i has a predetermined source node c_i and a predetermined sink node c_i' such that the set of paths from c_i and c_i' will be used to restrict the range of the paths where the path variable π_i in the formula φ_k can be witnessed. For convenience, when defining G_i we allow the use of regular expressions over Σ, as there is a standard polynomial-time translation from regular expressions to NFAs.

The graph databases $G_1, \ldots, G_{2k+1}$ are defined as follows:

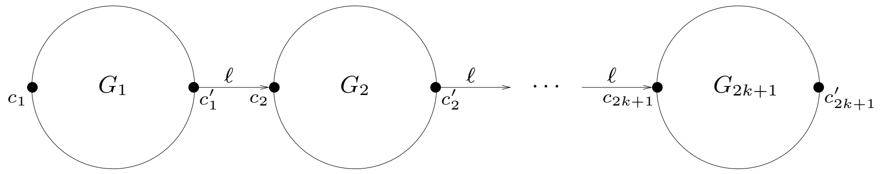
(1) The graph G_1 is defined by the regular expression $\big((0+1)\Box^{r+t}\big)^*$. That is, the set of the labels of paths that go from c_1 into c_1' in G_1 is precisely the set of strings that satisfy $\big((0+1)\Box^{r+t}\big)^*$, or, in other words, that represent annotations of strings in the first round of the $\mathcal{C}_k$-game for Ψ.
(2) The graph G_{2i} $(1 \leq i \leq k)$ is defined by the regular expression that defines all those strings in $\big((0+1) \cdot (\Box + \boxtimes)^{r+t}\big)^*$ that, in addition, satisfy the following: For each $1 \leq j \leq \sum_{h=1}^{i} r_h + \sum_{h=1}^{i-1} t_h$, *exactly one* box located j positions to the right of an occurrence of 0 or 1 is $\boxtimes$, and for each $\sum_{h=1}^{i} r_h + \sum_{h=1}^{i-1} t_h < j \leq r+t$, *each* box located j positions from a 0 or a 1 is $\Box$.

That is, G_{2i} $(1 \leq i \leq k)$, seen as an NFA with initial state c_{2i} and final state c'_{2i}, defines exactly those strings that represent valid annotations of strings in the $\mathcal{C}_k$-game for Ψ precisely after stage $(2i-1)$ of the game has been played. Indeed, each variable that appears in the first $(i-1)$ blocks $\forall \mathbf{x}_1 \exists \mathbf{y}_1 \cdots \forall \mathbf{x}_{i-1} \exists \mathbf{y}_{i-1}$ is assigned to a unique position in the annotated string (represented by the fact that for each $1 \leq j \leq \sum_{h=1}^{i-1} r_h + \sum_{h=1}^{i-1} t_h$, exactly one box located j positions to the right of an occurrence of 0 or 1 is $\boxtimes$), each quantifier in $\mathbf{x}_i$ is assigned to a unique position in the annotated string (represented by the fact that for each $\sum_{h=1}^{i-1} r_h + \sum_{h=1}^{i-1} t_h < j \leq \sum_{h=1}^{i} r_h + \sum_{h=1}^{i-1} t_h$, exactly one box located j positions right of an occurrence of 0 or 1 is $\boxtimes$), and no other variable is assigned to a position in the string (represented by the fact that for each position located j positions to the right of a 0 or a 1 is $\Box$, where $\sum_{h=1}^{i} r_h + \sum_{h=1}^{i-1} t_h < j \leq r + t$).

(3) The graph G_{2i+1} $(1 \leq i < k)$ is defined by the regular expression that defines all those strings in $\big((0+1) \cdot (\Box + \boxtimes)^{r+t}\big)^*$ that, in addition, satisfy the following: For each $1 \leq j \leq \sum_{h=1}^{i} r_h + \sum_{h=1}^{i} t_h$, *exactly one* box located j positions to the right of an occurrence of 0 or 1 is $\boxtimes$, and for each $\sum_{h=1}^{i} r_h + \sum_{h=1}^{i} t_h < j \leq r + t$, *each* box located j positions from a 0 or a 1 is $\Box$.
That is, G_{2i+1} $(1 \leq i < k)$, seen as an NFA with initial state c_{2i+1} and final state c'_{2i+1}, defines exactly those strings that represent valid annotations of strings in the $\mathcal{C}_k$-game for Ψ precisely after round $2i$ of the game has been played. Indeed, each variable that appears in the first i blocks $\forall \mathbf{x}_1 \exists \mathbf{y}_1 \cdots \forall \mathbf{x}_i \exists \mathbf{y}_i$ is assigned to a unique position in the annotated string (represented by the fact that for each $1 \leq j \leq \sum_{h=1}^{i} r_h + \sum_{h=1}^{i} t_h$, exactly one box located j positions right of an occurrence of 0 or 1 is $\boxtimes$), and no other variable is assigned to a position in the string (represented by the fact that for each position located j positions from a 0 or a 1 is $\Box$, where $\sum_{h=1}^{i} r_h + \sum_{h=1}^{i} t_h < j \leq r + t$).

(4) Finally, we define the graph G_{2k+1} to be the intersection of the language defined by $\big((0+1) \cdot (\Box + \boxtimes)^{r+t}\big)^*$ and the regular language of annotated strings satisfying ψ, for which there is a standard exponential-time construction of NFA from FO logic (recall that ψ is quantifier-free). That is, G_{2k+1}, when seen as an NFA with initial state c_{2k+1} and final state c'_{2k+1}, accepts precisely the annotations of strings that satisfy ψ. As we shall see later, with the help of φ_k we can enforce that these correspond to valid annotations of strings in the $\mathcal{C}_k$-game for Ψ precisely after round $2k+1$ of the game has been played.

The graph database G is constructed from $G_1, \ldots, G_{2k+1}$ by adding an ℓ-labeled edge from c'_i into c_{i+1}, for each $1 \leq i \leq 2k$, as shown in the following figure:



Clearly, each G_i can be constructed in exponential time from the $\mathcal{C}_k$ formula ψ, and hence G can also be constructed in exponential time.

To conclude, it is not hard to prove that there is a binary string $\bar{w}$ that satisfies ψ if and only if $G \models \varphi_k$, where G is as constructed above from Ψ. Indeed, assume first that $G \models \varphi_k$. Then $G \models \exists x \exists y \exists \pi_1 ((x, \pi_1, y) \wedge R'(\pi_1) \wedge \theta_1(y, \pi_1))$, and $\theta_1(y, \pi_1)$ is of the form $\exists z (\ell(y, z) \wedge \cdots)$. Since π_1 has to satisfy R, this implies that π_1 can only be witnessed in G by a path ρ_1 inside G_1. Further, since there is an outgoing edge labeled ℓ from the endpoint of ρ_1, this path can only go from c_1 into c_1' (and hence x and y can only be witnessed in G by nodes c_1 and c_1', respectively). Thus, the label of ρ_1 represents the annotation of a string $\bar{w}$ at the beginning of round 1 of the game for Ψ (that is, the "annotated" string belongs to the language defined by $\big((0+1)\Box^{r+t})^*$). We claim that $\bar{w}$ satisfies Ψ, and hence that Ψ is satisfiable inside the class of σ-structures. We explain below why this is the case.

One can prove the claim by a simple but rather cumbersome inductive argument. We omit this argument for the sake of readability and rather provide its underlying intuition. Since $G \models \theta_1(c_1', \rho_1)$, it means that

$$G \models \exists x \Bigg(\ell(c_1', x) \wedge \forall z \forall \pi_2 \big((x, \pi_2, z) \wedge R(\rho_1, \pi_2) \rightarrow \\ \exists u \exists y \exists \pi_3 (\ell(z, u) \wedge (u, \pi_3, y) \wedge R(\pi_2, \pi_3) \wedge \theta_2(y, \pi_3)) \big) \Bigg).$$

Thus, variable x in the previous formula can only be witnessed in c_2. Then for every node z and every path ρ_2 that goes from c_2 to z in G and is in the relationship R to ρ_1 the following holds:

— z has an outgoing edge labeled ℓ to a node u;
— there is a node y such that there is a path ρ_3 from u to y that is in the relationship R to ρ_2; and
— $G \models \theta_2(y, \rho_3)$, in particular, $G \models \exists x \ell(y, x)$.

Notice that there is a unique node z in G such that $G \models \exists \pi_2 (c_2, \pi_2, z) \wedge R(\rho_1, \pi_2)$. This node is precisely c_2' and π_2 is any path ρ_2 from c_2 into c_2' that encodes precisely the same binary string $\bar{w}$ as ρ_1. The reason is that since $R(\rho_1, \rho_2)$ holds, ρ_2 must encode the same binary string as ρ_1. Further, ρ_2 does not mention the letter ℓ, and hence it can only be witnessed in G inside G_2. Moreover, ρ_2 has the same length as ρ_1, and, thus, its last node can only be c_2'. Since ρ_2 goes from c_2 into c_2' it encodes a proper annotation of the variables $\mathbf{x}_1$ over $\bar{w}$.

Hence for every path ρ_2 that encodes a proper annotation of $\mathbf{x}_1$ over $\bar{w}$, there is path ρ_3 that starts in c_3 and that is in the relationship R to π_2. Thus, for the same reasons given in the previous paragraph, ρ_3 goes from c_3 into c_3' inside G_3, and it encodes the same binary annotation of $\mathbf{x}_1$ over $\bar{w}$ than ρ_2, but this time extended with a proper annotation of $\mathbf{y}_1$.

Summing up, we have that for every proper annotation $\mathbf{x}_1$ over $\bar{w}$ there exists a proper annotation of $\mathbf{y}_1$ over $\bar{w}$, encoded by path ρ_3, such that $G \models \theta_2(c_3', \rho_3)$. By continuing this process iteratively, we can conclude that for every proper annotation of $\mathbf{x}_1$ over $\bar{w}$ there exists a proper annotation of $\mathbf{y}_1$ over $\bar{w}$, such that ... for every annotation of $\mathbf{x}_k$ over $\bar{w}$ there exists a proper annotation of $\mathbf{y}_k$ over $\bar{w}$, and that this final annotation of a string is encoded in the path ρ_{2k+1} that witnesses the variable π_{2k+1} in G. But by following essentially the same arguments used above, one can show that π_{2k+1} can only be witnessed in G by a path ρ_{2k+1} that goes from c_{2k+1} into c_{2k+1}' in G_{2k+1}. This path must encode a valid annotation of $\bar{w}$ that satisfies Ψ. Therefore, we conclude that $\bar{w}$ satisfies Ψ.

The opposite direction, that is, that $G \models \varphi_k$ follows from the fact that φ is satisfiable in the class of σ-structures, can be proved with a similar argument. □

We know of course that the containment problem is undecidable for ECRPQs, and thus for ECRPQ$^\neg$ queries. We remark that even a simpler satisfiability problem, asking whether for a Boolean ECRPQ$^\neg$ query φ there is a graph database G such that

$\varphi(G) = \texttt{true}$, is undecidable. This is because the finite satisfiability problem for arbitrary FO formulas over binary predicates can easily (and effectively) be encoded into the satisfiability problem for $\mathrm{ECRPQ}^{\neg}$ formulas.

8.2. Adding non-regular relations

A well-known class viewed as a natural extension of regular relations is the class of *rational relations*. Binary rational relations can be viewed as sets of pairs (s, s') of strings over Σ such that s' is a possible output of a nondeterministic *transducer* on s. However, a standard PCP reduction shows the following:

PROPOSITION 8.4. *If rational relations are allowed in place of regular relations in* ECRPQ*s, then the query evaluation problem becomes undecidable.*

PROOF. We reduce from the following version of the *correspondence problem*, which is known as PCP. Given two equally long ordered lists $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of strings over alphabet Σ, decide whether there exists a sequence of indices $i_1, i_2, \ldots, i_k$ such that $1 \leq i_j \leq n$ $(1 \leq j \leq k)$ and $a_{i_1} a_{i_2} \cdots a_{i_k} = b_{i_1} b_{i_2} \cdots b_{i_k}$.

Assume without loss of generality that Σ is disjoint from $\mathbb{N}$. Corresponding to every input $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of PCP over alphabet Σ, we define the following:

— An alphabet $\Sigma(n) := \Sigma \cup \{1, 2, \ldots, n\}$;
— a regular language $R_{a,n} := (\bigcup_{1 \leq i \leq n} a_i \cdot i)^*$;
— a regular language $R_{b,n} := (\bigcup_{1 \leq j \leq n} b_j \cdot j)^*$;
— for each $\Sigma' \subseteq \Sigma(n)$, a binary rational relation $R_{\Sigma'}$ that contains all pairs (w_1, w_2) of strings over $\Sigma(n)$ such that w_1 is precisely the restriction to the alphabet Σ' of the string w_2.

We now define a Boolean ECRPQ Q, that makes use of rational relations over alphabet $\Sigma(n)$, as follows:

$$Ans() \leftarrow \bigwedge_{1 \leq i \leq 4} (x_i, \pi_i, y_i), R_{a,n}(\pi_1), R_{b,n}(\pi_2), R_{\{1,2,\ldots,n\}}(\pi_3, \pi_1),$$
$$R_{\{1,2,\ldots,n\}}(\pi_4, \pi_2), \pi_4 = \pi_3, R_{\Sigma}(\pi_5, \pi_1), R_{\Sigma}(\pi_6, \pi_2), \pi_5 = \pi_6,$$

and a graph G, that is exactly the graph $G_{\mathcal{R}}^{\Sigma(n)}$ that is used in the proof of the hardness part of Theorem 6.3. That is, for each node $v \in G_{\mathcal{R}}^{\Sigma(n)}$ and string $w \in \Sigma(n)$, there is a path $\rho \in G_{\mathcal{R}}^{\Sigma(n)}$ that starts at v and such that $\lambda(\rho) = w$.

It is now straightforward to prove that $G \models Q$ iff there is a solution for the PCP problem for the given input. This shows that the problem of query evaluation for ECRPQs that make use of rational relations is undecidable. □

Hence, we need to work with weaker extensions. We now look at two such examples.

Linear constraints on the number of occurrences of events. It is common to have preferences for one event over another when querying graph databases. For example, when planning an itinerary from London to Sydney, a Singaporean might want to find a sequence of flights with Singapore Airlines for at least 80% of the entire journey duration. In this case, only the ratio on the *number of occurrences* of events matter, not the order in which they occur. We will now extend ECRPQs with linear constraints on the number of occurrences of events. Suppose that we deal with graph databases over the alphabet $\Sigma = \{a_1, \ldots, a_k\}$. Then, ECRPQs *with linear constraints* *on the number of occurrences of events* are of the form:

$$Ans(\bar{z},\bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m} (x_i,\pi_i,y_i),\ \bigwedge_{1\leq j\leq t} R_j(\bar{\omega}_j),\ \mathbf{A}\bar{\ell} \geq \mathbf{b}\,, \tag{7}$$

where $Ans(\bar{z},\bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i,\pi_i,y_i), \bigwedge_{1\leq j\leq t} R_j(\bar{\omega}_j)$ is a ECRPQ as in (2), and

— $\mathbf{A}$ is a $h \times (k \cdot m)$ matrix over $\mathbb{Z}$, for some $h > 0$;
— $\mathbf{b}$ is a vector in $\mathbb{Z}^h$; and
— $\bar{\ell} = (\ell_{1,1},\cdots,\bar{\ell}_{1,k},\ldots,\ell_{m,1},\cdots,\ell_{m,k})$.

The semantics is extended as follows: each $\ell_{i,j}$ ($1 \leq i \leq m$ and $1 \leq j \leq k$) is interpreted as the number of occurrences of event (symbol) a_j in the path π_i. The last clause of the query is true if $\mathbf{A}\ell \geq \mathbf{b}$ under this interpretation. In an analogous way we define the class of CRPQs with linear constraints on the number of occurrences of events.

Before proving our main results on CRPQs and ECRPQs extended with linear constraints on the number of occurrences of events, let us first revisit the above example on finding an itinerary from London to Sydney with Singapore Airlines for 80% of the entire journey duration. One way to achieve this is to represent a graph database of flights in such a way that each a_j-labeled edge constitutes a small fixed fraction (e.g. 10 minutes) of a flight with the airline a_j. (If our graph database has cities as nodes and "full" flights as edges, we could introduce "intermediate" nodes and edges to indicate time information). In fact, supposing that a_1 indicates flights with Singapore Airlines, we could replace $a_2,\ldots,a_k$ with the event label a_2 to indicate "flights with airlines other than Singapore Airlines". The desired query is expressible as a CRPQ with linear constraints on the number of occurrences of events as follows:

$$Ans() \leftarrow (\text{London}, \pi_1, \text{Sydney}), (a_1 - 4a_2 \geq 0).$$

THEOREM 8.5.

— *The combined complexity of* ECRPQ*s with linear constraints on the number of occurrences of events is* PSPACE*-complete.*
— *The combined complexity of* CRPQ*s with linear constraints on the number of occurrences of events is* NP*-complete.*
— *For both classes of queries, data complexity is in* NLOGSPACE.

Thus, from the point of view of overall complexity, adding comparisons on the number of occurrences of events is free, as it does not increase the complexity of ECRPQs and CRPQs.

PROOF. We start by proving the second part of the theorem. To this end, we need the notion of Parikh images of string languages. Fix an ordering of the alphabet $\Sigma = \{a_1,\ldots,a_k\}$ of the language under consideration, say, $a_1 < \ldots < a_k$. Given a string w over Σ, the *Parikh image* $\text{par}(w)$ *of* w is a tuple $(|w|_{a_1},\ldots,|w|_{a_k}) \in \mathbb{N}^k$, where $|w|_a$ counts the number of occurrences of a in the string w. The *Parikh image of a language* L over Σ is simply the set $L_{\text{par}} := \{\text{par}(w) : w \in L\} \subseteq \mathbb{N}^k$. We will also use the linear time translation by Verma et al. [2005] from a given NFA $\mathcal{A}$ to an existential Presburger formula $\varphi_{\mathcal{A}}(x_1,\ldots,x_k)$ capturing the Parikh image of the language L of $\mathcal{A}$, i.e., for each tuple $\bar{n} \in \mathbb{N}^k$ it is the case that $\bar{n} \in L_{\text{par}}$ iff the formula $\varphi_{\mathcal{A}}(\bar{n})$ holds.

Let us assume that the input consists of a formula φ of the form

$$Ans(\bar{z},\bar{\chi}) \leftarrow \bigwedge_{i=1}^{m}(x_i,\pi_i,y_i),\ \bigwedge_{r=1}^{t} L_r(\omega_r),\ \mathbf{A}\bar{\ell} \geq \mathbf{b}\,,$$

a graph database G over the alphabet $\Sigma = \{a_1, \ldots, a_k\}$, and a mapping ν from node variables $\bar{z}$ to nodes in G, and path variables $\bar{\chi}$ to paths in G. An NP procedure M for checking whether $(\nu(\bar{z}), \nu(\bar{\chi})) \in \varphi(G)$ works as follows. First, M guesses an assignment σ to the node variables x_i and y_i, while making sure that σ agrees with ν. It then constructs in polynomial time an existential Presburger formula ψ with free variables $X = \{x_{i,j}\}_{1 \leq i \leq m, 1 \leq j \leq k}$ such that, for any mapping $\alpha : X \to \mathbb{N}$, it is the case that ψ is true under α iff $(\nu(\bar{z}), \nu(\bar{\chi})) \in \varphi(G)$ under the assignment (σ, μ), where μ is some assignment of path variables π_i's to paths ρ_i such that $|\rho_i|_{a_j} = \alpha(x_{i,j})$ for each $j = 1, \ldots, k$. Since checking satisfaction of existential Presburger formulas can itself be done by an NP procedure (e.g. see [Scarpellini 1984]), this will prove that the procedure M is an NP procedure. It remains to give the construction of ψ.

Without loss of generality, we assume that $m = t$, i.e., each π_i appears in the big conjunction $\bigwedge_{r=1}^{t} L_r(\omega_r)$. That way, we may rewrite $\bigwedge_{r=1}^{t} L_r(\omega_r)$ as $\bigwedge_{i=1}^{m} L_i(\pi_i)$. For each $i = 1, \ldots, m$, the procedure M considers whether π_i appears in $\bar{\chi}$. If it does, then it checks in polynomial time that $\sigma(x_i)$ reaches $\sigma(x_i')$ under the path $\nu(\pi_i)$, and that $\nu(\pi_i) \in L_i$. If the check is not successful, M rejects. If this check is successful, for each $j = 1, \ldots, k$, the variable $\ell_{i,j}$ in $\bar{\ell}$ is replaced by the number $|\nu(\pi_i)|_{a_j}$ of times a_j appears in $\nu(\pi_i)$. In the case when π_i does not appear in $\bar{\chi}$, the procedure M applies the linear-time translation of Verma et al. [2005] on the NFA $\mathcal{A}_i$ obtained via the standard product construction on the graph G construed as an NFA with initial state $\sigma(x_i)$ and final state $\sigma(x_i')$ and the NFA for L_i yielding an existential Presburger formula $\varphi_i(\ell_{i,1}, \ldots, \ell_{i,k})$ capturing the Parikh image of $\mathcal{A}_i$. The desired formula is

$$\exists \bar{\ell} \left(\bigwedge_{i=1}^{m} \varphi_i(\ell_{i,1}, \ldots, \ell_{i,k}) \wedge \mathbf{A}\bar{\ell} \geq \mathbf{b} \right).$$

Notice that this formula is of size polynomial in the size of the input. This completes the proof of the second part of the theorem.

In order to prove the first and third parts of the theorem, we will use the notion of reversal-bounded counter automata. A *k-counter automaton* over the alphabet Σ is a tuple $\mathcal{A} = (Q, X, \Sigma, \delta, q_0, q_F)$ where

— Q is a finite set of control states,
— $X = \{x_1, \ldots, x_k\}$ is a set of k counter variables,
— $q_0 \in Q$ is an initial state,
— $q_F \in Q$ is a final state, and
— δ is a finite subset of $(Q \times Cons_X) \times \Sigma_\epsilon \times (Q \times \{-1, 0, 1\}^k)$, where $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$ and $Cons_X$ is the set of "counter tests" of the form $\bigwedge_{i=1}^{k} x_i \sim_i 0$, where $\sim_i \in \{=, >\}$ for each $i = 1, \ldots, k$.

A configuration of $\mathcal{A}$ is a tuple of the form $(q, \mathbf{v}) \in Q \times \mathbb{N}^k$, which indicates the state $\mathbb{A}$ is in and the values of the k counters. A *run* σ of $\mathcal{A}$ is a sequence of configurations

$$(q_0, \mathbf{v}_0) \to_{a_1} \ldots \to_{a_n} (q_n, \mathbf{v}_n)$$

where

— each $q_i \in Q$, and q_0 is the initial state,
— $\mathbf{v}_0 = \mathbf{0}$, and
— for each $i = 0, \ldots, n-1$, there exists a transition

$$(q_i, \varphi(\bar{x}), a_i, q_{i+1}, \mathbf{u})$$

such that $\varphi(\mathbf{v}_i)$ holds and $\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{u}$.

The run σ is *accepting* if q_n is the final state q_F. In this case, we say that the string $a_1 \dots a_n$ is *accepted* by $\mathcal{A}$. The *language* $L(\mathcal{A})$ *of* $\mathcal{A}$ is the set of strings accepted by $\mathcal{A}$. For each $r \in \mathbb{N}$, the run σ is said to be *r-reversal-bounded* if, for each counter x_i, the number of reversals between non-incrementing and non-decrementing modes of the value of x_i in σ is at most r. For example, if the value of x_i in σ is $0, 1, 1, 1, 2, 3, 4, 4, \overline{4, 3}, 2, \overline{2, 3}$, then the number of reversals of x is 2 (reversals occur in between the overlined positions). An r-reversal k-counter automaton over Σ is a tuple $(\mathcal{A}, r)$, where $\mathcal{A}$ is k-counter automaton over Σ. The language of $(\mathcal{A}, r)$ is simply the set of strings accepted by $\mathcal{A}$ with a witnessing run that is r-reversal-bounded. When r is understood, we simply refer to $(\mathcal{A}, r)$ as $\mathcal{A}$.

We proceed to the proof of the first and third parts of the theorem. Suppose that the input consists of a query φ

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{i=1}^{m} (x_i, \pi_i, x_i'), \bigwedge_{1 \le j \le t} R_j(\bar{\omega}_j),\ \mathbf{A}\bar{\ell} \ge \mathbf{b}\,,$$

a graph database G over $\Sigma = \{a_1, \dots, a_k\}$, and a mapping ν from node variables $\bar{z}$ to nodes in G, and path variables $\bar{\chi}$ to paths in G. We proceed by proving a technical lemma stating a "small model property" of the query φ.

LEMMA 8.6. *If $(\nu(\bar{z}), \nu(\bar{\chi})) \in \varphi(G)$, then there exists a satisfying assignment of path variables in $\{\bar{\omega}_j\}_{j=1}^{t}$ of length at most polynomial in $|G| + |\nu|$ and exponential in $|\varphi|$.*

We first show how the first and third parts of the theorem follow from this technical lemma. It suffices to give a nondeterministic polynomial (respectively, logarithmic) space Turing machine M for combined (respectively, data) complexity. Initially, the Turing machine M guesses an assignment σ to the node variables, while making sure that σ is consistent with ν. Note that $|\sigma| = O(m \log(|G|)) = O(|\varphi| \log(|G|))$ since the nodes in the graphs can be named as numbers written in binary (and so requires only $\log(|G|)$ many bits). The machine M now proceeds to the stage of simultaneously guessing the paths $\{\pi_i\}_{i=1}^{m}$. To do so, it will "nondeterministically walk through" the graph G (from different starting nodes simultaneously) and the NFAs $\{R_j\}_{j=1}^{t}$. The machine M has a variable v_i stored on its working tape to remember the "current" node in the path π_i, initially setting v_i to be $\sigma(x_i)$. Likewise, M has a variable u_j to remember the current state of the NFA R_j, initially setting u_j to be the initial state of R_j. Additionally, we need to keep track of the following information:

— For each path variable π and each symbol $a \in \Sigma$, how many a's have been seen so far in π.
— The number of input letters that have been read so far.

Walking through this graph and the NFAs is simple: at each iteration stage, we nondeterministically guess a letter in $\Sigma_{\perp}^{m}$ while making sure that they are consistent with the path assignments in ν, which can be easily done since M keeps track of the number of input letters that have been read so far. After this, we guess the next move for each NFA and the graph G while incrementing the counters appropriately. The simulation stops if any one of the following conditions is satisfied:

(1) The number of input letters that have been guessed reaches the maximum limit given by the above lemma.
(2) Each variable v_i is set to $\sigma(x_i')$, each variable u_j is set to the final state of the NFA R_j, and the linear constraints evaluated against the variables as recorded in the counters are satisfied.

In the first case, the input formula is false. In the second case, the input formula is true. Since numbers can be represented in binary in the work tape of the Turing machine, this simulation takes space polynomial in $|\varphi|$, and logarithmic in $|G| + |\nu|$. This completes the proof of the first and third parts of the theorem.

We now proceed to the proof of the above technical lemma. Suppose that $(\nu(\bar{z}), \nu(\bar{\chi})) \in \varphi(G)$, where the ground node variables in φ are satisfied by the node valuation σ. Suppose φ' is obtained by dropping the linear constraints $\mathbf{A}\bar{\ell} \geq \mathbf{b}$ from φ. We have shown in the proof of Theorem 6.3 that there exists an NFA $\mathcal{A} = (Q, (\Sigma_\perp)^m, \Delta, q_0, q_F)$ capturing $\varphi'(G)$ whose number of states is polynomial in $|G| + |\nu|$ and exponential in $|\varphi'|$. Since the query φ has counting constraints, we modify the construction of the automaton $\mathcal{A}$ by introducing an unbounded counter $x_{i,j}$ (for each $i = 1, \ldots, m$ and $j = 1, \ldots, k$) for counting the numbers of occurrences of the letter a_j in the path ρ_i as the NFA $\mathcal{A}'$ traverses the input strings $\bar{\rho} = \rho_1 \otimes \ldots \otimes \rho_m$ over the alphabet $(\Sigma_\perp)^m$ instantiating the path variables $\bar{\varphi} = (\pi_1, \ldots, \pi_m)$. Note that, in doing so, we do not introduce extra states in $\mathcal{A}$; we merely add appropriate counter increments in the original transition relation Δ in $\mathcal{A}$. If we use non-succinct counter automaton representation, the modified transition relation will be of size at most $|\Delta| \times 2^{m+k} \leq |\mathcal{A}| \times 2^{m+k}$. After the entire input string has been fully read and q_F is reached, we need to additionally ensure that $\mathbf{A}\bar{\ell} \geq \mathbf{b}$ is satisfied. To this end, let us assume that q_F is a "dead end", i.e., has no outgoing transition in Δ. Bringing each summand with negative coefficient to the left hand side of the inequality ensures that each summand in the system of inequalities has a positive coefficient. Let us now add to $\mathcal{A}$ an extra counter for each side of each inequality, i.e., a total of $2h$ extra counters $y_1, \ldots, y_h$ and $y'_1, \ldots, y'_h$ for, respectively, left and right sides of the inequalities. From q_F, the automaton $\mathcal{A}$ will start decrementing each of the $x_{i,j}$ until all of them are zero. Additionally, while decrementing the counter $x_{i,j}$ by one, for each $r = 1, \ldots, h$ the automaton $\mathcal{A}$ adds the absolute value of $\mathbf{A}[r, i \times j]$ (i.e. the coefficient of the variable $\ell_{i,j}$ in the rth inequality counting the number of occurrences of event a_i in ρ_j) to the new counter y_r or y'_r depending whether or not $\mathbf{A}[r, i \times j]$ is positive. Likewise, we need to add the absolute value of b_r to either y_r or y'_r depending whether or not b_r is positive. After all of the $x_{i,j}$ counters have been decremented to zero, we make sure that $y_r \geq y'_r$ holds for each $r = 1, \ldots, h$ by stepping through each pair y_r and y'_r of variables and decrementing y_r and y'_r at the same time making sure at the end we have $y'_r = 0 \wedge y_r = 0$ or $y'_r = 0 \wedge y_r > 0$. Notice that the resulting non-succinct counter automaton $\mathcal{A}$ is 1-reversal-bounded (i.e. all runs of $\mathcal{A}$ are 1-reversal-bounded) and has $2h + mk$ counters. Furthermore, since we assume a binary representation of numbers in the matrix $\mathbf{A}$ and vector $\mathbf{b}$, the number of states in $\mathcal{A}$ is polynomial in $|G| + |\nu|$ and exponential in the size $|\varphi|$ of the input query φ.

We now make use of the following recent result on the Parikh images of non-succinct reversal-bounded counter automata from [To 2010]:

PROPOSITION 8.7. *Given an r-reversal-bounded k-counter automaton $\mathcal{B}$ with n states over an alphabet of size m, the Parikh image of its language can be represented as a disjunction of existential Presburger formulas each of size polynomial in $r + k + m + \log(n)$ with at most $O(rk + m)$ variables.*

In this proposition, the (in)equalities inside the Presburger formulas allow arbitrarily many summands on both sides of the (in)equalities, where each summand allows multiplication by an integer constant represented in binary. This proposition is a paraphrase of [To 2010, Proposition 7.5.2], where integer constants are represented in unary. For the sake of completeness, we provide a proof in the appendix. Now, applying this proposition on the counter automaton $\mathcal{A}$ that we just constructed, it follows that the Parikh image of the language L of $\mathcal{A}$ can be represented as a disjunction of existential Presburger formulas each of size polynomial in $2h + mk + |\varphi| \log(|G| + |\nu|)$

with at most $O(2h+mk)$ variables. Scarpellini [Scarpellini 1984, Theorem 6 (a)] proved that a satisfiable existential Presburger formula θ with c variables has solutions where each variable is assigned a number that is exponential in $|\theta|$ and in c. Using this result, it follows that if the Parikh image of L is nonempty, then it contains a tuple $\bar{n} = (\bar{n}_1, \ldots, \bar{n}_m)$ with each $\bar{n}_i = (n_{i,1}, \ldots, n_{i,k})$ and each $n_{i,j}$ is polynomial in $|G| + |\nu|$ and exponential in $|\varphi|$. Note that $\sum_{j=1}^{k} n_{i,j}$ equals the length of the path assignment in Σ^* for the path variable π_i witnessing the tuple $\bar{n}$ in the Parikh image of L. Therefore, the above lemma immediately follows. □

Theorem 8.5 shows that comparisons on the number of occurrences of events can be added "for free" to ECRPQs, at least in terms of the complexity of query evaluation. A natural question at this point is whether this is also true for the class $\mathrm{ECRPQ}^{\neg}$. That is, can linear constraints on the number of occurrences of events can be added to $\mathrm{ECRPQ}^{\neg}$ without making query evaluation more expensive? We provide next a strong negative answer to this question. Indeed, we show in Theorem 8.8 that the problem of query evaluation for the class $\mathrm{ECRPQ}^{\neg}$, extended with a very light form of counting the number of occurrences of events, becomes undecidable. Recall that Theorem 8.1 states, on the other hand, that query evaluation for the class $\mathrm{ECRPQ}^{\neg}$ is decidable.

Linear constraints on lengths of paths. One important application of ECRPQs with linear constraints on the number of occurrences of events is for comparing path lengths. We define a class of CRPQ*s with linear constraints on lengths of paths* as

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i),\ \bigwedge_{1 \le j \le t} L_j(\omega_j),\ \mathbf{A}\bar{\ell} \ge \mathbf{b}\,,$$

where $Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i), \bigwedge_{1 \le j \le t} L_j(\omega_j)$ is a CRPQ as in (1), and

— $\mathbf{A}$ is a $k \times m$ matrix over $\mathbb{Z}$, for some $k > 0$;
— $\mathbf{b}$ is a vector in $\mathbb{Z}^k$; and
— $\bar{\ell} = (\ell_1, \ldots, \ell_m)$.

The semantics is extended as follows: each ℓ_i is interpreted as the length of the path π_i, for $i \le m$. The last clause of the query is true if $\mathbf{A}\ell \ge \mathbf{b}$ under this interpretation. That is, we extend CRPQs with the ability to add $k > 0$ linear constraints on lengths of paths. Observe that the length of a path ρ in a graph database over the alphabet $\Sigma = \{a_1, \ldots, a_k\}$ coincides with the sum $\sum_{j=1}^{k} |\rho|_{a_j}$. This implies that ECRPQs with linear constraints on lengths of paths can be easily expressed as ECRPQs with linear constraints on the number of occurrences of events, but the contrary is not true. Hence the former class of queries can be considered to be a proper restriction of the latter.

In the same way that ECRPQs were extended with negation in Section 8.1, we can add negation to the class of ECRPQs with linear constraints on the lengths of paths. We omit the definition of such an extension for the sake of simplicity, but both the syntax and the semantics can be easily obtained from the corresponding syntax and semantics of the following classes of queries: ECRPQs with linear constraints on the lengths of paths and $\mathrm{ECRPQ}^{\neg}$.

We shall conclude this section by showing that adding both negation and linear constraints on path lengths to ECRPQs yields undecidability, therefore showing the tightness of the extensions that we obtained earlier in this section. In fact, undecidability holds for a fixed formula in this extended query language.

THEOREM 8.8. *There exists a fixed and Boolean* ECRPQ *query φ with negation and linear constraints on lengths of paths, such that it is undecidable to check whether φ holds in a graph database G.*

PROOF. The reduction is from the acceptance problem for a fixed and deterministic Turing machine (TM). Suppose that the fixed TM is $\mathcal{M} = (\Sigma, \Gamma, Q, \delta, q_0, q_F)$, where $\Sigma = \{0, 1\}$ is the input alphabet, $\Gamma = \Sigma \cup \{\mathtt{B}\}$ is the tape alphabet consisting of the input alphabet and the blank symbol $\mathtt{B}$, Q is a finite set of control states, $q_0 \in Q$ (resp. $q_F \in Q$) is the initial (resp. final) state, and $\delta : Q \times \Gamma \rightarrow (Q \times \Gamma \times D)$ is the transition function where $D = \{L, R\}$ indicates the direction of the movement of the head of the TM (L for left and R for right). We assume without loss of generality that $q_0 \neq q_F$.

A configuration C of $\mathcal{M}$ is a string in the regular language $\Sigma^*Q\Sigma^*\mathtt{B}^*$. For convenience, a configuration $vqw\mathtt{B}\cdots\mathtt{B}$, where $v, w \in \Sigma^*$ and $q \in Q$, may be thought of as follows. The tape of $\mathcal{M}$ contains the word vw and the head of $\mathcal{M}$ is scanning the last letter of v in state q. The length of C is the number of symbols in C that are not in Q. A sequence of configurations of $\mathcal{M}$ representing a computation of $\mathcal{M}$ on an input word $w \in \Sigma^*$ can then be represented as a string of the form $\pi = C_1\#_{\gamma_1}C_2\#_{\gamma_2}\ldots\#_{\gamma_{m-1}}C_m$, where each $\#_\gamma$ is a delimiter symbol with $\gamma \in \{L, R\}$, C_1 is $q_0w\mathtt{B}^k$, the initial configuration of length $n = |w| + k$ (for some $k \geq 0$ depending on w), C_m is $q_F\mathtt{B}^n$, the final configuration of length n (we assume that on acceptance the tape content is erased), and each C_i is a configuration of $\mathcal{M}$ of length n. Since the number k is existentially quantified, imposing a space upper bound on the tape of the TM of size k is not a limitation. As we will soon see, the subscript γ (i.e. left/right) for a delimiter symbol is used to indicate the direction of the movement of the head of the TM. In particular, given a substring $C_i\#_\gamma C_{i+1}$ of π, we shall use γ to indicate that the position of the head of the TM in C_{i+1} has moved in the direction γ with respect to the head of the TM in C_i. In the following, we use the symbol $\#$ as an abbreviation (or macro) for the regular expression $(\#_L + \#_R)$.

We now proceed with the reduction. The input to the problem is a string $w \in \{0, 1\}^*$ and the problem is to check whether w is accepted by $\mathcal{M}$. We construct a graph G over the alphabet $\Gamma \cup Q \cup \{\#_L, \#_R\}$ and a fixed formula φ such that w is in the language of $\mathcal{M}$ iff $G \models \varphi$. The main trick for the reduction is to "hardwire" w into the graph G and make the formula φ guess an accepting computation π of $\mathcal{M}$ on w. Notice that such a computation is a string in the regular language generated by the regular expression

$$E := q_0w\mathtt{B}^*\#(\Sigma^*Q\Sigma^*\mathtt{B}^*\#)^*q_F\mathtt{B}^*.$$

We will embed E (construed as an NFA) into G as a subgraph with distinguished nodes. In addition to checking that the guessed computation π is in $L(E)$, we also need to check that each configuration in π has the same length n. This can be expressed in the fixed formula φ as follows:

(1) Let v_0 be the longest prefix of π of the form $q_0\Sigma^*\mathtt{B}^*$. (This will ensure that v_0 coincides with the first configuration in π before any occurrence of the symbol $\#$. This requires at least three path quantifiers which alternate).
(2) Assert that the last configuration is of length $|v_0|$: if v is the longest prefix of π with $\#$ as the last symbol, then $|\pi| - |v| = |v_0|$.
(3) Assert that each intermediate configuration is of length $|v_0|$: for each pair of prefixes $v \prec v'$ of π both ending with the symbol $\#$ such that there is precisely one occurrence of the symbol $\binom{\perp}{\#}$ in the convolution $v \otimes v'$, it is the case that $|v'| - |v| - 1 = |v_0|$.
(4) Assert that for each substring $C\#_\gamma C'$ of π, where C, C' are configurations of $\mathcal{A}$, it is the case that the head position in C' is one position to the left/right of the head position in C and agrees with γ. To this end, we universally guess two prefixes $v \preceq v'$ of π such that (i) v ends with $\#_\gamma$, (ii) the convolution $v \otimes v'$ has precisely one symbol of the form $\binom{\perp}{\#}$ and (iii) the smallest prefix v'' of π such that $v' \prec v''$

ends with the symbol $\#$ (i.e. the next letter after v' is a delimiter symbol). This means that v ends with $C\#_\gamma$ and the suffix of v' that comes after v represents the configuration C'. We shall only show how to express this assertion for the case when the state symbol in C is not in the left/right-most cell of the tape leaving the reader to handle the exceptions as simple exercises. Let u and u' be the longest prefixes of v and v' ending in symbols in Q. In the case when $\gamma = L$, we assert that $|u'| = |u| + |v_0| - 1 + 1$ (note that there is the delimiter symbol $\#_\gamma$). In the case when $\gamma = R$, we assert that $|u'| = |u| + |v_0| + 2$.

(5) Assert that for each substring $C\#_\gamma C'$ of π, where C, C' are configurations of $\mathcal{A}$, it is the case that the configuration C' follows from its previous configuration C (with respect to the transition function δ). We first give the rough idea of how to assert this. Suppose that $abqc$ (resp. $a'b'q'c'$) is a substring of C (resp. C'), where $a, a', b, b'c, c' \in \Gamma$ and $q, q' \in Q$ (of course, we also need to take into account some tedious exceptions, e.g., if C ends with a state symbol, then we may assume that c is the symbol $\#$). Leaving the reader to handle the exceptions as exercises, we may assume that all a, b, c are in Γ. Then, we will make the first assertion (A1): either $\delta(q, b) = (q', c', L)$ and $\gamma = L$ OR $\delta(q, b) = (q', a', R)$ and $\gamma = R$. The second assertion (A2) is that: other symbols in C must stay the same in C'. The first assertion can be done by guessing two prefixes v and v' of π, where v ends with $C\#_\gamma$ and v' ends with C', and constructing a *fixed*[1] NFA running on the convolution $v \otimes v'$, i.e., the NFA will need to remember the symbols $abqc$ and $a'b'q'c'$ in the finite control and check these against the transition function δ as we previously described, while making sure of seeing precisely one occurrence of $\binom{\perp}{\#_\gamma}$ (and no occurrence of $\binom{\perp}{\#_{\gamma'}}$ where $\gamma' \in D - \{\gamma\}$). The second assertion can be expressed in the same way as the previous item.

Correctness of our reduction is immediate. Observe also that the construction of the formula φ depends *solely* on the TM $\mathcal{A}$, which is not part of the input. Therefore, the constructed formula φ is fixed and takes constant time to compute. Notice also that the input string w is now part of the graph database G. □

9. COMPARISON WITH OTHER LANGUAGES

The language of ECRPQs was motivated by recent proposals of graph database queries that can compare and output paths. One of them is the language of ρ-queries [Anyanwu and Sheth 2003], which allows to specify and check properties based on specific semantic associations. It is not possible to directly compare the language of ρ-queries with the languages of ECRPQs, since the former is defined over a different data model (namely, RDF/S). However, at a high level we can see that ECRPQs have more flexibility when it comes to defining semantic associations among paths. In fact, we can make use of arbitrary regular relations, while ρ-queries are defined with respect to a predetermined set of similarity measures.

Several other navigational languages for path specifications can be found in the graph database and RDF literature. While many of these languages can check whether the label of a path belongs to a particular regular language, none of them have the facilities for comparing labels of different paths, which is a distinctive feature of the ECRPQs. An example is the language of *nested* regular expressions [Barceló et al. 2012], which forms the basis of the navigational RDF language nSPARQL [Pérez et al. 2010]. It is not hard to see that nested regular expressions and ECRPQs are incomparable in terms of their expressive power. In fact, while nested regular expressions permit

[1] Recall that the TM is not part of the input.

				Acyclic		ECRPQ
Complexity	CQs	CRPQ	ECRPQ	CRPQ	ECRPQ	Q_{len}
data	AC^0	NL-c.	NL-c.	NL-c.	NL-c.	NL-c.
combined	NP-c.	NP-c.	PSPACE-c.	PTIME	PSPACE-c.	NP-c.

(a) CQs, CRPQs, ECRPQs, and subclasses

Complexity	with repetitions		with negation		CRPQ + linear constraints
	CRPQ	ECRPQ	CRPQ	ECRPQ	
data	NL-c.	NL-c.	NL-c.	non-elementary	PTIME
combined	PSPACE-c.	PSPACE-c.	PSPACE-c.		NP-c.

(b) Extensions of CRPQs and ECRPQs

Fig. 1. Summary of complexity results for classes of queries

complex branching patterns over graph-structured data that are inexpressible in the language of ECRPQs, even the simplest path comparisons are beyond reach for the language of nested regular expressions. The same is true for several other important navigational RDF languages, such as the language of SPARQL extended with regular expression patterns, PSPARQL [Alkhateeb et al. 2009], and its extended version with constraints, CPSPARQL [Alkhateeb et al. 2008].

While the usual RDF query language SPARQL provides limited navigational capabilities over the data, its more recent version, SPARQL 1.1, allows the specification of *property paths*. These are essentially regular expressions that specify pairs of nodes linked by a path whose label belongs to the expression. While SPARQL 1.1 can specify complex path expressions, in its current form it completely lacks operations for path comparisons. One of the goals of our work is to sufficiently advance the foundations of the query languages that compare paths, in order to draw a clear picture of what could and could not be implemented by the W3C (and others) in the future.

10. SUMMARY

The tables in Figure 1 give a summary of the complexity results; they also include, for comparison, known results on CQs and CRPQs in the first two columns of (a). We use the abbreviation NL for NLOGSPACE; '-c.' means, of course, that the problem is complete for the class. In the case of data complexity, it means that the problem is in a given class for all queries, and can be hard for the class for some of them. In addition to these complexity results, our technical results also include algorithms for query evaluation and representation of paths in the output as well as results on query containment.

ELECTRONIC APPENDIX

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

REFERENCES

Abiteboul, S., Buneman, P., and Suciu, D. 1999. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kauffman.

Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. 1997. The LOREL query language for semistructured data. *International Journal on Digital Libraries 1,* 1, 68–88.

Abulla, P., Jonnson, B., Nilsson, M., and Saksena, M. 2003. A survey of regular model checking. In *5th International Conference on Concurrence Theory (CONCUR)*. 35–48.

ALKHATEEB, F., BAGET, J.-F., AND EUZENAT, J. 2008. Constrained regular expressions in SPARQL. In *2008 International Conference on Semantic Web & Web Services (SWWS)*. 91–99.

ALKHATEEB, F., BAGET, J.-F., AND EUZENAT, J. 2009. Extending SPARQL with regular expression patterns (for querying RDF). *Journal of Web Semantics 7,* 2, 57–73.

ANYANWU, K., MADUKO, A., AND SHETH, A. P. 2007. SPARQ2L: Towards support for subgraph extraction queries in RDF databases. In *16th International World Wide Web Conference (WWW)*. 797–806.

ANYANWU, K. AND SHETH, A. 2003. ρ-queries: enabling querying for semantic associations on the semantic web. In *12th International World Wide Web Conference (WWW)*. 690–699.

BARCELÓ, P., HURTADO, C. A., LIBKIN, L., AND WOOD, P. T. 2010. Expressive languages for path queries over graph-structured data. In *29th ACM Symposium on Principles of Database Systems (PODS)*. 3–14.

BARCELÓ, P., PÉREZ, J., AND REUTTER, J. 2012. Relative expressiveness of nested regular expressions. In *6th Alberto Mendelzon Workshop on the Foundations of Data Management and the Web (AMW)*. 180–195.

BARRETT, C., JACOB, R., AND MARATHE, M. 2000. Formal-language-constrained path problems. *SIAM Journal on Computing 30,* 3, 809–837.

BENEDIKT, M., LIBKIN, L., SCHWENTICK, T., AND SEGOUFIN, L. 2003. Definable relations and first-order query languages over strings. *Journal of the ACM 50,* 5, 694–751.

BERSTEL, J. 1979. Transductions and Context-Free Languages. B. G. Teubner.

BLUMENSATH, A. AND GRÄDEL, E. 2000. Automatic structures. In *15th Annual IEEE Symposium on Logic in Computer Science (LICS)*. 51–62.

BRUYÈRE, V., HANSEL, G., MICHAUX, C., AND VILLEMAIRE, R. 1994. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society 1,* 191–238.

CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, M. 2000. Containment of conjunctive regular path queries with inverse. In *7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 176–185.

CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, M. 2002. Rewriting of regular expressions and regular path queries. *Journal of Computer and Systems Sciences 64,* 3, 443–465.

CHROBAK, M. 1986. Finite automata and unary languages. *Theoretical Computer Science 47,* 2, 149–158.

CONSENS, M. AND MENDELZON, A. 1990. GraphLog: A visual formalism for real life recursion. In *9th ACM Symposium on Principles of Database Systems (PODS)*. 404–416.

DEUTSCH, A. AND TANNEN, V. 2001. Optimization properties for classes of conjunctive regular path queries. In *8th International Workshop on Database Programming Languages (DBPL)*. 21–39.

ELGOT, C. AND MEZEI, J. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development 9,* 1, 47–68.

FLORESCU, D., LEVY, A., AND SUCIU, D. 1998. Query containment for conjunctive queries with regular expressions. In *17th ACM Symposium on Principles of Database Systems (PODS)*. 139–148.

FREYDENBERGER, D. AND REIDENBACH, D. 2010. Bad news on decision problems for patterns. *Information and Computation 208,* 1, 83–96.

FREYDENBERGER, D. AND SCHWEIKARDT, N. 2011. Expressiveness and static analysis of extended conjunctive regular path queries. In *5th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*.

FROUGNY, C. AND SAKAROVITCH, J. 1991. Rational relations with bounded delay. In *8th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*. 50–63.

GRAHNE, G. AND THOMO, A. 2004. Query answering and containment for regular path queries under distortions. In *3rd International Symposium on the Foundations of Information and Knowledge Systems (FoIKS)*. 98–115.

GUSFIELD, D. 1997. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press.

HOLLAND, D., BRAUN, U., MACLEAN, D., MUNISWAMY-REDDY, K., AND SELTZER, M. 2008. Choosing a data model and query language for provenance. In *2nd International Provenance and Annotation Workshop (IPAW)*.

IBARRA, H., SU, J., DANG, Z., BULTAN, T., AND KEMMERER, R. 2002. Counter machines and verification problems. *Theoretical Computer Science 289,* 1, 165–189.

KANZA, Y. AND SAGIV, Y. 2001. Flexible queries over semistructured data. In *20th ACM Symposium on Principles of Database Systems (PODS)*. 40–51.

KOCHUT, K. AND JANIK, M. 2007. SPARQLeR: Extended SPARQL for semantic association discovery. In *4th European Semantic Web Conference (ESWC)*. 145–159.

KOZEN, D. 1977. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*. 254–266.

LEE, W., RASCHID, L., SRINIVASAN, P., SHAH, N., RUBIN, D., AND NOY, N. 2007. Using annotations from controlled vocabularies to find meaningful associations. In *4th International Workshop on Data Integration in the Life Sciences (DILS)*. 247–263.

LEHMANN, J., SCHÜPELL, J., AND AUER, S. 2007. Discovering unknown connections—the DBpedia relationship finder. In *1st SABRE Conference on Social Semantic Web*. 99–110.

LENSTRA, H. 1983. Integer programming in a fixed number of variables. *Mathematical Operational Research 8,* 4, 538–548.

MENDELZON, A. AND WOOD, P. 1995. Finding regular simple paths in graph databases. *SIAM Journal on Computing 24,* 6, 1235–1258.

MILO, T. AND SUCIU, D. 1999. Index structures for path expressions. In *7th International Conference on Database Theory (ICDT)*. 277–295.

PAPADIMITRIOU, C. 1981. On the complexity of integer programming. *Journal of the ACM 28,* 4, 765–768.

PÉREZ, J., ARENAS, M., AND GUTIERREZ, C. 2010. nSPARQL: A navigational language for RDF. *Journal of Web Semantics 8,* 4, 255–270.

ROBERTSON, E. L. 1974. Structure of complexity in the weak monadic second-order theories of the natural numbers. In *Conference Record of Sixth Annual ACM Symposium on Theory of Computing*. 161–171.

SCARPELLINI, B. 1984. Complexity of subcases of Presburger arithmetic. *Transactions of the AMS 284,* 203–218.

SHETH, A., ALEMAN-MEZA, A. B., B., A. I., BERTRAM, C., WARKE, Y., RAMAKRISHNAN, C., HALASCHEK, C., ANYANWU, K., AVANT, D., ARPINAR, F., AND KOCHUT, K. 2005. Semantic association identification and knowledge discovery for national security applications. *Journal of Database Management 16,* 1, 33–53.

STOCKMEYER, L. J. 1974. The complexity of decision problems in automata theory and logic. Ph.D. thesis, Massachusetts Institute of Technology.

TO, A. 2009. Unary finite automata vs. arithmetic progressions. *Information Processing Letters 109,* 17, 1010–1014.

TO, A. 2010. Model checking infinite-state systems: Generic and specific approaches. Ph.D. thesis, LFCS, School of Informatics, University of Edinburgh.

VERMA, K., SEIDL, H., AND SCHWENTICK, T. 2005. On the complexity of equational Horn clauses. In *20th International Conference on Automated Deduction (CADE)*. 337–352.

WEIKUM, G., KASNECI, G., RAMANATH, M., AND SUCHANEK, F. 2009. Database and information-retrieval methods for knowledge discovery. *Communications of the ACM 52,* 4, 56–64.

Received Month Year; revised Month Year; accepted Month Year

Online Appendix to:
Expressive Languages for Path Queries over Graph-Structured Data

PABLO BARCELÓ, University of Chile
LEONID LIBKIN, University of Edinburgh
ANTHONY W. LIN, Univerity of Oxford
PETER T. WOOD, Birkbeck, University of London

A. PROOF OF PROPOSITION 3.2 FOR BOOLEAN QUERIES

The alphabet is $\Sigma = \{a, b, c, d\}$ and the query is

$$Q \ := \ Ans() \leftarrow \bigwedge_{1 \le i \le 3} (x_i, \pi_i, x_{i+1}), (y_i, \pi'_i, y_{i+1}),$$
$$c(\pi_1), c(\pi_3), d(\pi'_1), d(\pi'_3), a^+(\pi_2), b^+(\pi'_2), \mathrm{el}(\pi_2, \pi'_2).$$

Given a Σ-labeled graph database G, this query asks whether there exist nodes $u, v, u', v' \in V$ such that:

(i). v is reachable from u in G by a path ρ_1 labeled in a^+;
(ii). v' is reachable from u' in G by a path ρ_2 labeled in b^+;
(iii). the length of ρ_1 equals the length of ρ_2;
(iv). there exists a c-labeled edge entering u (resp. v) in G; and
(v). there exists a d-labeled edge leaving u' (resp. v') in G.

Assume, for the sake of contradiction, that there is CRPQ Q' that is equivalent to Q; that is, $Q(G) = \mathtt{true} \Leftrightarrow Q'(G) = \mathtt{true}$ for every Σ-labeled graph database G. Further, suppose that Q' is of the form $Ans() \leftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i), \bigwedge_{1 \le j \le t} L_j(\omega_j)$.

For each $n, p > 0$, let $G_{n,p}$ be the following Σ-labeled graph database. First, its set of nodes V is

$$\{e_0, e_1, \ldots, e_n, e_{n+1}, f_0, f_1, \ldots, f_p, f_{p+1}\}.$$

Second, there is an edge in G from e_j to e_{j+1}, for each $0 \le j \le n$, and from f_j to f_{j+1}, for each $0 \le j \le p$. Third, the edges (e_0, e_1) and (e_n, e_{n+1}) are labeled c, the edges (f_0, f_1) and (f_p, f_{p+1}) are labeled d, each edge (e_j, e_{j+1}) is labeled a $(1 \le j \le n)$, and each edge (f_j, f_{j+1}) is labeled b $(1 \le j \le p)$. Clearly, $Q(G_{n,n}) = \mathtt{true}$, for each $n > 1$, and, thus, by assumption, also $Q'(G_{n,n}) = \mathtt{true}$.

We denote by $G^e_{n,p}$ and $G^f_{n,p}$ the subgraphs of $G_{n,p}$ induced by $\{e_0, e_1, \ldots, e_{n+1}\}$ and $\{f_0, f_1, \ldots, f_{p+1}\}$, respectively. Recall that m and t are, respectively, the number of atoms in the relational part of Q and the number of regular languages used in Q. Let $S \subseteq \{1, \ldots, m\}$. We denote by S_{reg} the subset of $\{1, \ldots, t\}$ that contains all those indexes i such that $\omega_i = \pi_j$, for some $j \in S$. It is not hard to see that for each $S \subseteq \{1, \ldots, m\}$ there exist regular languages R_S and $R_{\bar{S}}$ over Σ such that

$$Ans() \leftarrow \bigwedge_{j \in S} (x_j, \pi_j, y_j), \bigwedge_{i \in S_{reg}} L_i(\omega_i) \quad \text{(resp. } \bigwedge_{j \notin S} (x_j, \pi_j, y_j), \bigwedge_{i \notin S_{reg}} L_i(\omega_i))$$

evaluates to $\mathtt{true}$ over $G^e_{n,p}$ (resp. $G^f_{n,p}$) iff the label of the path from e_0 to e_{n+1} in $G^e_{n,p}$ (resp. from f_0 to f_{p+1} in $G^f_{n,p}$) belongs to R_S (resp. $R_{\bar{S}}$).

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Let $\mathcal{A}_S$ and $\mathcal{A}_{\bar{S}}$ be NFAs recognizing R_S and $R_{\bar{S}}$, for $S \subseteq \{1,\dots,m\}$. Let q be the maximum number of states in an automaton of the form $\mathcal{A}_S$ or $\mathcal{A}_{\bar{S}}$, for $S \subseteq \{1,\dots,m\}$. Choose an arbitrary $r > q$. Since $Q'(G_{r,r}) = \texttt{true}$ and $G_{r,r}$ is formed by the disjoint union of $G^e_{r,r}$ and $G^f_{r,r}$, there exists an $S^r \subseteq \{1,\dots,m\}$ such that (1) the evaluation of $\bigwedge_{j\in S^r}(x_j,\pi_j,y_j), \bigwedge_{i\in S^r_{reg}} L_i(\omega_i)$ in $G^e_{r,r}$ is $\texttt{true}$, (2) the evaluation of $\bigwedge_{j\not\in S^r}(x_j,\pi_j,y_j), \bigwedge_{i\not\in S^r_{reg}} L_i(\omega_i)$ in $G^f_{r,r}$ is also $\texttt{true}$, and (3) for every two atoms (x_j,π_j,y_j) and $(x_{j'},\pi_{j'},y_{j'})$ $(1 \leq j,j' \leq m)$ that share at least one node variable, it is the case that $(x_j,\pi_j,y_j) \in S^r \Leftrightarrow (x_{j'},\pi_{j'},y_{j'}) \in S^r$.

It follows from conditions (1) and (2) that the label of the unique maximal path in $G^e_{r,r}$ belongs to R_{S^r} and the label of the unique maximal path in $G^f_{r,r}$ belongs to $R_{\overline{S^r}}$. By using a standard pumping argument there exists $r' > r$ such that the label of the unique maximal path in $G^e_{r',r}$ belongs to R_{S^r} and the label of the unique maximal path in $G^f_{r',r}$ belongs to $R_{\overline{S^r}}$. From condition (3) it follows that Q' evaluated on $G_{r',r}$ is $\texttt{true}$, and, thus, that $Q(G_{r',r}) = \texttt{true}$, which is a contradiction.

B. PROOF OF PROPOSITION 6.2.

We only have to prove membership since the query evaluation problem is NP-hard already for the class of conjunctive queries over directed graphs. Let Σ be a finite alphabet, $G = (V,E)$ a Σ-labeled graph database, Q an CRPQ over Σ of the form,

$$Ans(\bar{z},\bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq m}(x_i,\pi_i,y_i), \bigwedge_{1\leq j\leq t} L_j(\omega_j),$$

$\bar{v}$ a tuple of nodes in G such that $|\bar{v}| = |\bar{z}|$, and $\bar{\rho}$ a tuple of paths in G such that $|\bar{\chi}| = |\bar{\rho}|$.

A *witness* for $(\bar{v},\bar{\rho})$ wrt Q and G is a tuple $(u_1,\dots,u_m,u'_1,\dots,u'_m,\eta_1,\dots,\eta_m)$ that satisfies the following:

(1) Each u_i and u'_i $(i \in [1,m])$ is a node of G and the pair $((u_1,\dots,u_m),(u'_1,\dots,u'_m))$ of nodes in G^m is Q-compatible.
(2) Each η_i $(i \in [1,m])$ is a path in G from u_i to u'_i.
(3) If the variable z is the j-th element of $\bar{z}$ and $z = x_i$ (resp. y_i), for $i \in [1,m]$, then u_i (resp. u'_i) corresponds to the j-th element of $\bar{v}$.
(4) If the variable χ is the j-th element of $\bar{\chi}$ and $\chi = \pi_i$, for $i \in [1,m]$, then η_i corresponds to the j-th element of $\bar{\rho}$;
(5) For each $1 \leq i \leq m$, $\lambda(\eta_i)$ satisfies each regular expression L_j $(1 \leq j \leq t)$ such that $\omega_j = \pi_i$.

It is not hard to see that $(\bar{v},\bar{\rho}) \in Q(G)$ if and only if there is a witness for $(\bar{v},\bar{\rho})$ wrt Q and G.

It is well known that an NFA $\mathcal{A}_j$ that accepts exactly the set strings that satisfy L_j can be constructed in polynomial time from L_j, for each $1 \leq j \leq m$. We denote by Q_j the set of states of $\mathcal{A}_j$. In particular, Q_j is of polynomial size in L_j.

We first prove that if there is a witness for $(\bar{v},\bar{\rho})$ wrt Q and G then there is a witness in which each path is of length at most $|V| \times \max\{|Q_j| \mid j \in [1,t]\} \times \Sigma + 3$ (by the length of the path we mean the number of nodes from G in the path).

Let $(u_1,\dots,u_m,u'_1,\dots,u'_m,\eta_1,\dots,\eta_m)$ be a witness for $(\bar{v},\bar{\rho})$ wrt Q and G, and assume for the for $1 \leq i \leq m$ the path $\eta_i = v_1a_1v_2a_2v_3\cdots v_{\ell_i}a_{\ell_i}v_{\ell_i+1}$ is of length $\ell_i + 1 > |V| \times \max\{|Q_i| \mid i \in [1,m]\} \times \Sigma + 3$. (Notice that $v_1 = u_i$ and $v_{\ell_{i+1}} = u'_i$). Let $\theta_i : \{0,1,\dots,\ell_i\} \to Q_i$ be an accepting run of $\mathcal{A}_i$ over $\lambda(\eta_i)$ (we assume that $\theta_i(0)$ is the initial state of $\mathcal{A}_i$). Since $\ell_i + 1 > |V| \times \max\{|Q_i| \mid i \in [1,m]\} \times \Sigma + 3$ there are two integers $1 \leq j < j' < \ell_i$ such that $\theta_i(j-1) = \theta_i(j'-1)$, $v_j = v_{j'}$ and $a_j = a_{j'}$. Thus, $\eta'_i = v_1a_1v_1\cdots v_{j-1}a_{j-1}v_ja_{j'}v_{j'+1}\cdots v_{\ell_i}a_{\ell_i}v_{\ell_i+1}$ is also a path from u_i

to u'_i that satisfies L_i, and the length of η'_i is strictly less than the length of η_i. Thus, $(u_1, \ldots, u_m, u'_1, \ldots, u'_m, \eta_1, \ldots, \eta'_i, \eta_{i+1}, \ldots, \eta_m)$ is also a witness for $(\bar{v}, \bar{\rho})$ wrt Q and G. By iteratively continuing with this process we will end up with a witness for $(\bar{v}, \bar{\rho})$ wrt Q and G in which each path is of length at most $|V| \times \max\{|Q_i| \mid i \in [1,m]\} \times \Sigma + 3$.

It is now easy to define an algorithm that works in nondeterministic polynomial time and checks whether $(\bar{v}, \bar{\rho}) \in Q(G)$. The algorithm first guesses a tuple of the form

$$(u_1, \ldots, u_m, u'_1, \ldots, u'_m, \eta_1, \ldots, \eta_m),$$

such that each u_i and u'_i ($i \in [1,m]$) is a node in V and each η_i ($i \in [1,m]$) is a path from u_i to u'_i of length at most $|V| \times \max\{|Q_i| \mid i \in [1,m]\} \times \Sigma + 3$. Then the algorithm checks whether $(u_1, \ldots, u_m, u'_1, \ldots, u'_m, \eta_1, \ldots, \eta_m)$ is a witness for $(\bar{v}, \bar{\rho})$ wrt Q and G. This can easily be done in polynomial time.

C. PROOF OF PROPOSITION 8.7

The proof follows the main ideas used in the proofs of Theorems 6.7 and 8.5, but instead we use a representation of Parikh images by Presburger formulae more complex than unions of arithmetic progressions. Let Γ be an arbitrary k-letter alphabet, and L a regular language over Γ. Then the Parikh image of L, i.e., $\text{par}(L) = \{\text{par}(s) \mid s \in L\} \subseteq \mathbb{N}^k$ is known to be a semi-linear set and thus definable in Presburger arithmetic. According to [Verma et al. 2005], this image is definable by an existential Presburger formula $\psi^L_{\text{par}}(x_1, \ldots, x_n)$ which can be computed in polynomial time from an NFA defining L.

Suppose we have an m-ary regular relation R over an alphabet Σ. Applying the previous observation to the alphabet $(\Sigma_\perp)^m$ with $N = (|\Sigma| + 1)^m$ symbols, we get an existential Presburger formula $\psi^R_{\text{par}}(x_1, \ldots, x_N)$ for the Parikh image. From this formula we easily obtain another existential formula $\psi_R(x_{11}, \ldots, x_{nk})$, where $k = |\Sigma|$, which is satisfied on a tuple (a_{ij}) iff for some $(s_1, \ldots, s_m) \in R$, it is the case that a_{ij} is the number of occurrences of the jth letter of Σ in s_i. Indeed, ψ_R starts with a block of existential quantifiers $\exists x_1, \ldots, \exists x_N$ followed by a conjunction of linear constraints stating that x_{ij} is the sum of x_l's corresponding to those symbols in $(\Sigma_\perp)^m$ that contain the jth letter of Σ in the ith position. Note that for each fixed m, the formula ψ_R is constructed in polynomial time.

We next proceed as in the proof of Theorem 6.7. For a given ECRPQ in which all relations R were replaced with R_{par}, we guess a set of nodes witnessing the query as well as an ordered partition for the set of paths and assume that witnessing paths length-conform to it. Then, again as in that proof, we guess nodes on every path that mark the lengths of smaller paths in the set, and split each variable x_{ij} into several according to those points. Again, taking as an example an ordered partition $\{1,2\} < \{3\} < \{4\}$ for paths $\pi_1, \pi_2, \pi_3, \pi_4$, for each x_{4j} we introduce variables x'_{4j}, x''_{4j}, and x'''_{4j} to indicate the numbers of occurrences of the jth letter in the portion of the path π_4 consisting of the first $|\pi_1|$ positions, then positions from $|\pi_1| + 1$ to $|\pi_3|$, and finally the remaining positions. We modify ψ_R accordingly by quantifying over these new variables and then stating that $x_{4j} = x'_{4j} + x''_{4j} + x'''_{4j}$, and likewise for all other variables. Note that the resulting modification is still an existential Presburger formula, which is polynomial in size if the arities of relations are fixed.

As before, introducing these intermediate points lets us compare path lengths for equality only, under the assumption that they length-conform to a given ordered partition. We thus reduce the problem to the following. Given nodes $a_1, \ldots, a_n$ in a graph database, an existential Presburger ψ formula over variables x_{ijp}, where $i, j \leq n$, and $p \leq |\Sigma|$, and a family $\mathcal{S}$ of subsets of $\{1, \ldots, n\} \times \{1, \ldots, n\}$, do there exist paths π_{ij} between a_i and a_j for $i, j \leq n$ so that ψ holds when each x_{ijp} is interpreted as the number of occurrences of the pth letter in π_{ij}, and for every (i, j) and (i', j') in the same set of

$\mathcal{S}$, the length of the paths between a_i and a_j and between $a_{i'}$ and $a_{j'}$ are the same? We must now show that this problem is solvable in NP.

To do so, for every i, j we look at the graph as an automaton (G, a_i, a_j) and use the algorithm of [Verma et al. 2005] to produce an existential Presburger formula $\psi_{ij}(x_{ij1}, \ldots, x_{ijk})$ for the Parikh image of paths between a_i and a_j. We then check if the formula

$$\exists (x_{ijp})_{i,j \leq n,\ p \leq k} \Bigg(\bigwedge_{i,j=1}^{n} \psi_{ij}(x_{ij1}, \ldots, x_{ijk}) \ \wedge\ \psi\big((x_{ijp})_{ijp}\big) \wedge \bigwedge_{S \in \mathcal{S}} \ \bigwedge_{(i,j),(i',j') \in S} \Big(\sum_{p \leq k} x_{ijp} = \sum_{p \leq k} x_{i'j'p} \Big) \Bigg) \tag{8}$$

is satisfiable. This is an existential formula, and by the above, it is constructible in polynomial time for fixed arities of regular relations. Thus one can check satisfiability in NP. Again, as in the proof of Theorem 6.7, all the guesses can be combined, and the whole algorithm runs in NP.

For data complexity, note that the number of guessed tuples of nodes and partitions is fixed if the query is fixed, and so we need to check a fixed number of formulae of the form (8). Note that in that formula, the number n comes from the query, and k is the size of the alphabet, so the number of variables is fixed. Then the formula is reduced to solving integer linear programs in fixed dimension [Lenstra 1983], which therefore implies the PTIME bound on data complexity.

The proof for queries of the form Q°_{par} follows closely the proof of Theorem 8.5. We start by computing in polynomial time automata defining every projection of every regular relation R, followed by applying the algorithm of [Verma et al. 2005] to compute existential Presburger formulae for Parikh images of such automata, and then essentially follow the proof of Theorem 8.5, in which the condition $\mathbf{A}\bar{\ell} \geq \mathbf{b}$ can be replaced by an existential Presburger formula without affecting the complexity. This completes the proof of the proposition.