Querying Regular Graph Patterns

PABLO BARCELÓ, Universidad de Chile
LEONID LIBKIN, University of Edinburgh
JUAN L. REUTTER, University of Edinburgh and Pontificia Universidad Católica de Chile

Graph data appears in a variety of application domains, and many uses of it, such as querying, matching, and transforming data, naturally result in incompletely specified graph data, that is, graph patterns. While queries need to be posed against such data, techniques for querying patterns are generally lacking, and properties of such queries are not well understood.

Our goal is to study the basics of querying graph patterns. The key features of patterns we consider here are node and label variables and edges specified by regular expressions. We provide a classification of patterns, and study standard graph queries on graph patterns. We give precise characterizations of both data and combined complexity for each class of patterns. If complexity is high, we do further analysis of features that lead to intractability, as well as lower-complexity restrictions. Since our patterns are based on regular expressions, query answering for them can be captured by a new automata model. These automata have two modes of acceptance: one captures queries returning nodes, and the other queries returning paths. We study properties of such automata, and the key computational tasks associated with them. Finally, we provide additional restrictions for tractability, and show that some intractable cases can be naturally cast as instances of constraint satisfaction problems.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata (e.g., finite, push-down, resource-bounded)*; H.2.1 [**Database Management**]: Logical Design—*Data models*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Graph databases, graph patterns, query languages, complexity, automata, constraint satisfaction

ACM Reference Format:
Barceló, P., Libkin, L., and Reutter, J. L. 2014. Querying regular graph patterns. *J. ACM* 61, 1, Article 8 (January 2014), 54 pages.
DOI: http://dx.doi.org/10.1145/2559905

1. INTRODUCTION

Querying and mining graph-structured data has received much attention lately, due to numerous applications in areas such as biological networks [Leser 2005; Milo et al. 2002; Olken 2003], social networks [Ronen and Shmueli 2009; San Martín and

Partial support for this work was provided by Fondecyt grant 1110171, EPSRC grant G049165, and FET-Open Project FoX, grant agreement 233599.
Part of this work was done when P. Barceló visited Edinburgh and L. Libkin and J. L. Reutter visited Santiago.
Authors' addresses: P. Barceló, Department of Computer Science, University of Chile, Avda Blanco Encalada 2120, 3er piso, Santiago, Chile; email: pbarcelo@dcc.uchile.cl; L. Libkin, School of Informatics, University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom; email: libkin@ed.ac.uk; J. L. Reutter, Department of Computer Science, Pontificia Universidad Católica de Chile, Vicuna Mackenna 4860, Macul, Santiago, Chile; email: jreutter@ing.puc.cl.

DOI: http://dx.doi.org/10.1145/2559905

Gutierrez 2009], and the semantic Web [Gutierrez et al. 2011; Pérez et al. 2009]. In such applications, the underlying data is naturally modeled as graphs, in which nodes are objects, and edge labels define relationships between those objects [Angles and Gutierrez 2008].

A standard way of querying graph data is to look for reachability patterns. Such patterns specify that paths satisfying certain conditions should exist between nodes. Initially proposed in a simple form in Cruz et al. [1987] and Consens and Mendelzon [1990], pattern languages have been developed over time and used in a variety of applications, such as biology, studying network traffic, crime detection, modeling object-oriented data, querying and searching RDF data, etc. [Fan et al. 2010a, 2010b; Gutierrez et al. 2011; Gyssens et al. 1994; Leser 2005; Milo et al. 2002; Natarajan 2000; Pérez et al. 2009; Ronen and Shmueli 2009; SanMartín and Gutierrez 2009; Tong et al. 2007; Weikum et al. 2009]; see also the survey [Angles and Gutierrez 2008]. In their simplest form, patterns are just graphs, whose occurrences in large graphs are of interest. Already in this simple form, they are very important in biological applications, where search for network motifs [Milo et al. 2002] is a common task. But for applications such as, for example, crime detection or RDF data, more complex patterns are needed, as one can look for connections between elements in a network that involve complex paths via some intermediaries.

The notions of finding matches for complex patterns also evolved with time, from traditional NP-complete subgraph isomorphism (used, nonetheless, in practical applications, for example, in Cheng et al. [2008] and Tong et al. [2007]) to notions based on graph homeomorphisms (i.e., mapping edges to paths) and simulation relations between patterns and graphs [Buneman et al. 1996; Fan et al. 2010a, 2010b]. Outputs of matching queries are patterns themselves: their nodes are those that are involved in the simulation relation, and relationships between them are those specified in the pattern. For example, in a crime detection scenario, a query may output a set of individuals who might be involved in a crime network, together with descriptions of paths specifying their relationships. Similar scenarios arise in querying semistructured data as well, where it is sometimes natural to output incomplete query results [Kanza et al. 2002]. When such matching and query results require extracting additional information from them, one ends up querying patterns rather than graphs.

There are other scenarios where the need for querying patterns naturally arises. A pattern represents partial information about graph-structured data. Querying partial information is commonly present in integrating and exchanging (or translating) data [Arenas et al. 2010; Fagin et al. 2005; Lenzerini 2002]. In such applications, one queries the result of applying some schema mapping rules to source data, which yields a partially specified database. Partial databases – whether relational or XML – are typically viewed as patterns [Barceló et al. 2010b; Björklund et al. 2007; Imielinski and Lipski 1984]. For graph data, the study of schema mappings and transformations for data exchange an integration has started recently [Calvanese et al. 2011; San Martín and Gutierrez 2009], but techniques for querying resulting partially specified graphs are currently lacking.

Motivated by these considerations, we study querying partially defined graph data, that is, graph patterns. As for other data models [Arenas et al. 2010; Barceló et al. 2010b; Fagin et al. 2005; Imielinski and Lipski 1984; Lenzerini 2002], one is looking for answers that are independent of the way in which the missing parts of patterns are interpreted, that is, certain answers.

Based on the examples arising in querying and transforming graph data, we now analyze types of features that need to be addressed in the study of querying graph patterns. Recall that in the relational case, one deals with variables in place of missing data values [Imielinski and Lipski 1984]. In the case of XML, one may also have

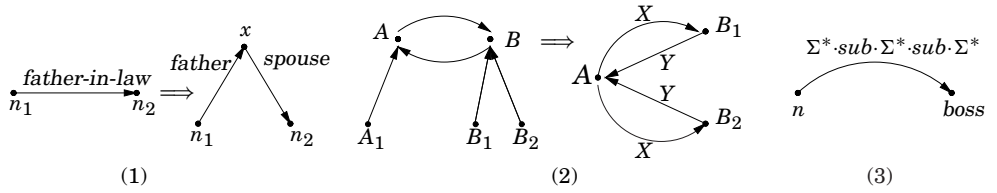


Fig. 1. Examples of (1) node variables, (2) label variables, and (3) regular expressions.

missing structural information [Barceló et al. 2010b]. For graph databases, partiality of specifications mainly arises in the following three ways.

Node Variables. Similarly to values missing in relational or XML data, identities of some nodes can be missing in graph data. For example, in transforming a social network that has different types of relationship edges, we can split an edge *(Name1, father-in-law, Name2)* into two edges *(Name1, father, x)* and *(x, spouse, Name2)*, with an unknown identity x. This is illustrated in Figure 1(a). Variables can also be used to model blank nodes in RDF [Pérez et al. 2009].

Label Variables. We may also miss the precise relationships between nodes. But even if we do not know them, we may still know that some of the relationships are the same. Taking an example from social networks, consider transforming a network where we have two "celebrities" A and B who have "followers" $A_1, \ldots, A_n$ and $B_1, \ldots, B_m$ (like on the Twitter network). Suppose we know the relationship between A and B (e.g., they like, or dislike each other). We may wish to record this as a relationship between their followers: for instance, if A hates B and A_i follows A, we may deduce something about how A_i relates to B. At the time of transforming a network, we may not know the exact nature of such a relationship, but we know there exists one, and it should be the same for all the followers of A. Likewise, all the followers of B will be in some relationship with A (but not necessarily the same as the followers of A with B). So we add edges

$$(A_1, X, B), \ldots, (A_n, X, B),\ (B_1, Y, A), \ldots, (B_m, Y, A)$$

where X and Y are edge labels: we do not yet know what the relationship will be, but want to record that it is the same among all the followers. This is illustrated in Figure 1(b).

Regular Languages. Returning to the example with crime detection in a network of people, the result of a matching may contain facts like "there is a path between x and the boss that goes via at least two intermediaries", which will be expressed by a regular expression $\Sigma^* \cdot sub \cdot \Sigma^* \cdot sub \cdot \Sigma^*$, where *sub* indicates subordinacy in the hierarchy, and Σ is the set of all labels. This is illustrated in Figure 1(c). In general, the situation where only regular paths between nodes can be deduced from a matching is very common [Fan et al. 2011]. Thus, when we do not have an exact path between two nodes, we attempt to replace it by an edge (A, e, B), where e is a regular expression.

These are the key features that we add to patterns. Note that replacing known data by variables is common to all models of incomplete information. One new element that is specific to graph patterns is adding regular expressions to label edges. Thus, essentially we look at patterns whose key features are captured by variables and regular languages.

Once we have these features added to patterns, we need to define a query language for them. Most commonly used query languages for graph databases specify the existence of paths between nodes, with the restriction that the labels of such path belong

to regular languages [Abiteboul et al. 1999; Calvanese et al. 2002; Consens and Mendelzon 1990; Cruz et al. 1987; Gyssens et al. 1994]. The simplest such queries are known as *regular path queries*, or RPQs [Cruz et al. 1987]; those select nodes connected by a path that belongs to a regular language. *Conjunctive* RPQs, or CRPQs, extend them by allowing intermediate nodes in paths. Dealing with incomplete data, we often have *duality* between data and queries. For example, relational naive tables are tableaux of conjunctive queries, and in XML, typical query languages are based on tree patterns, that is, incomplete descriptions of documents. We shall see that queries such as RPQs and CRPQs arise as special cases of graph patterns, continuing the analogy with the well studied cases.

To sum up, our main goal is to define classes of regular graph patterns, study their properties, and query answering over them. Our main contributions are as follows.

(1) We define classes of graph patterns that have the key features listed previously – node variables, label variables, and edges labeled with regular expressions – and provide a complete classification of their expressiveness.
(2) We study the complexity of query answering (i.e., the problem of finding certain answers to queries over graph patterns). We fully analyze it for CRPQs, both for data complexity (which ranges from NLOGSPACE to coNP) and for combined complexity (which ranges from NP to EXPSPACE). For classes of high complexity, we do an in-depth analysis, showing which features lead to intractability. We also show that upper bounds for CRPQs extend to more expressive queries.
(3) We provide an automaton model for query answering. Specifically, we define a class of automata, called *incomplete automata*, that naturally give rise to two acceptance notions that precisely capture certain answers: one of them corresponds to queries that return nodes, and the other to queries that return paths. In the latter case, answers to queries are represented by NFAs. We analyze the complexity of incomplete automata, and prove lower bounds on the sizes of NFAs representing query answers.
(4) Returning to the intractable cases for query answering, we look at two ways of reducing complexity: by imposing structural restrictions, and by reducing to problems for which many efficient heuristics are known. Along these lines, we prove that for several classes of graph patterns, the bounded treewidth restriction guarantees tractability. We also show how to cast finding certain answers as a constraint satisfaction problem, which allows us to use algorithmic techniques from that field.

Remarks. Graph Patterns in Other Areas. The motivations of this article come from dealing with graph databases, as is reflected in the word "querying" in the title. While the models based on variables and regular expressions that we use, and particular results we show, are specifically tailored to handling patterns as a model of incompleteness in graph databases, this is not the only possible application area of graph patterns. Indeed, a graph pattern π is just a compact representation of a (potentially infinite) set of graphs. As such, querying them can be seen as solving the validity problem. Indeed, suppose ϕ_π is a formula, in some logical formalism, describing the set of graphs given by π. If a query ψ is issued over the pattern, then evaluating it amounts to checking *validity* of the implication $\phi_\pi \rightarrow \psi$, saying that every graph represented by π satisfies ψ.

Logical formalisms capable of describing infinite families of graphs and having decidable validity problem have appeared in other areas, notably verification and description logics. In verification, the standard formalisms typically do not distinguish graphs up to bisimulation. However, up to bisimulation, several of them, for instance, the

μ-calculus, can describe arbitrary finite graphs, as well as regular properties of paths. Validity problem for the μ-calculus and several of its extension that add power in a way relevant for potential applications in graph databases [Bonatti et al. 2008] are EXPTIME-complete; with our formalisms the complexity generally jumps one exponent in most fragments. A slightly different take on incompleteness is present in the work on module checking [Kupferman et al. 2001] which introduces a form of open-world assumption, similar to the semantics that we are using. While such results and those in our paper are completely independent, it is worth mentioning them as another way of specifying sets of graphs and solving the validity problem over them.

Description logics is yet another area where satisfiability and validity are central problems, and reasonable-complexity algorithms are of great interest. Some of the formalism come very close to the μ-calculus and can describe finite graphs as well as some of the features of patterns. For instance, De Giacomo and Lenzerini [1997] gives a description logic that is extended with fixed points, thus gaining a lot of expressivity while remaining decidable, also in exponential time.

Thus, describing sets of graphs by means of incomplete descriptions and solving the validity problem for formulas over such sets is by no means unique to the area of graph databases; but specific models considered here are, as well as results about them that we show.

Organization. In Section 2, we define graph databases and queries over them. In Section 3, we define graph patterns and, in Section 4, we study their classifications and structural properties. In Section 5, we analyze both data and combined complexity of query answering. In Section 6, we deal with incomplete automata, and relate them to answering queries over graph patterns. In Section 7, we look at tractability restrictions and reduction to constraint satisfaction.

2. GRAPH DATABASES, RPQS, AND CRPQS

Graph Databases. A *graph database* [Angles and Gutierrez 2008; Calvanese et al. 2002; Cruz et al. 1987] is just a finite edge-labeled graph. Let Σ be a finite alphabet, and $\mathcal{N}$ a countably infinite set of node ids. Then, a graph database over Σ is a pair $G = (N, E)$, where N is the set of nodes (a finite subset of $\mathcal{N}$), and E is the set of edges, that is, $E \subseteq N \times \Sigma \times N$. That is, we view each edge as a triple (n, a, n'), whose interpretation, of course, is an a-labeled edge from n to n'. When Σ is clear from the context, we shall simply speak of a graph database.

A *path* ρ from n_0 to n_m in G is a sequence $(n_0, a_0, n_1), (n_1, a_1, n_2), \ldots, (n_{m-1}, a_{m-1}, n_m)$, for some $m \geq 0$, where each (n_i, a_i, n_{i+1}), for $i < m$, is an edge in E. In particular, all the n_i's are nodes in N and all the a_j's are letters in Σ. The *label* of ρ, denoted by $\lambda(\rho)$, is the word $a_0 \cdots a_{m-1} \in \Sigma^*$. We also define the empty path as (n, ϵ, n) for each $n \in N$; the label of such path is the empty word ϵ.

Regular Path Queries. The basic querying mechanism for graph databases is provided by means of *regular path queries*, or *RPQs* [Abiteboul et al. 1999; Calvanese et al. 2002; Cruz et al. 1987]. They retrieve pairs of nodes in a graph database connected by a path whose label belongs to a given regular language. Formally, an RPQ Q is an expression of the form (x, L, y) where $L \subseteq \Sigma^*$ is a regular language. We shall assume that syntactically L is given as a regular expression. Given a graph database $G = (N, E)$ and an RPQ Q, both over Σ, the answer $Q(G)$, is the set of all pairs $(n, n') \in N$ such that there is path ρ between them whose label $\lambda(\rho)$ is in L.

It has been argued (see, e.g., Abiteboul et al. [1999], Cruz et al. [1987], Consens and Mendelzon [1990], and Calvanese et al. [2000b]) that analogs of conjunctive queries whose atoms are RPQs are much more useful in practice than simple RPQs. In such queries, multiple RPQs can be combined, and some variables can be existentially

quantified. Formally, a *conjunctive regular path query*, or *CRPQ* Q over a finite alphabet Σ is an expression of the form:

$$Ans(\bar{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, L_i, y_i), \tag{1}$$

such that $m > 0$, each (x_i, L_i, y_i) is an RPQ, and $\bar{z}$ is a tuple of variables among $\bar{x}$ and $\bar{y}$. The atom $Ans(\bar{z})$ is the *head* of the query, the expression on the right of the $\leftarrow$ is its *body*. A query with the head $Ans()$ (i.e., no variables in the output) is called a *Boolean* query.

Intuitively, such a query Q selects tuples $\bar{z}$ for which there exist values of the remaining node variables from $\bar{x}$ and $\bar{y}$ such that each RPQ in the body is satisfied. Formally, given Q of the form (1) and a graph $G = (N, E)$, a valuation is a map $\sigma : \bigcup_{1 \leq i \leq m} \{x_i, y_i\} \to N$. We write $(G, \sigma) \models Q$ if $(\sigma(x_i), \sigma(y_i))$ is in the answer to RPQ $(x_i.L_i, y_i)$ in G, that is, if there is a path ρ_i in G from $\sigma(x_i)$ to $\sigma(y_i)$ with $\lambda(\rho_i) \in L_i$. Then, $Q(G)$ is the set of all tuples $\sigma(\bar{z})$ such that $(G, \sigma) \models Q$. If Q is Boolean, we let $Q(G)$ be `true` if $(G, \sigma) \models Q$ for some σ (i.e., as usual, the singleton set with the empty tuple models `true`, and the empty set models `false`).

In what follows, we also adopt a view of RPQs as potentially having some variables existentially quantified. That is, RPQs will be of the form $Ans(\bar{z}) \leftarrow (x, L, y)$, where $\bar{z}$ contains variables from $\{x, y\}$. For example, $Ans() \leftarrow (x, L, y)$ is a Boolean RPQ checking whether there is a path whose label is in L.

3. GRAPH PATTERNS

As in the case of tree-structured data, for example, XML, where the ability to find binding of variables that match a tree pattern is crucial for the basic querying mechanisms [Lakshmanan et al. 2004], our goal in this section is to define a class of graph patterns that can be considered the core of each query language that provides enough expressive power to express relevant graph properties [Abiteboul et al. 1999].

As explained in the introduction, the key new features of graph patterns are the ability to use the following (in addition to nodes and edge labels of graph databases):

— node variables, that is, marked nulls for graph nodes;
— label variables, that is, marked nulls for edge labels;
— regular expressions as labels for edges.

Thus, we shall define graph patterns as graph databases over constant nodes and node variables, whose edges will be labeled with regular expressions that may use label variables. To do this, we shall use the following (countably infinite) sets:

— $\mathcal{V}_{\text{node}}$ of *node variables* (normally denoted by lower-case letters), and
— $\mathcal{V}_{\text{lab}}$ of *label variables* (normally denoted by upper-case letters).

If Γ is an arbitrary (finite or infinite) set of symbols, we write $\text{REG}(\Gamma)$ to denote the set of nonempty regular languages over Γ (if Γ is infinite, then each $L \in \text{REG}(\Gamma)$ only uses finitely many symbols from Γ). Recall that a graph database over a labeling alphabet Σ was defined as a labeled graph, (N, E), where $N \subseteq \mathcal{N}$ is the set of nodes and $E \subseteq N \times \Sigma \times N$ is the set of labeled edges. We are now in a position to define graph patterns formally.

Definition 3.1 (*Graph Patterns*). A *graph pattern* over finite alphabet Σ is a pair $\pi = (N, E)$ where

— $N \subseteq \mathcal{N} \cup \mathcal{V}_{\text{node}}$ is the finite set of nodes, and
— $E \subseteq N \times \text{REG}(\Sigma \cup \mathcal{V}_{\text{lab}}) \times N$ is the set of edges.

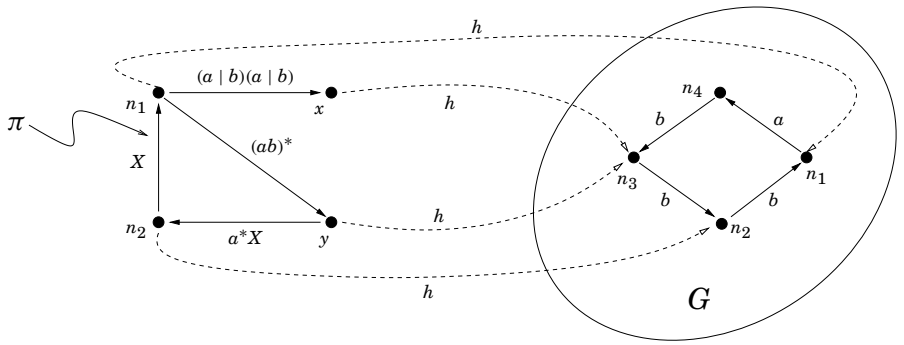


Fig. 2. A homomorphism $h : \pi \to G$.

Semantics. In complete analogy with relational naive tables or incomplete XML documents, the semantics is defined via homomorphisms. To define those, we need extensions of partial functions $f : \Gamma \to \Gamma$ to languages $L \in \mathrm{REG}(\Gamma)$ defined as $f(L) = \{f(w) \mid w \in L\}$, where $f(w)$ is obtained by replacing each symbol a of a word w on which f is defined by $f(a)$, and leaving symbols b on which f is not defined intact.

Since variables can occur at the level of both nodes and edge labels, homomorphisms will be in fact *pairs* of mappings. Given a graph database $G = (N, E)$ and a pattern $\pi = (N', E')$, a *homomorphism* $h : \pi \to G$ is a pair $h = (h_1, h_2)$ of mappings $h_1 : N' \to N$ and h_2 that maps label variables used in π to labels used in G such that:

(1) $h_1(n) = n$ for every node id $n \in \mathcal{N}$; and
(2) for every edge $(p, L, p') \in E'$, there is path between $h_1(p)$ and $h_1(p')$ in G whose label is in $h_2(L)$.

We now write $G \models \pi$ if there is a homomorphism $h : \pi \to G$. The semantics is defined with respect to a labeling alphabet Σ:

$$[\![\pi]\!]_\Sigma \;=\; \{G \text{ over } \Sigma \mid G \models \pi\}.$$

Most often Σ is clear from the context and we write simply $[\![\pi]\!]$ then.

Example 3.2. An illustration is given in Figure 2: a homomorphism is defined by letting label variable X be b, and by mapping both node variables x and y into n_3. The edge $(n_1, (a|b)(a|b), x)$ is then mapped into the *path* $(n_1, a, n_4), (n_4, b, n_3)$ with label ab. The edge $(n_1, (ab)^*, y)$ is mapped into the same path, since ab belongs to regular languages denoted by both $(a|b)(a|b)$ and $(ab)^*$. The edge (y, a^*X, n_2) is mapped into (n_3, b, n_2), since b is in the language denoted by a^*b.

Certain Answers. Consider queries Q that take graph databases as input and return sets of tuples of their nodes. For example, RPQs and CRPQs are such queries. For them, we can define their certain answers on graph patterns in the standard way:

$$\mathrm{CERTAIN}_\Sigma(Q, \pi) \;=\; \bigcap \{Q(G) \mid G \in [\![\pi]\!]_\Sigma\}.$$

Again, if Σ is clear from the context, we write simply $\mathrm{CERTAIN}(Q, \pi)$.

Example 3.3. The labeling alphabet can make a difference in finding certain answers. Consider a pattern with edges $(n_1, a, n_2), (n_2, X, n_3), (n_3, b, n_4)$, where X is a label variable. Let Q be the Boolean RPQ $Ans() \leftarrow (x, ab, y)$. Then $\mathrm{CERTAIN}_{\{a,b\}}(Q, \pi) =$ `true`: whether X is a or b, there is a path labeled ab. However, $\mathrm{CERTAIN}_{\{a,b,c\}}(Q, \pi) =$ `false` (by setting $X = c$).

Graph Patterns as Queries. Graph patterns can naturally be viewed as queries – again in complete analogy with relational databases (where naive tables are a natural representation of conjunctive queries, i.e., tableaux) and XML documents (where tree patterns form the basis of tree conjunctive queries [Björklund et al. 2007; Gottlob et al. 2006]). This view has also been explored in Cohen and Sagiv [2005].

We adopt the convention that patterns used as queries are denoted by ξ, and patterns used as data are denoted by π. A *graph query* is a pair $Q = (\xi, \bar{x})$, where $\xi = (N, E)$ is a graph pattern, and $\bar{x}$ is a tuple of elements from N. For example, a CRPQ $Ans(\bar{z}) \leftarrow \bigwedge_{i \leq m}(x_i, L_i, y_i)$, can be viewed as a graph query $(\xi, \bar{z})$, where ξ simply contains the edges (x_i, L_i, y_i) for $i \leq m$.

We now define the semantics of a graph query on graph databases (later, we shall extend it to graph patterns). Given a graph database $G = (N, E)$ with $N \subset \mathcal{N}$, and a graph query $Q = (\xi, \bar{x})$ with $|\bar{x}| = k$, the answer to Q on G is:

$$Q(G) \;=\; \{\bar{v} \in N^k \mid G \models \xi[\bar{v}/\bar{x}]\}.$$

Here, $\xi[\bar{v}/\bar{x}]$ is the result of substituting $\bar{v}$ for $\bar{x}$ in the pattern ξ.

It is easy to see that when Q is a CRPQ viewed as a graph query, the result $Q(G)$ coincides with the standard semantics of CRPQs.

Example 3.4. Consider again the example in Figure 2 and the homomorphism described in Example 3.2. Let ξ be the pattern obtained from π by changing X to b, and replacing n_1 and n_2 with variables z_1 and z_2. The resulting pattern can be viewed as a CRPQ (ξ, x, y):

$$\begin{aligned} Ans(x,y) \;\leftarrow\; & (z_1, (a|b)(a|b), x),\ (z_1, (ab)^*, y), \\ & (y, a^*b, z_2),\ (z_2, b, z_1). \end{aligned}$$

If it is evaluated in graph G shown in Figure 2, one tuple in the output will be (n_3, n_3), since $G \models \xi[n_3/x, n_3/y]$, as witnessed by homomorphism h shown in the figure.

4. CLASSIFICATION AND BASIC PROPERTIES

The three key features of graph patterns – node variables, label variables, and regular expressions – provide a natural classification of patterns. We shall refer to classes of patterns as $\mathcal{P}^\sigma$, where σ enumerates the present features. We use "nv" for node variables, "lv" for label variables, and "re" for regular expressions. This gives us 8 classes, from $\mathcal{P}$ (none of the features is present) to $\mathcal{P}^{\text{nv,lv,re}}$ (all are present).

Of course $\mathcal{P}$ is the class of graph databases (N, E) with $N \subseteq \mathcal{N}$ and $E \subseteq N \times \Sigma \times N$, and $\mathcal{P}^{\text{nv,lv,re}}$ is the class of all graph patterns as in Definition 3.1 with $N \subseteq \mathcal{N} \cup \mathcal{V}_{\text{node}}$ and $E \subseteq N \times \text{REG}(\Sigma \cup \mathcal{V}_{\text{lab}}) \times N$. We now examine some others.

— $\mathcal{P}^{\text{nv}}$ is the class of graphs where nodes could be either constants, or node variables; all edges are labeled with alphabet letters, that is, $N \subseteq \mathcal{N} \cup \mathcal{V}_{\text{node}}$ and $E \subseteq N \times \Sigma \times N$. These patterns can be represented by relational naive tables.

— $\mathcal{P}^{\text{nv,re}}$ is the class of patterns where nodes could be either constants or node variables, and edges are labeled with regular expressions over Σ. That is, $N \subseteq \mathcal{N} \cup \mathcal{V}_{\text{node}}$ and $E \subseteq N \times \text{REG}(\Sigma) \times N$.

These are essentially CRPQs, which are graph queries $(\xi, \bar{x})$ where ξ is from $\mathcal{P}^{\text{nv,re}}$ and uses only node variables (without this restriction, we have the class of CRPQs that can mention constants).

— $\mathcal{P}^{\text{nv,lv}}$ is the class of patterns where nodes could be either constants or node variables, and edges are labeled with letters or variables. That is, $N \subseteq \mathcal{N} \cup \mathcal{V}_{\text{node}}$ and $E \subseteq N \times (\Sigma \cup \mathcal{V}_{\text{lab}}) \times N$. The class $\mathcal{P}^{\text{lv}}$ is its restriction when $N \subseteq \mathcal{N}$.

Since patterns from $\mathcal{P}^{\text{nv}}$ can be represented by relational naive tables, this suggests that naive query evaluation [Imielinski and Lipski 1984] will work for them, and we shall see that this is indeed true. However, this will turn out to be the largest class for which such naive evaluation works.

Given multiple features of graph patterns, it is natural to ask whether all are necessary, or some are expressible with others. We now show that all three are essential.

- We write $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$ if $\mathcal{P}^{\sigma'}$ is at least as expressive as $\mathcal{P}^{\sigma}$. That is, for every pattern $\pi \in \mathcal{P}^{\sigma}$, there is a pattern $\pi' \in \mathcal{P}^{\sigma'}$ so that $[\![\pi]\!] = [\![\pi']\!]$ (i.e., $[\![\pi]\!]_{\Sigma} = [\![\pi']\!]_{\Sigma}$ for each Σ containing the labels used in π).
- We write $\mathcal{P}^{\sigma} \sim \mathcal{P}^{\sigma'}$ if $\mathcal{P}^{\sigma}$ and $\mathcal{P}^{\sigma'}$ are equally expressive (i.e., $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$ and $\mathcal{P}^{\sigma'} \preceq \mathcal{P}^{\sigma}$).
- Finally, $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\sigma'}$ means that $\mathcal{P}^{\sigma'}$ is strictly more expressive than $\mathcal{P}^{\sigma}$: that is, $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$, but they are not equally expressive.

THEOREM 4.1. *Adding each new feature to graph patterns strictly increases their expressiveness: in other words, $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\sigma'}$ if and only if $\sigma \subsetneq \sigma'$, and $\mathcal{P}^{\sigma} \sim \mathcal{P}^{\sigma'}$ if and only if $\sigma = \sigma'$.*

PROOF. In order to prove the first part of the theorem, we make use of the following lemma:

LEMMA 4.2. *The following holds:*

(1) *There exists a pattern π in $\mathcal{P}^{\text{nv}}$ over alphabet $\Sigma = \{a\}$, such that $[\![\pi]\!]_{\Sigma} \neq [\![\pi']\!]_{\Sigma}$ for all patterns π' in $\mathcal{P}^{\text{lv,re}}$ over the same alphabet.*
(2) *There exists a pattern π in $\mathcal{P}^{\text{re}}$ over alphabet $\Sigma = \{a\}$, such that $[\![\pi]\!]_{\Sigma} \neq [\![\pi']\!]_{\Sigma}$ for all patterns π' in $\mathcal{P}^{\text{nv,lv}}$ over the same alphabet.*
(3) *There exists a pattern π in $\mathcal{P}^{\text{lv}}$ over alphabet $\Sigma = \{a, b\}$, such that $[\![\pi]\!]_{\Sigma} \neq [\![\pi']\!]_{\Sigma}$ for all patterns π' in $\mathcal{P}^{\text{nv,re}}$ over the same alphabet.*

Indeed, we show next, using Lemma 4.2, that $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\sigma'}$ if and only if $\sigma \subsetneq \sigma'$.

($\Leftarrow$): From the definition, it is clear that $\sigma \subsetneq \sigma'$ implies $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$. Assume for the sake of contradiction that $\sigma \subsetneq \sigma'$, but $\mathcal{P}^{\sigma} \sim \mathcal{P}^{\sigma'}$. Since $\sigma \subsetneq \sigma'$, there is an element of $\{\text{nv}, \text{lv}, \text{re}\}$ that belongs to σ', but not to σ. It follows from statements (4.2, 4.2, and 4.2) of Lemma 4.2 that there is a pattern π' in $\mathcal{P}^{\sigma'}$ over some alphabet Σ, such that $[\![\pi']\!]_{\Sigma} \neq [\![\pi]\!]_{\Sigma}$, for all patterns $\pi \in \mathcal{P}^{\sigma}$ over Σ. This is a contradiction.

($\Rightarrow$): To prove that $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\sigma'}$ implies $\sigma \subsetneq \sigma'$, assume for the sake of contradiction that for some σ, σ' it is the case that $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\sigma'}$, but it is not the case that $\sigma \subsetneq \sigma'$. Then, the only possibility is that $\sigma \not\subseteq \sigma'$. (Indeed, if $\sigma = \sigma'$ then clearly $\mathcal{P}^{\sigma'} \preceq \mathcal{P}^{\sigma}$, which is a contradiction). Then, there exists an element of $\{\text{nv}, \text{lv}, \text{re}\}$ that belongs to σ but not to σ'. It follows again from statements (4.2, 4.2, and 4.2) in Lemma 4.2 that it is not the the case that $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$, which is a contradiction.

Thus, in order to prove the first part of Theorem 4.1, we only have to prove Lemma 4.2. This is what we do next.

PROOF OF LEMMA 4.2. We begin by proving statement (4.2). Consider a pattern $\pi = (N, E)$ over alphabet $\Sigma = \{a\}$, where N consists of the node variables x and y, and

E consists of the edge (x, a, y). Clearly, π belongs to $\mathcal{P}^{\text{nv}}$. We now prove that there is no pattern π' in $\mathcal{P}^{\text{lv,re}}$ such that $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$. The idea is as follows. First, notice that the set $\bigcap\{N_G \mid G = (N_G, E_G) \text{ and } G \in [\![\pi]\!]_\Sigma\}$ containing the node id's that appear in all graphs in $[\![\pi]\!]_\Sigma$ is equal to the empty set (this can be easily proved using the fact that we only enforce homomorphisms to be the identity on constants). Second, it is easy to see that no pattern without edges over Σ can represent exactly the graphs in $[\![\pi]\!]_\Sigma$, since all graphs in $[\![\pi]\!]_\Sigma$ must have at least one edge. Thus, all that we need to prove is that no pattern π' in $\mathcal{P}^{\text{lv,re}}$ over Σ, with at least one edge, satisfies the following: $\bigcap\{N_G \mid G = (N_G, E_G) \text{ and } G \in [\![\pi']\!]_\Sigma\} \neq \emptyset$. That is, all the graphs in $[\![\pi']\!]_\Sigma$ must have at least one node in common. But this is quite obvious since every pattern π' in $\mathcal{P}^{\text{lv,re}}$, with at least one edge, contains at least one constant, and such a constant must belong to every graph G in $[\![\pi]\!]_\Sigma$.

Now we prove statement (4.2); namely, that there exists a pattern π in $\mathcal{P}^{\text{re}}$ over alphabet $\Sigma = \{a\}$, such that there is no pattern π' in $\mathcal{P}^{\text{nv,lv}}$ over the same alphabet that satisfies $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$. Define $\pi = (N, E)$ over alphabet $\{a\}$ as follows: The set N of nodes consists of node ids $\{n_1, n_2\}$, and E consists of the edge (n_1, aa^*, n_2).

Assume, for the sake of contradiction, that there is a pattern $\pi' \in \mathcal{P}^{\text{nv,lv}}$ over Σ, such that $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$. It is clear then that the only node ids that appear in pattern π' are n_1 and n_2. We distinguish two cases, depending on the structure of π'.

— The node n_2 is not reachable from node n_1 in π'. It is then easy to construct a graph $G \in [\![\pi']\!]_\Sigma$ such that n_2 is not reachable from n_1: It suffices to replace every node variable in π' to a fresh node constant, and every label variable with the symbol a. This is a contradiction, since every graph in $[\![\pi]\!]_\Sigma$ must satisfy that nodes n_1 and n_2 are in the same connected component.

— Node n_2 is reachable from n_1 in π'. Let $\rho \geq 0$ be the longest simple path between n_1 and n_2 in π'. We prove here the following property, which immediately yields to a contradiction: For every graph $G \in [\![\pi']\!]_\Sigma$, there is a path in G from n_1 to n_2, and the length of the shortest such path is at most $|\rho|$. On the other hand, it is easy to construct a graph in $[\![\pi]\!]_\Sigma$ such that n_1 and n_2 are not connected by any path of size ρ or less. This is a contradiction.

  Clearly, every graph $G \in [\![\pi']\!]_\Sigma$ contains a path from n_1 to n_2, since these node ids are in the same connected component of π'. Assume now, for the sake of contradiction, that there is a graph $G \in [\![\pi']\!]_\Sigma$ such that G has no path of size $\leq |\rho|$ from n_1 to n_2. Furthermore, assume that ρ in π' is of form $n_1, x_1, \ldots, x_{|\rho|-1}, n_2$, where each x_i, $1 \leq i \leq |\rho| - 1$, is a node variable. Since $G \in [\![\pi']\!]_\Sigma$, there is a homomorphism $h = (h_1, h_2)$ from π' to G. Further, $h(n_1)$ and $h(x_1))$ must be connected in G with a path of size 1, and the same is true for ($h(x_{|\rho|-1}$ and $h(n_2))$ and for $h(x_i)$ and $h(x_{i+1}))$, for each $1 \leq i \leq |\rho| - 2$. (Indeed, since $\pi' \in \mathcal{P}^{\text{nv,lv}}$, the regular expressions in the edges of π' can only be label variables or letters from the alphabet). We have just constructed a path from n_1 to n_2 in G of size at most $|\rho|$. This proves the claim.

This concludes the proof of the second statement of the lemma. □

For statement (4.2), we prove that there exists a pattern π in $\mathcal{P}^{\text{lv}}$ over alphabet $\Sigma = \{a, b\}$, such that $[\![\pi]\!]_\Sigma \neq [\![\pi']\!]_\Sigma$ for all patterns π' in $\mathcal{P}^{\text{nv,re}}$ over Σ. We use the following claim.

CLAIM 1. *Let π be a pattern in $\mathcal{P}^{\text{nv,re}}$ over alphabet $\{a, b\}$ such that the nodes n_1, n_2, n_3, n_4 are the only node ids of π, and assume that the graph databases G and G' belong to $[\![\pi]\!]_\Sigma$, where G consists of edges $e_{12} = (n_1, a, n_2)$ and $e_{34} = (n_3, a, n_4)$, and G'*

consists of edges $e'_{12} = (n_1, b, n_2)$ *and* $e'_{34} = (n_3, b, n_4)$. *Then, the graph* G'' *that consists of edges* e_{12} *and* e'_{34} *also belongs to* $[\![\pi]\!]_\Sigma$.

PROOF. Let $h = (h_1, h_2)$ and $h' = (h'_1, h'_2)$ be homomorphisms from π into G and G', respectively. Notice that since π belongs to $\mathcal{P}^{\text{nv,re}}$, we are only interested in the mappings h_1 and h'_1 that map nodes of π into nodes of G.

Define, from h_1, a mapping h''_1 from the nodes of π into the nodes of G'' as follows:

— $h''_1(n) = n$, if n is a node id;
— $h''_1(x) = n_1$, if $h_1(x) = h'_1(x) = n_1$;
— $h''_1(x) = n_2$, if $h_1(x) = h'_1(x) = n_2$;
— $h_1(x) = n_3$ if $h_1(x) = n_3$ or $h'_1(x) = n_3$; and
— $h_1(x) = n_4$ if $h_1(x) = n_4$ or $h'_1(x) = n_4$.
— $h_1(x) = n_1$ otherwise.

We claim that h''_1 is a homomorphism from π into G''. It is clear that h''_1 maps nods of π into nodes of G'' and it is the identity on constants. Thus, we only need to prove that for every edge of form (p, R, q) in π, there exists a path in G'' from $h''_1(p)$ into $h''_1(q)$ that is labeled with a word from R.

Let $e = (p, R, q)$ be an arbitrary edge of π. Notice that, since h_1 and h'_1 are homomorphisms, the fact that $h_1(p) = n_1$ implies that $h_1(q) = n_2$, and $h_1(p) = n_3$ implies $h_1(q) = n_4$. This is due to the properties of homomorphisms and the fact that the only edge in G starting from n_1 is (n_1, a, n_2), and the only edge in G starting with n_3 is (n_3, a, n_4). Same argument holds for the case of h'_1, namely that $h'_1(p) = n_1$ implies that $h'_1(q) = n_2$, and $h'_1(p) = n_3$ implies $h'_1(q) = n_4$. We consider all possible cases, depending on the values of $h_1(p)$ and $h'_1(p)$.

— Suppose first that $h_1(p) = h'_1(p) = n_1$. Then, as we mentioned previously, it must be the case that $h_1(q) = h'_1(q) = n_2$, and thus $h''_1(p) = n_1$ and $h''_1(q) = n_2$. Since h_1 is a homomorphism from π to G, there must be a path from $h_1(p)$ to $h_1(q)$ in G labeled with a word in $L(R)$; it follows that a belongs to $L(R)$. Then, it is clear that there is path in G'' from $h''_1(p)$ to $h''_1(q)$ that is labeled with a word in $L(R)$ (namely, the word a).
— Suppose that $h_1(p) = n_1$, but $h'_1(p) = n_3$. Then, we have that $h_1(q) = n_2$ and $h'_1(q) = n_4$, and thus $h''_1(p) = n_3$, $h''_1(q) = n_4$. Since h'_1 is a homomorphism, there must be a path from $h'_1(p)$ to $h'_1(q)$ in G' labeled with a word in $L(R)$; it follows that b belongs to $L(R)$. Then, it is clear that there is path in G'' from $h''_1(p)$ to $h''_1(q)$ that is labeled with a word in $L(R)$ (namely, the word b).
— Suppose that $h_1(p) = n_3$, but $h'_1(p) = n_1$. Then, we have that $h_1(q) = n_4$ and $h'_1(q) = n_2$, and thus $h''_1(p) = n_3$ and $h''_1(q) = n_4$. Since h'_1 is a homomorphism, there must be a path from $h'_1(p)$ to $h'_1(q)$ in G' labeled with a word in $L(R)$; it follows that b belongs to $L(R)$. Then, it is clear that there is path in G'' from $h''_1(p)$ to $h''_1(q)$ that is labeled with a word in $L(R)$ (namely, the word b).
— Suppose that $h_1(p) = h'_1(p) = n_3$. Then, $h_1(q) = h'_1(q) = n_4$, and thus $h''_1(p) = n_3$ and $h''_1(q) = n_4$. Since h'_1 is a homomorphism, there must be a path from $h'_1(p)$ to $h'_1(q)$ in G' labeled with a word in $L(R)$; it follows that b belongs to $L(R)$. Then, it is clear that there is path in G'' from $h''_1(p)$ to $h''_1(q)$ that is labeled with a word in $L(R)$ (namely, the word b).
— Suppose that $h_1(p) \notin \{n_1, n_3\}$. This is not possible due to the fact that h_1 is a homomorphism from π to G, and there are no edges in G that start from nodes n_2 or n_4.

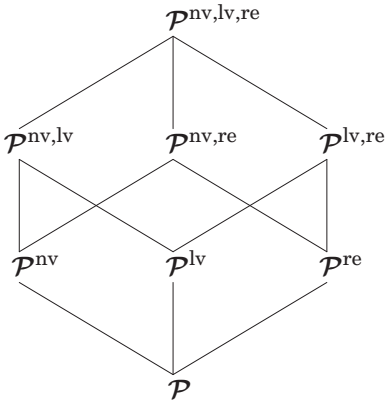


Fig. 3. Relationships between classes of graph patterns.

— Suppose finally that $h'_1(p) \notin \{n_1, n_3\}$. This is also not possible due to the fact that h'_1 is a homomorphism from π to G', and there are no edges in G' that start from nodes n_2 or n_4. □

To prove the statement, construct the following pattern π in $\mathcal{P}^{\text{lv}}$: It contain nodes $\{n_1, n_2, n_3, n_4\}$, and edges (n_1, X, n_2) and (n_3, X, n_4), where X is a label variable. Clearly, the graphs G and G', as defined in the statement of Claim 1, belong to $[\![\pi]\!]_\Sigma$. On the other hand, it is straightforward to prove that $G'' \notin [\![\pi]\!]_\Sigma$. Notice that if π' is a pattern in $\mathcal{P}^{\text{nv,re}}$ that is equivalent to π over Σ, then the set of node ids of π' must be exactly $\{n_1, n_2, n_3, n_4\}$. It follows from Claim 1 that there is no pattern π' in $\mathcal{P}^{\text{nv,re}}$ over Σ, such that $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$. This finishes the proof of the lemma. □

The proof of the second part of the theorem (that $\mathcal{P}^\sigma \sim \mathcal{P}^{\sigma'}$ if and only if $\sigma = \sigma'$) uses essentially the same arguments and is omitted.

The relationships mentioned in Theorem 4.1 are summarized in Figure 3.

In both relational and XML patterns, it is common to consider a restriction in which variables cannot be repeated. In relations, these are Codd tables [Imielinski and Lipski 1984] that model SQL's nulls. We say that a graph pattern is a *Codd pattern* if every variable – node or label – occurs at most once in it. In other words, Codd patterns do not allow us to express equality between unknown entities.

If σ contains nv or lv, we shall write $\mathcal{P}^\sigma_{\text{Codd}}$ for the Codd patterns in $\mathcal{P}^\sigma$. We next show that Codd patterns are strictly weaker than the usual patterns, and describe classes of patterns for which adding variables under Codd interpretation increases expressiveness.

PROPOSITION 4.3.

— *Codd patterns are strictly less expressive:* $\mathcal{P}^\sigma_{\text{Codd}} \prec \mathcal{P}^\sigma$ *when* σ *contains* nv *or* lv.
— *Adding variables under Codd interpretation makes patterns more expressive except adding label variables to regular expressions. That is, if* $\sigma' \subsetneq \sigma$ *and* $\sigma - \sigma'$ *contains either* nv *or* lv, *then* $\mathcal{P}^{\sigma'} \prec \mathcal{P}^\sigma_{\text{Codd}}$ *except one case:* $\mathcal{P}^{\text{re}} \sim \mathcal{P}^{\text{lv,re}}_{\text{Codd}}$.

PROOF. We begin with the last part of the second statement, namely that $\mathcal{P}^{\text{re}} \sim \mathcal{P}^{\text{lv,re}}_{\text{Codd}}$. Clearly, every pattern π in $\mathcal{P}^{\text{re}}$ is also in $\mathcal{P}^{\text{lv,re}}_{\text{Codd}}$. Then, we only need to prove that for every pattern π in $\mathcal{P}^{\text{lv,re}}_{\text{Codd}}$ over alphabet Σ there exists a pattern π' in $\mathcal{P}^{\text{re}}$ over Σ such that $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$.

Let $\pi = (N, E)$ be an arbitrary pattern in $\mathcal{P}^{\text{lv,re}}_{\text{Codd}}$ over alphabet Σ. We define a pattern $\pi' = (N', E')$ over Σ as follows:

— $N' = N$;
— E' contains all edges in E of the form (p, R, q), where R does not use label variables; and
— For each edge in E of the form (p, R, q) such that R uses label variables $X_1, \ldots, X_n$, let $R[X_1 \to a_1, \ldots, X_n \to a_n]$, for $a_1, \ldots, a_n \in \Sigma$, be the regular expression resulting of replacing each label variable X_i in R with the symbol a_i, for $1 \leq i \leq n$, and define

$$R' = \bigcup_{a_1,\ldots,a_n \in \Sigma} R[X_1 \to a_1, \ldots, X_n \to a_n] .$$

Then, E' contains the triple (p, R', q).

We first prove that $[\![\pi]\!]_\Sigma \subseteq [\![\pi']\!]_\Sigma$. Assume that the graph database G over Σ belongs to $[\![\pi]\!]_\Sigma$, and let $h = (h_1, h_2)$ be a homomorphism from π into G. We claim that $h = (h_1, h_2)$ is also a homomorphism from π' into G. (Notice that π' does not use label variables, so we may disregard h_2 in order to show that h is a homomorphism from π' into G). Clearly, h_1 sends nodes of π' into nodes of G, and is the identity on node ids. Thus, we only need to show that for every edge (p, R', q) in π', there is a path ρ in G from $h_1(p)$ to $h_1(q)$ such that $\lambda(\rho)$ belongs to $L(R')$. Let (p, R, q) be an arbitrary edge in π'. We have to consider two cases.

— There exists an edge of form (p, R, q) in π, in which case the proof is trivial.
— For some edge (p, R', q) in π, such that R' uses label variables $X_1, \ldots, X_n$, it is the case that $R = \bigcup_{a_1,\ldots,a_n \in \Sigma} R'[X_1 \to a_1, \ldots, X_n \to a_n]$. Then, we know that there is a path ρ from n_1 to n_2 in G such that $h_1(p) = n_1$, $h_1(q) = n_2$ and $\lambda(\rho)$ belongs to $h_2(R')$. But, clearly, $h_2(R)$ is of the form $R'[X_1 \to a_1, \ldots, X_n \to a_n]$, for some $a_1, \ldots, a_n \in \Sigma$. This implies that there is a path ρ in G from $h_1(p) = n_1$ to $h_1(q) = n_2$ in G such that $\lambda(\rho)$ belongs to $L(R)$.

Next, we show that $[\![\pi']\!]_\Sigma \subseteq [\![\pi]\!]_\Sigma$. Assume that G belongs to $[\![\pi']\!]_\Sigma$, and let $h = (h_1, h_2)$ be a homomorphism from π' into G. (Notice that π' does not use label variables, so we are only interested in the function h_1 that maps nodes of π' into nodes of G). Let $\mathcal{W}$ be the set of label variables mentioned in π. We construct a mapping $h'_2 : \mathcal{W} \to \Sigma$ such that $h' = (h_1, h'_2)$ is a homomorphism from π into G.

Define $h'_2 : \mathcal{W} \to \Sigma$ as follows. For each edge $e = (p, R, q)$ in π do the following: Assume that $X_1, \ldots, X_n$ are the label variables mentioned in R. Since $h = (h_1, h_2)$ is a homomorphism from π' into G, there is a path ρ_e in G from $h_1(p)$ to $h_1(q)$ such that $\lambda(\rho_e)$ belongs to $R' = \bigcup_{a_1,\ldots,a_n \in \Sigma} R[X_1 \to a_1, \ldots, X_n \to a_n]$. This means that $\lambda(\rho_e)$ belongs to $R[X_1 \to a^e_1, \ldots, X_n \to a^e_n]$, for some $a^e_1, \ldots, a^e_n \in \Sigma$. We then define $h'_2(X_i)$ to be a^e_i, for each $1 \leq i \leq n$. Notice that h'_2 defined in this way is indeed a mapping from $\mathcal{W}$ into Σ, as each variable X mentioned in π appears in exactly one edge of π. (This is because π belongs to $\mathcal{P}^{\text{lv,re}}_{\text{Codd}}$.)

We now show that $h' = (h_1, h'_2)$ is a homomorphism from π into G. Clearly, h_1 sends nodes of π into nodes of G, and is the identity on node ids. Thus, we only need to show that for every edge (p, R, q) in π, there is a path ρ in G from $h_1(p)$ to $h_1(q)$ such that $\lambda(\rho)$ belongs to $L(R)$. Let $e = (p, R, q)$ be an arbitrary edge in π. Once again, we have to consider two cases.

— Regular expression R does not use label variables, in which case the proof is trivial since π' also contains the edge (p, R, q).

— Regular expression R uses label variables $X_1, \ldots, X_n$. But then the path ρ_e in G goes from $h_1(p)$ to $h_1(q)$, and satisfies that $\lambda(\rho_e)$ belongs to $R[X_1 \to a_1^e, \ldots, X_n \to a_n^e]$. But, by definition, we have that $R[X_1 \to a_1^e, \ldots, X_n \to a_n^e] = h_2'(R)$, and thus ρ_e is a path from $h_1(p)$ to $h_1(q)$ such that $\lambda(\rho_e)$ belongs to $h_2'(R)$.

We conclude that $h' = (h_1, h_2')$ is a homomorphism from π into G, and hence that G belongs to $[\![\pi]\!]_\Sigma$.

Next, we prove that for all the remaining cases in which $\sigma' \subsetneq \sigma$ and $\sigma - \sigma'$ contains either nv or lv, it is the case that $\mathcal{P}^{\sigma'} \prec \mathcal{P}^{\sigma}_{\text{Codd}}$.

Let σ and σ' as stated. By definition, $\mathcal{P}^{\sigma'} \preceq \mathcal{P}^{\sigma}_{\text{Codd}}$. Thus, we only need to show that $\mathcal{P}^{\sigma'}$ and $\mathcal{P}^{\sigma}_{\text{Codd}}$ are not equally expressive. This follows easily from the following cases.

(1) There exists a pattern π in $\mathcal{P}^{\text{nv}}_{\text{Codd}}$ over $\Sigma = \{a\}$, such that $[\![\pi]\!]_\Sigma \neq [\![\pi']\!]_\Sigma$ for all patterns π' in $\mathcal{P}^{\text{lv,re}}$ over Σ.
(2) There exists a pattern π in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ over $\Sigma = \{a, b\}$, such that $[\![\pi]\!]_\Sigma \neq [\![\pi']\!]_\Sigma$ for all patterns π' in $\mathcal{P}^{\text{nv}}$ over Σ.

In particular, from case (1) we obtain that $\mathcal{P}^{\sigma'} \prec \mathcal{P}^{\sigma}_{\text{Codd}}$, for every $\sigma \subseteq \{\text{nv}, \text{lv}, \text{re}\}$ and $\sigma' \subseteq \{\text{lv}, \text{re}\}$ such that $\sigma' \subseteq \sigma$ and $\sigma - \sigma'$ contains nv. On the other hand, from case (2) we obtain that $\mathcal{P}^{\sigma'} \prec \mathcal{P}^{\sigma}_{\text{Codd}}$, for each $\sigma \subseteq \{\text{nv}, \text{lv}\}$ and $\sigma' \subseteq \{\text{nv}\}$ such that $\sigma' \subseteq \sigma$ and $\sigma - \sigma'$ contains lv.

Case (1) follows directly from the proof of the first statement of Lemma 4.2, as the proof only uses patterns in $\mathcal{P}^{\text{nv}}_{\text{Codd}}$. To prove case (2), we use the following fact: Let π be a pattern in $\mathcal{P}^{\text{nv}}$ over an alphabet Σ such that π contains at least one edge. Then there is a symbol $a \in \Sigma$ such that the certain answer to the Boolean RPQ $Q = Ans() \leftarrow (x, a, y)$ over π is `true`. Indeed, since π belongs to $\mathcal{P}^{\text{nv}}$, the edges of π are labeled only by symbols from Σ. Take an arbitrary edge in π, and assume that it is of the form (p, a, q), for $a \in \Sigma$. It is now easy to see that every graph G in $[\![\pi]\!]_\Sigma$ will contain an edge labeled with the symbol a. This proves that the certain answer to $Q = Ans() \leftarrow (x, a, y)$ over π is `true`.

We now continue with the proof of case (2). Let $\pi = (N, E)$ be the following pattern in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ over alphabet $\Sigma = \{a, b\}$: N contains two node ids n_1 and n_2, and E contains the edge (n_1, X, n_2), where X is a label variable. Notice then that $[\![\pi]\!]_\Sigma$ contains the graph database G_0 that consists only of the edge (n_1, a, n_2), as well as the graph database G_1 that consists only of the edge (n_1, b, n_2). Thus, it is easy to see that the certain answer to Q_0 and Q_1 over π is `false`, where $Q_0 = Ans() \leftarrow (x, a, y)$ and $Q_1 = Ans() \leftarrow (x, b, y)$. Furthermore, notice that each graph database in π contain at least one edge, so every pattern π' over Σ such that $[\![\pi]\!]_\Sigma = [\![\pi']\!]_\Sigma$ must also contain at least one edge. The proof then follows, by contradiction, from the fact we proved previously that for every pattern π in $\mathcal{P}^{\text{nv}}$ over Σ, such that π contains at least one edge, the certain answer to either the RPQ Q_0 or to the RPQ Q_1 over π must be `true`.

We prove next the first statement of the proposition, namely that $\mathcal{P}^{\sigma}_{\text{Codd}} \prec \mathcal{P}^{\sigma}$ when σ contains nv or lv. Again, by definition, it is the case that $\mathcal{P}^{\sigma}_{\text{Codd}} \preceq \mathcal{P}^{\sigma}$. Thus, we only need to prove that $\mathcal{P}^{\sigma}_{\text{Codd}}$ and $\mathcal{P}^{\sigma}$ are not equally expressive.

Assume first that σ contains lv, but not nv: that is, σ is {lv} or {lv, re}, and assume for the sake of contradiction that it holds that $\mathcal{P}^{\sigma}_{\text{Codd}} \sim \mathcal{P}^{\sigma}$. Using the same construction as in the proof for the second statement of this proposition, it is possible to show that $\mathcal{P}^{\sigma}_{\text{Codd}} \preceq \mathcal{P}^{\text{re}}$ (since, in particular, we have shown that $\mathcal{P}^{\text{lv,re}}_{\text{Codd}} \sim \mathcal{P}^{\text{re}}$). We then obtain

that $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\mathrm{re}}$, and then either $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\mathrm{re}}$, or $\mathcal{P}^{\sigma} \sim \mathcal{P}^{\mathrm{re}}$. However, any of these two facts contradicts Theorem 4.1.

Next, assume that σ contains nv. To prove that $\mathcal{P}^{\sigma}_{\mathrm{Codd}}$ is not equally expressive as $\mathcal{P}^{\sigma}$, we shall prove a more general statement: There exists a pattern π in $\mathcal{P}^{\mathrm{nv}}$ over alphabet $\Sigma = \{a\}$, such that $[\![\pi]\!]_{\Sigma} \neq [\![\pi']\!]_{\Sigma}$ for all π' in $\mathcal{P}^{\mathrm{nv,lv,re}}_{\mathrm{Codd}}$ over Σ.

Let π be the pattern over alphabet $\{a\}$ that consists of the single edge (x,a,x), where x is a node variable. Then, notice that all database graphs $G \in [\![\pi]\!]_{\Sigma}$ must contain at least one edge that forms a self-loop with a node of G. Assume now, for the sake of contradiction, that there is a pattern π' in $\mathcal{P}^{\mathrm{nv,lv,re}}_{\mathrm{Codd}}$ over Σ, such that $[\![\pi]\!]_{\Sigma} = [\![\pi']\!]_{\Sigma}$. Then, it is clear that π' contains no node ids (since homomorphisms are enforced to be the identity on constants). We now prove the following fact that implies that $[\![\pi]\!]_{\Sigma} \neq [\![\pi']\!]_{\Sigma}$, which is the desired contradiction: Let $\pi = (N,E)$ be a pattern in $\mathcal{P}^{\mathrm{nv,lv,re}}_{\mathrm{Codd}}$ over alphabet $\{a\}$ such that N does not contain node ids. Then, there exists a graph $G \in [\![\pi]\!]_{\Sigma}$ that does not contain any self loops.

Indeed, consider the graph database G resulting of replacing each node variable x in π with a fresh constant n_x, and each edge $e = (x,L,y)$ of π with a path ρ_e of fresh node ids from n_x to n_y, such that $\lambda(\rho)$ satisfies the regular expression L' that is obtained by replacing each label variable in L with letter a. (Notice that paths of the form ρ_e are node and edge disjoint; that is, only start and end nodes can be shared between them.) Clearly, G belongs to $[\![\pi]\!]_{\Sigma}$ and contains no self-loops.

This finishes the proof of Proposition 4.3. □

5. QUERY ANSWERING

The goal of this section is to study the complexity – both data and combined – of query answering over graph patterns. Recall that for queries Q returning tuples of nodes, we want to find certain answers defined as $\mathrm{CERTAIN}(Q,\pi) = \bigcap\{Q(G) \mid G \in [\![\pi]\!]\}$. More precisely, one needs to find $\mathrm{CERTAIN}_{\Sigma}(Q,\pi)$, with G ranging over graph databases with edges labeled in Σ; it will be clear from the proofs, however, that the complexity of query answering does not depend on the labeling alphabet.

Since each class of patterns gives rise to a class of graph queries $Q = (\xi,\bar{x})$, one could potentially ask for the exact bounds on combined and data complexity for all these classes of queries on all the classes of patterns. Of course, we are not going to consider all the resulting 128 cases. Instead, we do the following.

As our benchmark language, we use CRPQs, and provide exact complexity bounds for CRPQs over all classes of patterns. Recall that CRPQs can be viewed as graph queries $(\xi,\bar{x})$ with $\xi \in \mathcal{P}^{\mathrm{nv,re}}$. We then show that the upper bounds for CRPQs extend to the most expressive patterns from $\mathcal{P}^{\mathrm{nv,lv,re}}$. After that, we delve further into intractable cases, and analyze what really causes intractability. In such cases, we consider restricted classes of queries based on simpler graph patterns.

Certain Answers as Pattern Implication. It is a standard and yet useful observation that the problem of computing certain answers can be cast as the problem of *implication of patterns*. Recall that pattern implication is defined as follows: if π_1 and π_2 are two patterns, then we say that π_1 *implies* π_2, and write $\pi_1 \models \pi_2$ if $[\![\pi_1]\!] \subseteq [\![\pi_2]\!]$. In other words, $\pi_1 \models \pi_2$ if $G \models \pi$ entails $G \models \pi_2$ for every graph database G. The following is now immediate from the definitions.

LEMMA 5.1. *Given a graph pattern $\pi = (N,E)$ and a graph query $Q = (\xi,\bar{x})$ with $|\bar{x}| = k$,*

$$\mathrm{CERTAIN}(Q,\pi) \;=\; \{\bar{v} \in N^k \mid \pi \models \xi[\bar{v}/\bar{x}]\}.$$

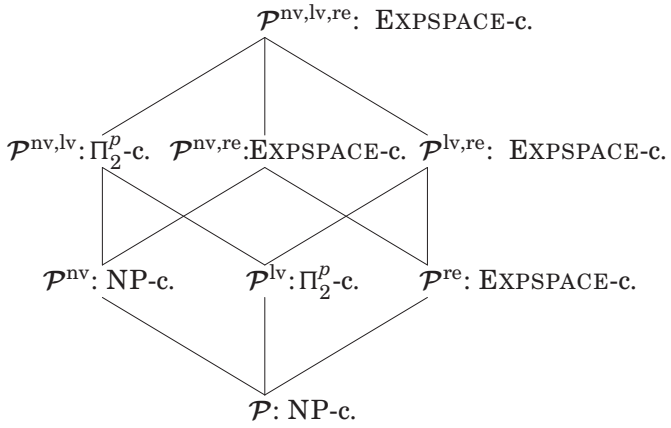


Fig. 4. Combined complexity for CRPQs over graph patterns.

For Boolean graph queries $Q = (\xi, ())$ with the empty tuple of output variables (i.e., `true`/`false` queries), Lemma 5.1 states that $\text{CERTAIN}(Q, \pi) = \texttt{true}$ if and only if $\pi \models \xi$. This simple connection with the implication problem will let us use known results on containment of CRPQs [Calvanese et al. 2000b] to obtain some of the bounds for the combined complexity of query answering.

Remark. Using Naive Evaluation. Some classes of patterns can be represented as naive tables, perhaps with constraints. For example, patterns from $\mathcal{P}^{\mathrm{nv}}$ can be stored as naive tables, and patterns without regular expressions (from $\mathcal{P}^{\mathrm{nv,lv}}$) are represented as relational naive tables with an additional constraint that the interpretation for label variables must come from the labeling alphabet Σ. This can easily be coded as an inclusion constraint.

Since CRPQs can be expressed in datalog, such a representation gives us good tractable bounds for data complexity for $\mathcal{P}^{\mathrm{nv}}$ patterns. But for combined complexity, and for data complexity for other classes, we cannot use known results to get tight bounds. For example, even evaluating conjunctive queries over naive tables with inclusion constraints is known to be PSPACE-hard [Johnson and Klug 1984], and we shall see better bounds obtained for CRPQs over $\mathcal{P}^{\mathrm{nv,lv}}$ patterns.

5.1. Combined Complexity

The problem we are dealing with is as follows:

INPUT:	A pattern $\pi = (N, E)$, a graph query $Q = (\xi, \bar{x})$ with $\lvert\bar{x}\rvert = k$, a tuple $\bar{v} \in N^k$.
QUESTION:	Is $\bar{v} \in \text{CERTAIN}(Q, \pi)$?

Checking $\bar{v} \in \text{CERTAIN}(Q, \pi)$ amounts to checking $\pi \models \xi[\bar{v}/\bar{x}]$, and the problem is known to be EXPSPACE-complete when both π and ξ are in $\mathcal{P}^{\mathrm{nv,re}}$ [Calvanese et al. 2000b]. We now provide a complete analysis of the complexity.

THEOREM 5.2. *The combined complexity of answering CRPQs over classes of graph patterns is as shown in Figure* 4 *(The abbreviation "-c." in the figure means, of course, complete for the class).*

PROOF. The EXPSPACE upper bound for $\mathcal{P}^{\text{nv,lv,re}}$ follows from Proposition 5.3. The EXPSPACE-hardness for patterns in $\mathcal{P}^{\text{re}}$ follows by easily adapting the proof of Theorem 6 in Calvanese et al. [2000b], which proves that containment of CRPQs is EXPSPACE-complete. Indeed, it immediately follows from such result that combined complexity of CRPQs over patterns in $\mathcal{P}^{\text{nv,re}}$ is EXPSPACE-hard. By slightly adapting the reduction, one can also show that the problem remains EXPSPACE-hard over the class of patterns in $\mathcal{P}^{\text{re}}$.

For patterns in $\mathcal{P}$ and $\mathcal{P}^{\text{nv}}$, notice that the certain answers of a CRPQ Q over a pattern π in $\mathcal{P}^{\text{nv}}$ can be obtained by naïve evaluation of Q over π; that is, by directly evaluating Q over π, treating node variables as if they were ordinary node ids. This can be simply explained by the following facts: (i) CRPQs are preserved under homomorphisms, and (ii) patterns in $\mathcal{P}^{\text{nv}}$ can be represented as relational naïve tables [Libkin 2011]. The problem of evaluating a CRPQ over a graph database G is in NP [Barceló et al. 2010a], and hence performing a naïve evaluation (and, therefore, computing certain answers) of a CRPQ over a pattern in $\mathcal{P}^{\text{nv}}$ is in NP. The problem is clearly also NP-hard, even over $\mathcal{P}$, as it contains as a subinstance the problem of conjunctive query evaluation over graphs.

Next we prove the Π_2^p-completeness for the classes $\mathcal{P}^{\text{lv}}$ and $\mathcal{P}^{\text{nv,lv}}$. First, we show that Π_2^p is an upper bound for the problem over patterns in $\mathcal{P}^{\text{nv,lv}}$. Let π be a graph pattern in $\mathcal{P}^{\text{nv,lv}}$ and Q a CRPQ, both over alphabet Σ. Assume, without loss of generality, that Q is Boolean and that $\mathcal{W}$ is the set of label variables mentioned in π. Then, clearly $\text{CERTAIN}(Q, \pi) = \texttt{false}$ if and only for some mapping $\nu : \mathcal{W} \to \Sigma$ it is the case that $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$, where π_ν is the graph pattern in $\mathcal{P}^{\text{nv}}$ that is obtained from π by simultaneously replacing each label variable $X \in \mathcal{W}$ with $\nu(X)$. Then, a Σ_2^p algorithm that checks whether $\text{CERTAIN}(Q, \pi) = \texttt{false}$ does the following: It first guesses a polynomial size mapping $\nu : \mathcal{W} \to \Sigma$, where $\mathcal{W}$ is the set of label variables mentioned in π. Then, it constructs in polynomial time the pattern π_ν in $\mathcal{P}^{\text{nv}}$, and checks that $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$. As we mentioned previously, the latter can be solved in coNP.

The proof of Π_2^p-hardness for the class $\mathcal{P}^{\text{lv}}$ is rather technical, and can be found in the appendix. □

Theorem 5.2 tells us that the combined complexity of CRPQs on usual graph databases is the same as the combined complexity of conjunctive queries over usual relational databases, that is, NP-complete. Thus, adding node variables comes with no cost, while adding both node and label variables carries a small cost in terms of combined complexity (jumping up one level in the polynomial hierarchy). Adding regular expressions comes at a significant cost (jumping up an exponential).

Using essentially the same techniques as in Calvanese et al. [2000b], we can prove that the previous upper bound extends beyond CRPQs.

PROPOSITION 5.3. *The combined complexity of arbitrary graph queries on arbitrary patterns is in* EXPSPACE.

PROOF. The containment problem for CRPQs is known to be in EXPSPACE [Calvanese et al. 2000b]. The EXPSPACE algorithm proposed in Calvanese et al. [2000b] does the following: Given two CRPQs, Q_1 and Q_2, the algorithm first constructs in EXPSPACE an NFA $\mathcal{A}_1$, of exponential size, that accepts precisely the "codifications" of the graph databases that satisfy Q_1, and then constructs in EXPSPACE an NFA $\mathcal{A}_2$, of double-exponential size, that accepts precisely the "codifications" of the graph databases that do not satisfy Q_2. Then, it is possible to prove that $Q_1 \not\subseteq Q_2$ if and

only the language accepted by $\mathcal{A}_1 \cap \mathcal{A}_2$ is nonempty. The latter can be done in EXPSPACE by using a standard "on-the-fly" verification algorithm. We use this idea to show that the implication problem (i.e., the containment problem) between arbitrary graph patterns in $\mathcal{P}^{\text{nv,lv,re}}$ can also be solved in EXPSPACE. Allowing constants in CRPQs comes at no cost, and esentially the same construction shows that containment of CRPQs with constants (and, thus, implication of patterns in $\mathcal{P}^{\text{nv,re}}$) can be solved in EXPSPACE.

Let π be a graph pattern $\mathcal{P}^{\text{nv,lv,re}}$, and let Q be a graph query such that its underlying graph pattern ξ also belongs to $\mathcal{P}^{\text{nv,lv,re}}$. Suppose that both patterns are defined over alphabet Σ and that the set of label variables used in π or ξ is $\mathcal{W}$. We assume without loss of generality that Q is Boolean. (Indeed, since patterns in $\mathcal{P}^{\text{nv,lv,re}}$ are allowed to make use of node ids, this is not a restriction, at least in terms of the complexity analysis.) Then clearly, $\text{CERTAIN}(Q, \pi) = \texttt{false}$ if and only if for some assignment $\nu : \mathcal{W} \to \Sigma$ it is the case that $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$, where π_ν is the pattern in $\mathcal{P}^{\text{nv,re}}$ that is obtained from π by replacing each occurrence of the label variable X with $\nu(X)$. Notice that π_ν is a pattern in $\mathcal{P}^{\text{nv,re}}$.

First, we show that for each valuation $\nu : \mathcal{W} \to \Sigma$, the problem of checking whether $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$ can be solved in EXPSPACE. Clearly, $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$ if and only if there is a graph database $G \in [\![\pi_\nu]\!]$ such that for each mapping $\nu' : \mathcal{W} \to \Sigma$ it is the case that $G \notin [\![\xi_{\nu'}]\!]$. (Notice that $\xi_{\nu'}$ belongs to $\mathcal{P}^{\text{nv,re}}$, for each mapping $\nu' : \mathcal{W} \to \Sigma$.) First, construct in EXPSPACE an automaton $\mathcal{A}^{\nu}_{\pi}$, of exponential size, that accepts precisely the "codifications" of the graph databases that belong to $[\![\pi_\nu]\!]$ – as done in Calvanese et al. [2000b] and explained at the beginning of the proof. Then, for each valuation $\nu' : \mathcal{W} \to \Sigma$, construct in EXPSPACE an automaton $\mathcal{A}^{\nu'}_{\xi}$, of double-exponential size, that accepts precisely the "codifications" of the graph databases that do not belong to $[\![\xi_{\nu'}]\!]$ – as done in Calvanese et al. [2000b] and explained at the beginning of the proof. Then, $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$ if and only the language accepted by the NFA $\mathcal{B} = \mathcal{A}^{\nu}_{\pi} \cap \bigcap_{\nu' : \mathcal{W} \to \Sigma} \mathcal{A}^{\nu'}_{\xi}$ is nonempty. Notice that the size of $\mathcal{B}$ is double-exponential on the size of the input, and, further, that checking whether $\mathcal{B}$ accepts some word can be done in EXPSPACE using a standard "on-the-fly" verification algorithm.

Thus, an EXPSPACE procedure that checks whether $\text{CERTAIN}(Q, \pi) = \texttt{false}$ does the following: For each $\nu : \mathcal{W} \to \Sigma$, the procedure first constructs π_ν and then checks whether $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$ using the algorithm described in the previous paragraph. If $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$, for some $\nu : \mathcal{W} \to \Sigma$, then we declare $\text{CERTAIN}(Q, \pi) = \texttt{false}$. Otherwise, we declare $\text{CERTAIN}(Q, \pi) = \texttt{true}$. Clearly, the whole procedure can be performed in exponential space. □

The next question is whether we can lower the EXPSPACE bound for patterns in $\mathcal{P}^{\text{re}}$. There are two natural ways of looking for better behaved subclasses: by restricting queries, or restricting patterns. Restrictions on queries by means of simplifying regular languages were studied in Deutsch and Tannen [2001]. For example, it showed that for regular languages built with concatenation and the Kleene star, the combined complexity drops to Π^p_2-complete. Another possibility is to restrict to RPQs; then, using techniques similar to Calvanese et al. [2000b], we can prove a PSPACE bound, matching the combined complexity of relational calculus. It also follows from Calvanese et al. [2000b] that restricting the class of patterns does not help lower the combined complexity.

PROPOSITION 5.4.

— *The combined complexity of answering CRPQs on patterns $\pi \in \mathcal{P}^{\text{re}}$ is* EXPSPACE*-hard even for patterns π that contain a single edge.*

— *The combined complexity of answering RPQs on graph patterns from* $\mathcal{P}^{\text{nv,lv,re}}$ *is* PSPACE*-complete. The problem remains* PSPACE*-hard even for answering RPQs on patterns* $\pi \in \mathcal{P}^{\text{re}}$ *that contain a single edge.*

PROOF. The first part follows directly from the proof of Theorem 6 in Calvanese et al. [2000b]. Next, we prove the second part.

It follows from the proof of Theorem 5 in Calvanese et al. [2000b], that the problem of checking whether a CRPQ Q_1 is contained in CRPQ Q_2 can be solved in PSPACE, if we assume the number of variables used in Q_2 to be fixed. It immediately follows that checking whether a CRPQ is contained in an RPQ is in PSPACE. Again, allowing constants in CRPQs comes at no cost, and esentially the same construction shows that containment of a CRPQ with constants into an RPQ (and, thus, combined complexity of answering RPQs on patterns in $\mathcal{P}^{\text{nv,re}}$) can be solved in EXPSPACE. Next, we use this fact to construct a PSPACE procedure that checks, for a given pattern $\pi \in \mathcal{P}^{\text{nv,lv,re}}$ and a RPQ Q, whether $\text{CERTAIN}(Q, \pi) = \texttt{true}$.

Let π be an arbitrary graph pattern in $\mathcal{P}^{\text{nv,lv,re}}$ and Q an arbitrary RPQ. Again, we can assume without loss of generality that Q is Boolean. Assume that both π and Q are defined over alphabet Σ and that $\mathcal{W}$ is the set of label variables used in π. Then, it is clear that $\text{CERTAIN}(Q, \pi) = \texttt{false}$ if and only if for some mapping $\nu : \mathcal{W} \to \Sigma$ it is the case that $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$, where π_ν is the pattern in $\mathcal{P}^{\text{nv,re}}$ that is obtained from π by replacing each label variable $X \in \mathcal{W}$ with $\nu(X)$. Notice that each pattern of the form π_ν, for ν a mapping from $\mathcal{W}$ to Σ, is a CRPQ.

It is clear that checking whether $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$ can be done in PSPACE. Indeed, this is equivalent to checking whether the pattern π_ν in $\mathcal{P}^{\text{nv,re}}$ is contained in the RPQ Q, which by the observations provided above can be solved in polynomial space. Now, define a procedure that does the following: For each mapping $\nu : \mathcal{W} \to \Sigma$, first construct π_ν and then compute $\text{CERTAIN}(Q, \pi_\nu)$. If $\text{CERTAIN}(Q, \pi_\nu) = \texttt{false}$, for some $\nu : \mathcal{W} \to \Sigma$, then we declare $\text{CERTAIN}(Q, \pi) = \texttt{false}$. Otherwise, we declare $\text{CERTAIN}(Q, \pi) = \texttt{true}$. Clearly, the whole procedure can be performed in polynomial space.

The PSPACE-hardness for RPQs over patterns in $\mathcal{P}^{\text{re}}$ that contain a single edge follows from the following reduction from the problem of containment of regular expressions, which is known to be PSPACE-hard. Assume that L and L' are two regular expressions over alphabet Σ. Let a and a' be two distinct symbols that do not belong to Σ. Define π_L to be the following graph pattern in $\mathcal{P}^{\text{re}}$: (n, aLa', n'). Notice that π_L is defined over alphabet $\Sigma \cup \{a, a'\}$ and has a single edge. Further, define RPQ $Q_{L'}$ to be $Ans() \leftarrow (x, aL'a', y)$. Then it can be easily proved that $\text{CERTAIN}(Q_{L'}, \pi_L) = \texttt{true}$ if and only if $L \subseteq L'$. Further, π_L and $Q_{L'}$ can be constructed in polynomial time from L and L'. This finishes our proof. □

5.2. Data Complexity

We now turn to data complexity, i.e. the complexity of query answering when the query is fixed. In what follows, Q refers to a graph query $(\xi, \bar{x})$ with $|\bar{x}| = k$.

PROBLEM:	DATA COMPLEXITY(Q)
INPUT:	a pattern $\pi = (N, E)$, a tuple $\bar{v} \in N^k$.
QUESTION:	Is $\bar{v} \in \text{CERTAIN}(Q, \pi)$?

Notice that this can also be viewed as a pattern-implication problem $\pi \models \xi[\bar{v}/\bar{x}]$, but for a *fixed* pattern ξ.

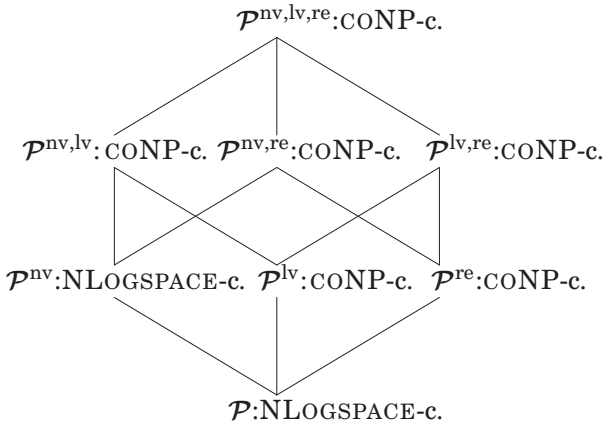


Fig. 5. Data complexity for CRPQs over graph patterns.

As already mentioned, some cases are simple: for example, patterns in $\mathcal{P}$ are graphs, and thus due to the monotonicity of CRPQs, computing certain answers is the same as evaluating CRPQs on graphs, that is, NLogspace-complete. Similarly, since $\mathcal{P}^{nv}$ patterns can be represented as naïve tables, and since CRPQ certain answers can be obtained by naïve evaluation over naïve tables, we retain an NLogspace bound. For other cases, as it turns out, the complexity is intractable.

Theorem 5.5. *The data complexity of answering CRPQs over classes of graph patterns is as shown in Figure* 5.

Proof Sketch. We have already explained how to obtain the NLogspace upper bounds, and we shall prove a stronger coNP upper bound in Proposition 5.6. We now present a simple hardness proof for $\mathcal{P}^{re}$. It will be tightened significantly (and extended to $\mathcal{P}^{lv}$) in the remainder of the section.

For $\mathcal{P}^{re}$, we use reduction to non-3-colorability. Assume we have an arbitrary undirected graph G; we represent it as a labeled graph where between two nodes n_1 and n_2 connected by an edge we have two edges labeled a, that is, (n_1, a, n_2) and (n_2, a, n_1). Now we turn it into a $\mathcal{P}^{re}$ pattern π_G over the alphabet $\{a, r, g, b\}$ by adding edges $(n, rr|gg|bb, n)$ for each node n. That is, in every graph represented by this pattern, associated with each node n there is a node n' and edges $(n, \ell, n'), (n', \ell, n)$ where ℓ is one of r, g, b. It is now easy to see that the certain answer to the Boolean RPQ $Ans() \leftarrow (x, rar|gag|bab, y)$ over π_G is `true` if and only if G is not 3-colorable.

The upper bound again extends to arbitrary queries. In order to prove this, we apply similar techniques to those used in Calvanese et al. [2000a] to show that the data complexity of the problem of answering RPQs using views is in coNP.

Proposition 5.6. *Data complexity of arbitrary graph queries over arbitrary graph patterns is in* coNP.

Proof. Before starting the proof, we need to introduce the notion of a *canonical* graph database for a graph pattern π, which will be useful for the rest of the proof. Let σ be an assignment from the nodes of π into $\mathcal{N}$ such that (1) σ is the identity map on node ids, and (2) σ assigns a fresh node id n_x to each node variable x mentioned in π. (In particular, n_x does not appear in π.) Then, we say that σ is *canonical* for π.

Let π be a graph pattern over Σ. Assume that π consists of the edges $\{(p_i, L_i, q_i) \mid 1 \leq i \leq m\}$, where each p_i and q_i is either a node variable or a node id and each L_i belongs to $\mathrm{REG}(\Sigma \cup \mathcal{V}_{\mathrm{lab}})$ $(1 \leq i \leq m)$. Further, let σ be a canonical assignment for π. Then, the graph database G over Σ is *σ-canonical* for π if and only if there is a mapping $\nu : \mathcal{V}_{\mathrm{lab}} \to \Sigma$ such that the following holds:

— G consists of m simple paths, one for each edge in π, which are node and edge disjoint, that is, only the start and end nodes can be shared between different paths; and
— for each $1 \leq i \leq m$, if ρ_i is the path associated with the edge (p_i, L_i, q_i), then ρ_i starts in the node id $\sigma(p_i)$ and ends in the node id $\sigma(q_i)$, and $\lambda(\rho_i) \in \nu(L_i)$.

From now on, whenever G is σ-canonical for π, for some canonical assignment σ, then we simply say that it is canonical for π. Clearly, if G is canonical for π, then $G \models \pi$.

Using essentially the same techniques as in Calvanese et al. [2000b] it is possible to prove the following semantic characterization.

CLAIM 2. *For each graph query Q and tuple $\bar{n}$ of node ids in π, it is the case that $\bar{n} \notin \mathrm{CERTAIN}(Q, \pi)$ if and only if there is a a graph database G over Σ that is canonical for π and such that $\bar{n} \notin Q(G)$.*

Now we have the appropriate tools to prove the proposition. Let Q be a fixed graph query over the fixed alphabet Σ. We assume without loss of generality that Q is Boolean (indeed, since queries are allowed to make use of node ids this is not a restriction). We first prove the following small model property: There is a polynomial $p(x)$ such that for every graph pattern π over Σ, if

(1) there is a graph database $G \in [\![\pi]\!]$ such that $Q(G) = \texttt{false}$, and
(2) every node id that is mentioned in Q is also mentioned in π,

then there is a canonical graph database G' for π such that (1) $Q(G') = \texttt{false}$, and (2) the length of each path in G' that is associated with an edge of π is bounded by $p(|\pi|)$, where $|\pi|$ is the size of π. (Notice that this immediately implies that G' is of size polynomial on $|\pi|$). We prove this by applying usual cutting techniques.

Let π be a graph pattern over Σ. Assume that every node id that is mentioned in Q also appears in π. Further, assume that there is a graph database $G \in [\![\pi]\!]$ such that $Q(G) = \texttt{false}$. Then, we can also assume, without loss of generality, that G is σ-canonical for π via some mapping $\nu : \mathcal{V}_{\mathrm{lab}} \to \Sigma$, for some canonical assignment σ (Claim 2). The problem is that some paths in G may be too long, and, thus, not necessarily every path in G that is associated with some edge of π is of polynomial size. Next, we show how to prune the long paths in G without changing its semantics with respect to π and Q.

Consider the query Q' defined as $\bigvee_{\{\nu \mid \nu : \mathcal{V}_{\mathrm{lab}} \to \Sigma\}} Q_\nu$, where Q_ν is the graph query obtained from Q by simultaneously replacing each label variable X mentioned in Q with $\nu(X)$. Clearly, Q' is a finite disjunction of graph queries whose underlying graph pattern belongs to $\mathcal{P}^{\mathrm{nv,re}}$. We assume the semantics of disjunctions of graph queries to be defined in the standard way from the semantics of graph queries. Then, it is not hard to see that $Q(G) = \texttt{false}$ if and only if $Q'(G) = \texttt{false}$.

Further, Q' is a CRPQ with constants, and hence it can be expressed as a sentence ϕ in *monadic* second-order logic (MSO) – which is the extension of first-order logic with quantification over sets – with the help of constants for the node ids that appear in Q.

The vocabulary of ϕ consists of binary relation symbols E_a, for each $a \in \Sigma$. A graph database G over Σ can be interpreted in the standard way as a first-order structure $\mathcal{S}_G$ over this vocabulary: The interpretation of symbol E_a in this structure contains all pairs (n, n') of node ids in G such that there is an edge labeled a from n to n' in G. Then, one can construct ϕ in such a way that $G \models Q' \Leftrightarrow \mathcal{S}_G \models \phi$, for each graph database G.

Assume that the quantifier depth of ϕ is $k \geq 0$. Notice that k depends only on ϕ. It is well known that there is a finite number of different *rank-k MSO types* (cf., Libkin [2004]) of words over vocabulary Σ with one distinguished element. Assume that such a number is $K \geq 0$. Again, K only depends on k, and thus, on ϕ.

Also, with each regular language of the form $\nu(L)$, where L is a regular language in $\mathrm{REG}(\Sigma \cup \mathcal{V}_{\mathrm{lab}})$ that appears in π, we associate an NFA $\mathcal{A}_{\nu(L)}$ that recognizes $\nu(L)$. Since each regular language can be converted into an equivalent NFA of polynomial size, we can assume that there is a polynomial $p'(x)$ such that the number of states of each NFA of the form $\mathcal{A}_{\nu(L)}$ is bounded by $p'(|L|)$, and hence by $p'(|\pi|)$.

Let $\rho = n_0 a_0 n_1 \cdots a_{\ell-1} n_\ell a_\ell n_{\ell+1}$ be an arbitrary path in G, such that both n_0 and $n_{\ell+1}$ are mentioned in π, but none of the node ids $n_1, \ldots, n_\ell$ is mentioned in π. Recall that G is σ-canonical for π, and, thus, ρ is associated with some edge (p, L, q) in π. That is, $\sigma(p) = n_0$, $\sigma(q) = n_{\ell+1}$ and $a_0 a_1 \cdots a_\ell$ belongs to $\nu(L)$. With each node n_i, $1 \leq i \leq \ell$, we associate a pair (α_1^i, α_2^i) such that:

— α_1^i is the rank-k type of the word $\lambda(\rho^i_{\rightarrow})$, where $\rho^i_{\rightarrow} = n_i a_i n_{i+1} \cdots a_\ell n_{\ell+1}$; and
— α_2^i is the rank-k type of the word $\lambda(\rho^i_{\leftarrow})$, where $\rho^i_{\leftarrow} = n_0 a_0 \cdots a_{i-2} n_{i-1} a_{i-1}$.

Then, it is clear that if $\ell \geq p'(|\pi|) \cdot K + 3$, there must be two nodes n_i and n_j ($2 \leq i < j \leq \ell$) such that (1) $\alpha_1^i = \alpha_1^j$ and $\alpha_2^i = \alpha_2^j$, and (2) there is an accepting run of $\mathcal{A}_{\nu(L)}$ over $a_0 a_1 \cdots a_\ell$ such that the state assigned by this run to position $i-1$ is the same than the one assigned to position $j-1$. Thus, the word $a_0 a_1 \cdots a_{i-1} a_j \cdots a_\ell$ belongs to $\nu(L)$, and further, if G' is the graph database that is obtained from G by replacing path ρ by path $\rho' = n_0 a_0 n_1 \cdots a_{i-1} n_i a_j n_{j+1} \cdots n_\ell a_\ell n_{\ell+1}$, then $G' \models \pi$.

We need to show now that the semantics of Q is invariant with respect to G and G'. First, assume that $\bar{n}$ is the tuple of all distinct node ids mentioned in π. Then, G contains each node id n mentioned in $\bar{n}$, and so does G' (because we only cut internal node ids of paths in G that are associated to edges in π, and those nodes – since G is canonical for π – do not appear in π). Further, let $(G, \bar{n})$ and $(G', \bar{n})$ be the first-order structures that extend the standard first-order interpretations of G and G' over vocabulary $\{E_a \mid a \in \Sigma\}$ with distinguished tuple $\bar{n}$. By using a standard Ehrenfeucht-Fraïssè game argument for MSO, it is possible to prove that $(G, \bar{n})$ and $(G', \bar{n})$ are indistinguishable by MSO sentences of quantifier rank $\leq k$. (This is due to the facts that (1) $\alpha_1^i = \alpha_1^j$ and $\alpha_2^i = \alpha_2^j$ implies that the rank-k types of $\lambda(\rho)$ and $\lambda(\rho')$ are the same, and (2) there are no two different paths in G that share internal nodes from ρ). We conclude that $(G, \bar{n}) \models \phi$ if and only if $(G', \bar{n}) \models \phi$ (since every node id that is mentioned in ϕ is among those in $\bar{n}$), and, thus, $Q'(G) = \text{false}$ iff $Q'(G') = \texttt{false}$. Therefore, $Q(G) = \texttt{false}$ iff $Q(G') = \texttt{false}$.

By recursively applying the cutting technique, one can show that if there is a graph database $G \in [\![\pi]\!]$ such that $\bar{n} \notin Q(G)$, then there is a graph database G' that is canonical for π, $Q(G') = \texttt{false}$, and the length of each path in G' that is associated with an edge of π is bounded by the polynomial $p'(|\pi|) \cdot K + 4$. This finishes the proof of our small model property. Next, we continue with the proof of the proposition.

In order to do this, we design an NP algorithm that verifies $\textsc{certain}(Q, \pi) = \texttt{false}$. Let π be a graph pattern over Σ. If Q contains some node id that does not appear in π,

then clearly CERTAIN$(Q,\pi) = \texttt{false}$. If this is not the case, then we can use our small model property.

The algorithm first guesses an assignment ν from the label variables mentioned in π into alphabet Σ. Then, it guesses a canonical graph G for π via assignment ν, such that the length of each path in G that is associated to some edge in π is bounded by $p'(|\pi|) \cdot K + 4$. Clearly, both ν and G are polynomial size witnesses. Finally, the algorithm checks that $Q'(G) = \texttt{false}$, which can be done in polynomial time [Consens and Mendelzon 1990]. □

Looking at Figure 5, we see that there are two features that cause coNP-hardness: label variables, and regular expressions. We now analyze their role in causing the high complexity of query answering. In both cases, we need to investigate two ways of lowering the complexity: by restricting queries, and by restricting their inputs.

The Role of Label Variables. For restrictions on queries, we shall look at simple RPQs. To define restrictions on inputs, we use the notion of the *underlying graph* G_π of a pattern $\pi = (N,E)$: this is simply the graph obtained by erasing labels on edges, that is, $G_\pi = (N, \{(v_1,v_2) \mid (v_1,L,v_2) \in E)$.

We now show that the coNP-hardness result is very robust. Recall that $\mathcal{P}^\sigma_{\text{Codd}}$ stands for class of Codd patterns in $\mathcal{P}^\sigma$, that is, patterns that use each variable once.

THEOREM 5.7.

— *There is a Boolean RPQ Q such that* DATA COMPLEXITY*(Q) is* coNP*-hard even over input patterns in $\mathcal{P}^{\text{lv}}$ whose underlying graph is a path. Moreover, the regular language in Q is built using only concatenation and the Kleene star.*

— *There is a Boolean RPQ Q of the form $Ans() \leftarrow (x,w,y)$, where w is a word in $\{0,1\}^*$, such that* DATA COMPLEXITY*(Q) is* coNP*-hard even over $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ patterns whose underlying graph is a DAG.*

PROOF. We prove the first part and leave the second, more technical proof, to the appendix. We prove that there exists a Boolean RPQ Q of the form $Ans() \leftarrow (x,L,y)$, where L is a regular expression built using only concatenation and Kleene star, and DATA COMPLEXITY(Q) is coNP-hard even over input patterns in $\mathcal{P}^{\text{lv}}$ whose underlying graph is a path.

We establish a reduction from monotone 1-in-3 3SAT, which is known to be NP-hard, to the complement of DATA COMPLEXITY(Q). The input to monotone 1-in-3 3SAT is a conjunction ϕ of clauses, with exactly three literals each, in which no negated variable occurs. The problem is determining whether there is a truth assignment to the variables so that each clause has exactly one true variable.

Let $\Sigma = \{\#, 0, 1, in, out\}$. The query Q over Σ is the Boolean RPQ that consists of the atom $Ans() \leftarrow (x,L,y)$, where L is the regular language $in \cdot L_1^* \cdot L_2^* \cdots L_{10}^* \cdot out$, and languages L_i, $1 \le i \le 10$, are defined as follows (we assume that Σ^* corresponds to the expression $(in^*0^*1^*\#^*out^*)^*$):

$$L_1 := \Sigma^* in \Sigma^*; \quad L_2 := \Sigma^* out \Sigma^*; \quad L_3 := \Sigma^* \#\# \Sigma^*$$
$$L_4 := \Sigma^* \#0\# \Sigma^*; \quad L_5 := \Sigma^* \#1\# \Sigma^*; \quad L_6 := \Sigma^* \#111 \Sigma^*;$$
$$L_7 := \Sigma^* \#011 \Sigma^*; \quad L_8 := \Sigma^* \#101 \Sigma^*; \quad L_9 := \Sigma^* \#110 \Sigma^*; \quad L_{10} := \Sigma^* \#000 \Sigma^*.$$

Clearly, L is a regular expression that uses concatenation and Kleene-star only.

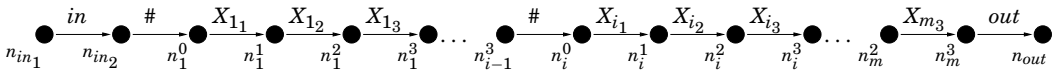
The reduction is as follows. Let $\phi = C_1 \wedge \cdots \wedge C_m$ be a formula in 3CNF using variables $\{x_1, \ldots, x_k\}$, and assume that for each $1 \le i \le m$ clause C_i is of form

$C_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$, where $1 \leq i_j \leq k$ for $j = 1, 2, 3$. With each variable x_ℓ ($1 \leq \ell \leq k$) we associate a different label variable X_ℓ. We construct a pattern π over Σ that uses variables $\{X_1, \ldots, X_\ell\}$ and node ids $\{n_{in_1}, n_{in_2}, n_{out}, \{n_i^j \mid 1 \leq i \leq m, 0 \leq j \leq 3\}\}$.

Moreover, π contains the following edges:

— it contains the edges (n_i^0, X_{i_1}, n_i^1), (n_i^1, X_{i_2}, n_i^2) and (n_i^2, X_{i_3}, n_i^3), for each $1 \leq i \leq m$;
— for each $1 < i \leq m$, π contains as well the edge $(n_{i-1}^3, \#, n_i^0)$; and
— finally, π contains the edges (n_{in_1}, in, n_{in_2}), $(n_{in_2}, \#, n_1^0)$, and (n_m^3, out, n_{out}).

Graphically, this pattern looks as follows:



Clearly, π belongs to $\mathcal{P}^{\text{lv}}$ and can be constructed in polynomial time from ϕ. Also, notice that the underlying graph of π is a path. Next we prove that there is a truth assignment to the variables of ϕ so that each clause has exactly one true variable if and only if $\text{CERTAIN}(Q, \pi) = \texttt{false}$.

($\Rightarrow$) Let $\gamma : \{x_1 \ldots, x_k\} \to \{0, 1\}$ be a truth assignment for the variables of ϕ so that γ assigns the value 1 to exactly one variable in each clause of ϕ. In order to prove that $\text{CERTAIN}(Q, \pi) = \texttt{false}$, we show the existence of a graph $G \in [\![\pi]\!]$ such that $Q(G) = \texttt{false}$.

To define G, we construct a mapping $\nu : \{X_1, \ldots, X_k\} \to \{\#, 0, 1, in, out\}$ as follows. For each $1 \leq \ell \leq k$, we set $\nu(X_\ell) = \gamma(x_\ell)$. Then, we define G as the graph resulting of replacing each variable X_i in $\{X_1, \ldots, X_\ell\}$ with $\nu(X_i)$.

We now prove that $Q(G) = \texttt{false}$. Assume for the sake of contradiction that this is not the case. That is, assume that there is a path ρ in G such that $\lambda(\rho)$ belongs to the language defined by L. Simply by construction of G, it is easy to check then that if $Q' := Ans() \leftarrow (n_{in_2}, L_1^* \cdot L_2^* \cdots L_{10}^*, n_m^3)$ then it must be the case that $G \models Q'$. Let ρ be the unique path from n_{in_2} into n_m^3 in G. Clearly, ρ is nonempty and, further, does not satisfy L_j, for each $1 \leq j \leq 5$. Thus, it must be the case that $\lambda(\rho)$ contains at least one subword in the set $\{\#111, \#011, \#101, \#110, \#000\}$, thus matching one of $\{L_6, \ldots, L_{10}\}$. We only derive a contradiction in the case when $\lambda(\rho)$ contains the subword #111, all other cases are completely symmetrical.

Assume then that #111 is a subword of $\lambda(\rho)$. In other words, we have that G contains a path ρ' such that $\lambda(\rho') = \#111$ (and, of course, that is a subpath of ρ).

Notice that, from the construction of π and ν, the only edges labeled with # are those of the form $(n_{i-1}^3, \#, n_i^0)$, for $1 < i \leq m$, and the edge $(n_{in_2}, \#, n_1^0)$.

Then, it must be the case that ρ' start in some node n_{i-1}^3 ($1 < i \leq m$), or in node n_{in_2}, and therefore (by the construction of G), ρ' uses edges $(n_{i-1}^3, \#, n_i^0)$, $(n_i^0, \nu(X_{i_1}), n_i^1)$, $(n_i^1, \nu(X_{i_2}), n_i^2)$ and $(n_i^2, \nu(X_{i_3}), n_i^3)$ (or starting with $(n_{in_2}, \#, n_1^0)$ if $i = 1$). Given that $\lambda(\rho')$ is #111, we have that $\nu(X_{i_1}) = \nu(X_{i_2}) = \nu(X_{i_3}) = 1$; by the construction of π, this means that there is a clause $C_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$, $1 \leq i \leq m$, such that $\gamma(x_{i_1}) = \gamma(x_{i_2}) = \gamma(x_{i_3}) = 1$, which contradicts the fact that γ assigns the value 1 to exactly one variable in each clause.

($\Leftarrow$): Assume now that $\text{CERTAIN}(Q, \pi) = \texttt{false}$. Then there must be a graph $G \in [\![\pi]\!]$ such that $Q(G) = \texttt{false}$. Since $G \in [\![\pi]\!]$ there is a homomorphism $h = (h_1, h_2)$ from π

into G, where h_1 maps nodes of π into nodes of G and h_2 maps the label variables of π into the alphabet $\{\#, 0, 1, in, out\}$.

Consider the path ρ in G defined as

$$n_{in_1}\, in\, n_{in_2}\, \#\, n_1^0\, h_2(X_{1_1})\, n_1^1\, h_2(X_{1_2})\, n_1^2\, h_2(X_{1_3})\, n_1^3\, \#\, n_2^0 \cdots \\ n_m^0\, h_2(X_{m_1})\, n_m^1\, h_2(X_{m_2})\, n_m^2\, h_2(X_{m_3})\, n_m^3\, out\, n_{out}.$$

Then, $\lambda(\rho)$ does not belong to L, which implies that if ρ' is the subpath of ρ that starts in n_{in_2} and finishes in n_m^3, then $\lambda(\rho')$ does not belong to the language given by $L_1^* \cdots L_{10}^*$. In particular, there is no subword of $\lambda(\rho')$ that satisfies L_j, for $1 \leq j \leq 5$. It can be easily checked that this implies that $h_2(X_\ell) \in \{0, 1\}$, for each $1 \leq \ell \leq k$.

Thus, from h_2, we define a valuation $\gamma : \{x_1, \ldots, x_k\} \to \{0, 1\}$ for the variables of ϕ as follows: For every $1 \leq \ell \leq k$, we let $\gamma(x_\ell) = h_2(X_\ell)$. We prove next that γ assigns the value 1 to exactly one variable in each clause of ϕ.

Assume for the sake of contradiction that γ does not satisfy this property. Then, there is a clause $C_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$, $1 \leq i \leq m$, such that γ does not assign the value 1 to exactly one of $\{x_{i_1}, x_{i_2}, x_{i_3}\}$. There are five symmetric cases, for each one of the possible valuations that do not satisfy this property. It is easy to derive a contradiction for each one of these cases, and we only show how to do it for the case when γ is such that $\gamma(x_{i_1}) = \gamma(x_{i_2}) = \gamma(x_{i_3}) = 1$. But then it is clear that $\lambda(\rho') \in L_6$, and, thus, $\lambda(\rho)$ belongs to L. This contradicts the fact that Q does not hold in G. □

The only possibility for a polynomial-time query answering algorithm left open by this result appears to be Codd patterns in $\mathcal{P}^{\text{lv}}$ with very nice underlying graphs. We shall see in Section 7, when we study tractable restrictions, that there is indeed a tractable class obtained along these lines.

The Role of Regular Expressions. In the case of patterns from $\mathcal{P}^{\text{re}}$, we have an additional parameter to vary: the regular expressions used in patterns. Nevertheless, we shall see that coNP-hardness is already witnessed by very simple regular expressions.

THEOREM 5.8.

— *There exists a Boolean RPQ Q of the form $Ans() \leftarrow (x, w, y)$, where w is a single word over $\Sigma = \{0, 1\}$, such that* DATA COMPLEXITY*(Q) is* coNP*-hard even over input patterns in $\mathcal{P}^{\text{re}}$ over Σ whose underlying graph is a DAG. It remains* coNP*-hard even if each regular expression used in input patterns is* $0|1$.

— *There exists a Boolean RPQ Q such that* DATA COMPLEXITY*(Q) is* coNP*-hard even over input patterns in $\mathcal{P}^{\text{re}}$ that only use regular expressions of the form a, for $a \in \Sigma$, or $a_1^* \ldots a_n^*$, where the a_i's are distinct letters in Σ.*

PROOF. The first part of the theorem follows directly from the second part of Theorem 5.7. This is because each pattern $\pi \in \mathcal{P}^{\text{lv}}_{\text{Codd}}$ over $\Sigma = \{0, 1\}$ is equivalent to the pattern $\pi' \in \mathcal{P}^{\text{re}}$ over Σ that is obtained from π by replacing each label variable X mentioned in π by the regular expression $(0|1)$ (i.e., $[\![\pi]\!] = [\![\pi']\!]$). Clearly, the underlying graphs of π and π' are the same.

For the second part, we use a reduction from non-3-colorability. Assume we have an arbitrary undirected graph G; we represent it as a labeled graph where between two nodes n_1 and n_2 connected by an edge we have two edges labeled a, that is, (n_1, a, n_2) and (n_2, a, n_1). Now we turn it into a $\mathcal{P}^{\text{re}}$ pattern π_G over the alphabet $\{a, r, g, b, d\}$ as follows. For each node n do the following: First, create a self-loop labeled on n labeled d. Second, add a new node n' and add edges $(n', (r^*g^*b^*), n)$ and $(n, (r^*g^*b^*), n')$, It can be shown that the certain answer to the Boolean RPQ

$Ans() \leftarrow (x, L, y)$ over π_G is true if and only if G is not 3-colorable, where L is the language $(rdb|rdg|gdb|gdr|bdg|bdr|gag|rar|bab)$. □

Like the case of patterns with label variables, this leaves open the possibility that more restrictive underlying graphs may lead to tractability. Indeed, we shall prove such results in Section 7.

6. INCOMPLETE AUTOMATA FOR QUERYING PATTERNS

Notice that graph databases can be viewed as finite automata. Graph patterns in turn can be viewed as *incomplete automata*. We now define those, and show that they naturally generate two notions of acceptance. These notions correspond to certain answers: one for certain answers as we defined them, and the other for certain answers for queries that can output paths.

Extensions of CRPQs outputting paths have been defined in Barceló et al. [2010a]. We shall present this notion for RPQs (for CRPQs, it includes the concept of synchronizing paths, which will complicate the presentation). An *RPQ with a path output* is a query of the form

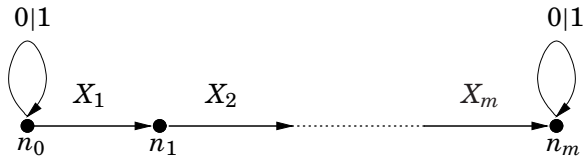
$$Ans(\bar{z}, \rho) \leftarrow (x, \rho : L, y),$$

where, on top of the usual RPQ $Ans(\bar{z}) \leftarrow (x, L, y)$, one is allowed to name the path ρ witnessing the query, and to output its label. Of course the number of L-paths between two nodes could be infinite, but one easily observes that for every nodes n_1, n_2 in a graph database, the set of labels of L-paths between them is regular, and thus can be represented by a finite automaton.

Assume we have an RPQ Q with a path variable, as described here, and a graph pattern π. Let n_1, n_2 be two nodes from $\mathcal{N}$ that occur in π. We say that a word $\rho \in \Sigma^*$ is a *certain path between n_1 and n_2 with respect to Q* if for every $G \in [\![\pi]\!]$, there is an L-path between n_1 and n_2 with label ρ. The set of such certain paths will be denoted by $\text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. We shall write $\text{CERTAIN}^{\text{path}}_{\Sigma}$ when Σ is not clear from the context.

The following example illustrates this concept.

Example 6.1. For $m > 0$, consider the pattern π_m over $\Sigma = \{0, 1\}$ shown here.



Notice that each $G \in [\![\pi_m]\!]$ will contain a path from node n_0 to node n_m. In particular, (n_0, n_m) is a certain answer to the RPQ Q given by $(x, \rho : (0|1)^*, y)$.

However, one can see that every word in $\text{CERTAIN}^{\text{path}}_{\Sigma}(Q; \pi, n_0, n_m)$ must contain, as subwords, all the 2^m words of length m over $\{0, 1\}$ since the X_i's can be instantiated arbitrarily. Due to the presence of the loops, the converse also holds, and $\text{CERTAIN}^{\text{path}}_{\Sigma}(Q; \pi, n_0, n_m)$ consists precisely of the words that contain all the 2^m subwords of length m. In particular, the smallest certain paths are precisely the noncircular De Bruijn sequences of order m, and thus have length $2^m + m - 1$. One can also easily show that any NFA accepting $\text{CERTAIN}^{\text{path}}_{\Sigma}(Q; \pi, n_0, n_m)$ will have exponentially many states (in m).

This example suggests that the problem of computing the certain paths is inherently different from the problem of computing certain answers for graph patterns, and thus we need to develop new tools for solving this problem. This is what we do next.

6.1. Incomplete Automata and Certain Answers

For convenience, we shall assume that NFAs can have edges labeled by words. That is, NFAs will be of the form $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$, where Q is the set of states, Σ is the alphabet, q_0 is the initial state, F is the set of final states, and the transition relation δ is a finite subset of $Q \times \Sigma^* \times Q$ (we impose finiteness to maintain equivalence with the standard notion of NFAs used in the literature). The notion of acceptance extends to such an automaton in the standard way: if there is a transition (q, w, q'), the automaton is in state q, then, if w is a subword that starts in the current position, the automaton skips it and moves to the state q'. When all the w's used in transitions are single letters, this is the standard notion of NFAs; in that case, we shall refer to them as *standard NFAs*.

The language accepted by an NFA is denoted by $L(\mathcal{A})$. Note that for each NFA, one can construct, in polynomial time, a standard NFA $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. This is done by converting each word in a transition into a DFA (in polynomial time) and plugging it in place of the transition. Hence, using extended transitions is indeed just a matter of convenience.

Definition 6.2 (*Incomplete Automata*). An incomplete automaton $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$, where $\mathcal{W}$ is a finite set of label variables from $\mathcal{V}_{\text{lab}}$, and $\delta \subseteq Q \times \text{REG}(\Sigma \cup \mathcal{W}) \times Q$.

Thus, an incomplete automaton is really just a graph pattern from $\mathcal{P}^{\text{nv,lv,re}}$ with a distinguished node corresponding to the initial state, and a set of nodes corresponding to the final states.

To define acceptance by these automata, we need the notion of *valuation*. For an incomplete automaton $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$, a valuation is a pair $\nu = (\eta, \theta)$, where $\eta : \mathcal{W} \to \Sigma$ maps label variables in $\mathcal{W}$ to Σ, and $\theta : (Q \times \text{REG}(\Sigma \cup \mathcal{W}) \times Q) \to (Q \times \Sigma^* \times Q)$ assigns to each transition $(q, L, q') \in \delta$ a transition (q, w, q'), where w is a word that belongs to $\eta(L)$. Thus, a valuation $\nu = (\eta, \theta)$ for an incomplete automaton $\mathcal{A}$ defines an NFA $\nu(\mathcal{A}) = (Q, \Sigma, q_0, F, \theta(\delta))$.

We now consider two notions of acceptance. *Weak acceptance* refers to those *languages* over Σ that are related to all valuations of an automaton, and *strong acceptance* defines *words* over Σ that are accepted by all these valuations.

Definition 6.3 (*Weak and strong acceptance*).

— A regular language $L \subseteq \Sigma^*$ is *weakly accepted* by an incomplete automaton $\mathcal{A}$ if $L \cap L(\nu(\mathcal{A})) \neq \emptyset$ for every valuation ν.
— A word $w \in \Sigma^*$ is *strongly accepted* by an incomplete automaton $\mathcal{A}$ if $w \in L(\nu(\mathcal{A}))$ for every valuation ν.

We write $\mathcal{L}_w(\mathcal{A})$ for the set of languages weakly accepted by $\mathcal{A}$, and $L_s(\mathcal{A})$ for the set of words strongly accepted by $\mathcal{A}$. Note that $\mathcal{L}_w(\mathcal{A}) \subseteq 2^{\Sigma^*}$ while $L_s(\mathcal{A}) \subseteq \Sigma^*$.

It is easy to see that for words the two previous notions can be related as follows: For each incomplete automaton $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, q_f, \delta)$ and word $u \in \Sigma^*$, it is the case that $u \in \mathcal{L}_s(\mathcal{A})$ if an only if $\{u\} \in \mathcal{L}_w(\mathcal{A})$. We prefer to continue talking, however, about both weak and strong acceptance since in this way we can clearly distinguish when we refer to acceptance of a word or of a regular language.

While not immediately obvious from the definition, we can show that languages strongly accepted by incomplete automata are regular.

PROPOSITION 6.4. *For an incomplete automaton $\mathcal{A}$, the language $L_s(\mathcal{A})$ of words strongly accepted by $\mathcal{A}$ is regular. An NFA accepting $L_s(\mathcal{A})$ can be constructed in doubly exponential time.*

PROOF. Let $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$ be an incomplete automaton. Assume that (q, L, q') is a transition in δ. We use the following technical but self-evident claim:

CLAIM 3. *The regular expression L defines a finite language over alphabet $\Sigma \cup \mathcal{W}$ if and only if $\eta(L)$ defines a finite language over alphabet Σ, for each mapping $\eta : \mathcal{W} \to \Sigma$.*

The key idea of the proof of the Proposition is the fact that, in terms of strong acceptance, we can dismiss all transitions in $\mathcal{A}$ that are labeled by expressions that define infinite languages. More precisely, let $\delta^{fin} \subseteq \delta$ be the set of transitions of form (q_1, L, q_2) such that L defines a finite language over alphabet $\Sigma \cup \mathcal{W}$. We denote by $\mathcal{A}^{fin}$ the automaton $(Q, \Sigma, \mathcal{W}, q_0, F, \delta^{fin})$ (notice that $\mathcal{A}^{fin}$ may contain variables in $\mathcal{W}$ that do not appear in any transition in δ^{fin}). The following lemma formalizes the idea just presented.

LEMMA 6.5. $L_s(\mathcal{A}) = L_s(\mathcal{A}^{fin})$.

PROOF. The fact that $L_s(\mathcal{A}^{fin}) \subseteq L_s(\mathcal{A})$ is straightforward. Thus, we only need to show that $L_s(\mathcal{A}) \subseteq L_s(\mathcal{A}^{fin})$. Assume that a word w belongs to $L_s(\mathcal{A})$. To prove that w belongs to $L_s(\mathcal{A}^{fin})$, we show next that for every valuation ν of $\mathcal{A}^{fin}$ it is the case that $\nu(\mathcal{A}^{fin})$ accepts w.

Let $\nu = (\eta, \theta)$ be an arbitrary valuation for $\mathcal{A}^{fin}$. Construct a valuation $\nu' = (\eta', \theta')$ for $\mathcal{A}$ as follows:

— valuation η' is a copy of η,
— define $\theta'((q_1, L, q_2)) = \theta((q_1, L, q_2))$, if (q_1, L, q_2) belongs to δ^{fin}, and
— otherwise, $\theta'((q_1, L, q_2)) = w'$, where w' is an arbitrary word in $\eta(L)$ such that $|w'| > |w|$ (We know that such word exists since L is an infinite language, and, therefore, from Claim 3, $\eta(L)$ is also an infinite language).

Since $w \in L_s(\mathcal{A})$ and ν' is a valuation for $\mathcal{A}$, the word w is accepted by $\nu'(\mathcal{A})$. Furthermore, notice that any accepting run ρ of w for $\nu'(\mathcal{A})$ is also an accepting run for $\nu(\mathcal{A}^{fin})$, as clearly ρ cannot use any transition labeled by a word of size larger than w. This shows that $\nu(\mathcal{A}^{fin})$ accepts w. Since ν is an arbitrary valuation for $\mathcal{A}^{fin}$, we conclude that $w \in L_s(\mathcal{A}^{fin})$, which finishes the proof of the lemma. □

We continue now with the proof of the proposition. Let $\mathcal{A}$ be an incomplete automaton. First, we prove that $L_s(\mathcal{A})$ is regular. By Lemma 6.5, we know that $L_s(\mathcal{A})$ can be defined as the intersection of all NFAs of the form $\nu(\mathcal{A}^{fin})$, where ν is a valuation for $\mathcal{A}^{fin}$. But notice that the set $\{\nu(\mathcal{A}^{fin}) \mid \nu \text{ is a valuation for } \mathcal{A}^{fin}\}$ is finite. This is because every edge in $\mathcal{A}^{fin}$ is labeled by an expression L that defines a finite language over alphabet $\Sigma \cup \mathcal{W}$, and, thus, from Claim 3, for each valuation $\eta : \mathcal{W} \to \Sigma$ its is the case that $\eta(L)$ also defines a finite language over Σ. The proof then follows from the fact that every finite intersection of regular languages is regular.

It remains to show that we can construct in double exponential time an NFA $\mathcal{B}$ such that $L(\mathcal{B}) = L_s(\mathcal{A})$. We have argued in the previous paragraph that $L_s(\mathcal{A})$ can be defined as the intersection of each automaton in the set $\{\nu(\mathcal{A}^{fin}) \mid \nu \text{ is a valuation for } \mathcal{A}^{fin}\}$. But notice that all of these automata are standard NFAs, so they can be intersected using the standard cross product construction. Thus, we just define $\mathcal{B}$ as

$\prod_{\nu} \nu(\mathcal{A}^{fin})$. That $\mathcal{B}$ can be constructed in double exponential time follows from the next claim, which can be easily proved using standard automata tools.

CLAIM 4. *Let r be a regular expression over an alphabet Σ, such that $L(r)$ is finite. Then, all words in $L(r)$ are of size at most $|r|$ (i.e., they have at most $|r|$ symbols). Furthermore, $L(r)$ contains at most $O(|\Sigma|^{|r|})$ words.*

From Claims 3 and 4, we immediately obtain that, for each valuation $\nu = (\eta, \theta)$ for $\mathcal{A}^{fin}$, it is the case that $\nu(\mathcal{A}^{fin})$ is of size polynomial with respect to $\mathcal{A}$. Let us now analyze the number of different valuations $\nu = (\eta, \theta)$ that can be defined for $\mathcal{A}^{fin}$. Clearly, we have $|\Sigma|^{|\mathcal{W}|}$ possible mappings η from $\mathcal{W}$ to Σ. For each of one of those mappings, different mappings θ can be constructed by mapping each edge (p, L, q) in δ^{fin} to different words in $\eta(L)$. By Claim 3, we have that $\eta(L)$ always defines a finite language, and thus, by Claim 4, the number of words in $\eta(L)$ is bounded by $O(|\Sigma|^{|L|})$ (recall that we assume that L is given as a regular expression). This clearly shows that the the number of different valuations that can be defined for $\mathcal{A}^{fin}$ is at most exponential in the size of $\mathcal{A}^{fin}$, and then $\mathcal{B} = \prod_{\nu} \nu(\mathcal{A}^{fin})$ is a product of exponentially many automata, each one of polynomial size. Using this observation, it is now easy to show that $\mathcal{B}$ can be constructed in double exponential time. □

We shall see later (Theorem 6.11) that the bound of Proposition 6.4 is tight.

Given a graph pattern $\pi = (N, E) \in \mathcal{P}^{\text{nv,lv,re}}$ over Σ that uses label variables $\mathcal{W}$, and two nodes n_1, n_2 from $\mathcal{N} \cap N$ (i.e., nodes which are not variables), we let $\mathcal{A}_\pi(n_1, n_2)$ be the incomplete automaton $(N, \Sigma, \mathcal{W}, n_1, \{n_2\}, E)$. The following theorem shows the relation between querying graph patterns and incomplete automata.

THEOREM 6.6. *Let $Ans(x, y, \rho) \leftarrow (x, \rho : L, y)$ be an RPQ, $\pi = (N, E)$ a graph pattern, and n_1, n_2 two of its nodes from $\mathcal{N}$. Then*

(1) $(n_1, n_2) \in \text{CERTAIN}(Q, \pi)$ *if and only if L is weakly accepted by $\mathcal{A}_\pi(n_1, n_2)$.*
(2) $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$ *if and only if $w \in L$ and w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$.*

PROOF. We only prove that $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$ if and only if $w \in L$ and w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$. The first part of the theorem, being similar, is left to the appendix.

($\Rightarrow$): Assume that $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. By definition, we have that w belongs to L. Thus, we only prove that w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$. Let $\nu = (\eta, \theta)$ be an arbitrary valuation for $\mathcal{A}_\pi(n_1, n_2)$; that is, η is a mapping from $\mathcal{W}$ into Σ and $\theta : (N \times \text{REG}(\Sigma \cup \mathcal{W}) \times N) \rightarrow (N \times \Sigma^* \times N)$ assigns to each edge $(p, r, q) \in E$ a transition (p, w, q), where w is a word that belongs to $\eta(r)$. Next we show that $w \in L(\nu(\mathcal{A}_\pi(n_1, n_2)))$.

Let σ be the assignment from the nodes of π into $\mathcal{N}$ that is the identity on node ids and maps each node variable x into a different node id n_x. Then, we define a graph database G as the unique (up to isomorphism) σ-canonical graph database[1] for π that satisfies the following: For every edge $e = (p, r, q)$ of π, the path ρ that is associated with e in G is such that $\lambda(\rho) = w$, where $\theta(e) = (p, w, q)$. Notice that G is, indeed, a σ-canonical assignment via η. This is because for each edge $e = (p, r, q)$ in E, it is the case that if $\theta(e) = (p, w, q)$ then $w \in \eta(r)$.

It is immediately clear that $G \in [\![\pi]\!]$ (since G is σ-canonical for π), and that $\sigma(n_1) = n_1$ and $\sigma(n_2) = n_2$. Furthermore, since $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$, there is a path

[1]For a definition, see the proof of Proposition 5.6.

ρ in G from n_1 to n_2 such that $\lambda(\rho) = w$. It is now easy to show that there is a run of $\nu(\mathcal{A}_\pi(n_1, n_2))$ that accepts w. This is because the transitions of $\nu(\mathcal{A}_\pi(n_1, n_2))$ are precisely the paths of G that are associated with the edges of π; that is, if (p, u, q) is a transition in $\nu(\mathcal{A}_\pi(n_1, n_2))$ then there is a path in G from p to q labeled u.

Thus, $w \in L(\nu(\mathcal{A}_\pi(n_1, n_2)))$. Since ν is an arbitrary valuation, we conclude that w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$.

($\Leftarrow$): Assume that $w \in L$ and that w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$. We prove that $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. Let G be an arbitrary graph in $[\![\pi]\!]$, and $h = (h_1, h_2)$ a homomorphism from π to G. Clearly, both node ids n_1 and n_2 belong to G.

Construct a valuation $\nu = (\eta, \theta)$ for $\mathcal{A}_\pi(n_1, n_2)$ as follows. Define $\eta(X) = h_2(X)$ for every variable X in $\mathcal{A}_\pi(n_1, n_2)$, and for every edge $e = (p, r, q)$ in E, nondeterministically choose a word $u \in L(r)$ such that there is a path from $h_1(p)$ to $h_1(q)$ in G that is labeled with u (we know there is at least one such word since $G \in [\![\pi]\!]$). Then, define $\theta(e) = (p, u, q)$. It is clear that $\nu = (\eta, \theta)$ is a valid valuation for $\mathcal{A}_\pi(n_1, n_2)$. Thus, since w is strongly accepted by $\mathcal{A}_\pi(n_1, n_2)$, we have that w is accepted by $\nu(\mathcal{A}_\pi(n_1, n_2))$. It is not hard to prove then that there is an L-path in G from n_1 to n_2 that is labeled w. This is because the transitions of $\nu(\mathcal{A}_\pi(n_1, n_2))$ are a subset of the paths of G that are associated with the edges of π; that is, if (p, u, q) is a transition in $\nu(\mathcal{A}_\pi(n_1, n_2))$, then there is a path in G from p to q labeled u. Since G is an arbitrary graph database in $[\![\pi]\!]$, and $w \in L$, we conclude that $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. This finishes the proof. □

The results in this section show that the query evaluation problem, for both nodes and paths, can be stated in purely automata-theoretic terms. In particular, the set $\text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$ is regular for every RPQ. Thus, our next goal is to study properties of incomplete automata.

6.2. Computational Problems for Incomplete Automata

Theorem 6.6 suggests studying computational problems for incomplete automata related to query evaluation. Results for weak acceptance have, in essence, been established earlier, so we are interested in strong acceptance, which accounts for having paths in the output.

For weak acceptance, *membership* (i.e., given incomplete automaton $\mathcal{A}$ and a regular language L, presented as a regular expression or as an NFA, does L belong to $\mathcal{L}_w(\mathcal{A})$?) is the problem of finding certain answers to RPQs. Hence, we have the following corollary.

COROLLARY 6.7. *The membership problem for incomplete automata under weak acceptance is* PSPACE*-complete, and* CONP*-complete if the language L is fixed.*

It can also be easily seen that the emptiness problem under weak acceptance, that is, whether $\mathcal{L}_w(\mathcal{A}) \neq \emptyset$, is solvable in polynomial time.

Now we address the case of strong acceptance, which, by Theorem 6.6, gives us complexity bounds for computing paths that are returned with certainty. There are the following three versions of the problem we consider.

— Checking whether the query output is not empty. In automata-theoretic terms, this is the *emptiness problem under strong acceptance*: given an incomplete automaton $\mathcal{A}$, check whether $L_s(\mathcal{A}) \neq \emptyset$.
— Checking whether a specific path belongs to the output, that is, whether $w \in \text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. In automata-theoretic terms, we are interested in the *membership problem under strong acceptance*, that is, given an incomplete automaton $\mathcal{A}$ and a word w, check whether $w \in L_s(\mathcal{A})$.

— Computing $\text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$. As this set is regular, in automata-theoretic terms, we study the following problem: For an incomplete automaton $\mathcal{A}$, construct an NFA $\mathcal{A}'$ so that $L(\mathcal{A}') = L_s(\mathcal{A})$.

As we analyze these problems, we shall see that hardness results will be witnessed by an especially simple kind of incomplete automata: namely, *wildcard automata*, in which all regular languages used in transitions are single letters (alphabet letters or variables). Formally, a wildcard automaton $\mathcal{A}$ is $(Q, \Sigma, \mathcal{W}, q_0, F, \delta)$, where $\delta \subseteq Q \times (\Sigma \cup \mathcal{W}) \times Q$.

We now show that problems related to computing certain paths are computationally hard as long as regular expressions or label variables are present in the edges.

The following does not appear to follow from known EXPSPACE-completeness results for graph databases [Calvanese et al. 2000b; Barceló et al. 2010a].

THEOREM 6.8. *The emptiness problem under strong acceptance is* EXPSPACE*-complete, and remains* EXPSPACE*-hard for wildcard automata.*

The upper bound is easy to obtain: From the proof of Proposition 6.4, it follows that $L_s(\mathcal{A})$, for each incomplete automata $\mathcal{A}$, can be represented as an intersection of exponentially many NFAs of polynomial size (measured, of course, in terms of the size of $\mathcal{A}$). Moreover, it is well known that checking nonemptiness of the intersection of a family of k NFAs of polynomial size can be solved in polynomial space in k. Given that k in this case is exponential in the size of $\mathcal{A}$, we get an EXPSPACE upper bound for the emptiness problem under strong acceptance.

The lower bound requires a new and quite involved proof. However, before proving this lower bound, we show that an already high lower bound – PSPACE-hardness – can be obtained with a relatively simple proof. The proof of this lower bound relies directly on the following lemma, which esentially reduces, in polynomial time, the problem of emptiness of the intersection of NFAs (which is known to be PSPACE-hard) to the problem of checking emptiness of a wildcard automaton under strong acceptance. We will also need this lemma later in order to prove the EXPSPACE lower bound of Theorem 6.8. In such case, we will actually need a stronger version of this result, as stated in Lemma 6.9, that shows that the problem of emptiness of the intersection of the languages accepted under strong acceptance by a set of wildcard automata can be reduced, in polynomial time, to the problem of checking emptiness of a single wildcard automaton under strong acceptance.

LEMMA 6.9. *Given a set $\{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ of wildcard automata over an alphabet Σ with at least two symbols, then one can construct in polynomial time a wildcard automaton $\mathcal{A}'$ over Σ such that $\bigcap_{j \leq k} L_s(\mathcal{A}_j) \neq \emptyset$ if and only if $L_s(\mathcal{A}') \neq \emptyset$.*

It is well known that deciding wether the intersection of a finite set of NFAs is nonempty is PSPACE-complete [Kozen 1977]. Thus, Lemma 6.9 reduced to the case when $\{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ is a set of standard NFAs, immediately gives us a PSPACE lower bound for the emptiness problem for wildcard automata under strong acceptance.

It is important to remark that Lemma 6.9 also shows a striking difference between incomplete automata and standard NFAs. Indeed, under the widely held complexity theoretic assumption that PTIME $\neq$ PSPACE, there is no efficient algorithm that, given a set $\{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ of NFAs, constructs an NFA $\mathcal{A}$ such that $L(\mathcal{A})$ is empty if and only if $\bigcap_{1 \leq i \leq k} L(\mathcal{A}_i)$ is empty.

PROOF OF LEMMA 6.9. Assume that each $\mathcal{A}_j$ $(1 \leq j \leq k)$ is of the form $\mathcal{A}_j := (Q_j, \Sigma, \mathcal{W}_j, q_0^j, F_j, \delta_j)$. We assume, without loss of generality, that the Q_j's are pairwise disjoint, and that the same is true for the $\mathcal{W}_j$'s.

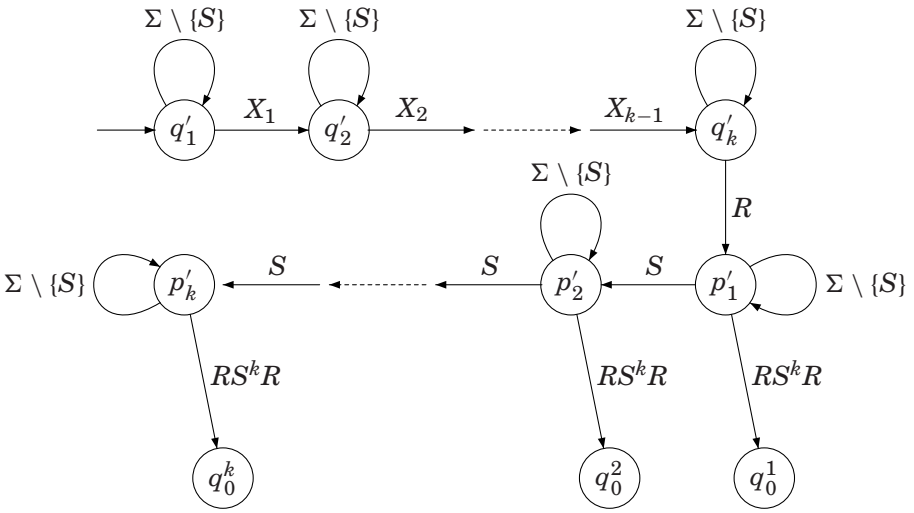


Fig. 6. Control section of wildcard automaton $\mathcal{A}'$.

Pick up two arbitrary symbols S and R from Σ. The wildcard automaton $\mathcal{A}'$ contains a copy of each $\mathcal{A}_j$ plus a *control* that helps simulating the intersection of the $\mathcal{A}_j$'s. Formally, we construct the wildcard automaton $\mathcal{A}' = (Q', \Sigma, \mathcal{W}', q'_0, F', \delta')$ as follows:

- The set Q' of states is $\{q'_1, \ldots, q'_k\} \cup \{p'_1, \ldots, p'_k\} \cup \bigcup_{j \leq k} Q_j$, where we assume that the states in $\{q'_1, \ldots, q'_k, p'_1, \ldots, p'_k\}$ are pairwise distinct and $\{q'_1, \ldots, q'_k, p'_1, \ldots, p'_k\} \cap \bigcup_{j \leq k} Q_j = \emptyset$;
- $F' = \bigcup_{j \leq k} F_j$;
- The initial state is q'_1;
- $\mathcal{W}' = \{X_1, \ldots, X_{k-1}\} \cup \bigcup_{j \leq k} \mathcal{W}_j$, where each X_i $(1 \leq i \leq k-1)$ is a fresh label variable;
- the set δ' of transitions contains the triples in each δ_j, $1 \leq j \leq k$, plus the following:
  - triples (q'_i, a, q'_i), for each $a \in \Sigma \setminus \{S\}$ and $i \in [1, k]$;
  - triples (p'_i, a, p'_i), for each $a \in \Sigma \setminus \{S\}$ and $i \in [1, k]$;
  - the triple (q'_k, R, p'_1);
  - triples (q'_i, X_i, q'_{i+1}), for every $i \in [1, k-1]$;
  - triples (p'_i, S, p'_{i+1}), for every $i \in [1, k-1]$;
  - and the triples (p'_j, RS^kR, q^j_0), for every $j \in [1, k]$

The *control* part of automaton $\mathcal{A}'$ is depicted in Figure 6. Notice that the states $q^1_0, \ldots, q^k_0$ are the initial states of automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$, respectively.

It is clear that $\mathcal{A}'$ can be constructed in polynomial time from $\{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$. We prove next that $\bigcap_{j \leq k} L_s(\mathcal{A}_j)$ is empty if and only if $L_s(\mathcal{A}')$ is empty.

($\Longrightarrow$): Assume that $\bigcap_{j \leq k} L_s(\mathcal{A}_j)$ is empty, and assume for the sake of contradiction that there is a word $w \in \Sigma^*$ such that w belongs to $L_s(\mathcal{A}')$. It is easy to see from the construction of $\mathcal{A}'$ that w must contain the word RS^kR as a subword. Then there are

words u, v in Σ^* such that $w = uRS^kRv$. We assume, without loss of generality, that u does not contain the word RS^kR (if not, one can always pick different words u and v).

Next we prove that the word u contains exactly $k-1$ appearances of S. Assume for the sake of contradiction that this is not the case. Then there are two cases to consider.

(1) First, suppose that u contains a number $p > k-1$ appearances of the symbol S. Let $\nu = (\eta, \theta)$ be a valuation for $\mathcal{A}'$, such that η does not assign the symbol S to any of the variables of $\mathcal{A}'$. It is now easy to see from the construction of $\mathcal{A}'$ that $\nu(\mathcal{A}')$ cannot accept w, as no state of $\nu(\mathcal{A}')$ can be reached from q'_1 using u: first, none of the states in $\{q'_1, \ldots, q'_k\}$ or $\{p'_1, \ldots, p'_k\}$ in $\nu(\mathcal{A}')$ can be reached from q'_1 with a word that contains more than $k-1$ appearances of the symbol S, and, second, the remaining states in $\mathcal{A}'$ can only be reached with a word containing the subword RS^kR. This is our desired contradiction since $w \in L_s(\mathcal{A}')$, and hence $w \in \nu(\mathcal{A}')$.
(2) On the other hand, if u contains a number $p < k-1$ appearances of S, consider a valuation $\nu = (\eta, \theta)$ such that η assigns an S to every label variable in $\mathcal{A}'$. Now notice that the state q'_k in $\nu(A')$ can only be reached by a word containing exactly $k-1$ appearances of S, and that it cannot be reached with a word containing RS^kR. It follows that $\nu(\mathcal{A}')$ cannot accept w, which is again a contradiction.

Thus, it must be the case that the word u contains exactly $k-1$ appearances of S.

We claim now that the word v belongs to $\bigcap_{j \le k} L_s(\mathcal{A}_j)$, which contradicts the fact that $\bigcap_{j \le k} L_s(\mathcal{A}_j)$ is empty.

Assume for the sake of contradiction that there exists $1 \le j \le k$ such that v does not belong to $L_s(\mathcal{A}_j)$. Then, there is a valuation $\nu = (\eta, \theta)$ for $\mathcal{A}_j$ such that $\nu(\mathcal{A}_j)$ does not accept the word v. Construct a valuation $\nu' = (\eta', \theta')$ for $\mathcal{A}'$ as follows: ν' extends ν by assigning values to the label variables in $\mathcal{W}' \setminus \mathcal{W}_j$ in the following way. It assigns symbol R to each label variable in $\{X_1, \ldots, X_j\}$, and symbol S to each variable in $\{X_{j+1}, \ldots, X_{k-1}\}$ and each variable in the sets $\mathcal{W}_i$, for $1 \le i \le k$ and $i \ne j$.

Recall that we assume, for the sake of contradiction, that $w \in L_s(\mathcal{A}')$, and thus w belongs to $L(\nu'(\mathcal{A}'))$. Fix an accepting run ρ for w over $\nu'(\mathcal{A}')$. Given that u has exactly $k-1$ appearances of S, by counting the transitions in $\nu'(\mathcal{A}')$ labeled with S we conclude that the run ρ can only lead to the state p_j after reading the word u. Then ρ must lead to state q_0^j after reading uRS^kR. Given that w is accepted by $\nu'(\mathcal{A}')$, and that valuation ν' is an extension of ν, it must be possible to reach a final state of $\nu(\mathcal{A}_j)$ using word v. This is a contradiction since we have assumed that v is not accepted by $\nu(\mathcal{A}_j)$.

($\Longleftarrow$): Assume that $L_s(\mathcal{A}')$ is empty, and assume for the sake of contradiction that there is a word $w \in \Sigma^*$ such that w belongs to $\bigcap_{j \le k} L_s(\mathcal{A}_j)$. Let $\bar{c}$ be a concatenation (in any order) of the symbols in $\Sigma \setminus \{S\}$. We prove below that $L_s(\mathcal{A}')$ contains the word $(\bar{c}^kSR)^{k-1}RS^kRw$, which is a contradiction.

Let $\nu = (\eta, \theta)$ be an arbitrary valuation for $\mathcal{A}'$. We show that $(\bar{c}^kRS)^{k-1}RS^kRw$ belongs to $\nu(\mathcal{A}')$. The proof depends on the number of label variables in $\{X_1, \ldots, X_{k-1}\}$ that are assigned value S by η. We only show two cases, the other ones are similar.

— Suppose that η does not assign the symbol S to any of the variables in $\{X_1, \ldots, X_{k-1}\}$. Then, clearly, it is possible to reach state q'_k in $\nu(\mathcal{A}')$ from q'_1 using word $\bar{c}^k$. Furthermore, state p'_2 is reachable from q'_k using word RS, and q_0^k is reachable from p'_2 using word $(\bar{c}^kRS)^{k-2}RS^kR$. Let ν^k be the restriction of ν over the variables of $\mathcal{A}_k$. Since w belongs to $L_s(\mathcal{A}_k)$, a final state of $\nu_k(\mathcal{A}_k)$ can be reached

from q_0^k using w. We conclude that a final state of $\nu(\mathcal{A}')$ can be reached from q'_1 using word $(\bar{c}^k RS)^{k-1} RS^k Rw$, and hence that $(\bar{c}^k RS)^{k-1} RS^k Rw$ belongs to $\nu(\mathcal{A}')$.

— Suppose that η assigns the symbol S to a single variable X_p, $1 \leq p \leq k-1$ (and, therefore, it assigns a symbol different from S to each X_j, for $1 \leq j \leq k-1$ and $j \neq p$). Then it is easy to see that state q'_p can be reached from q'_1 in $\nu(\mathcal{A}')$ using word $\bar{c}^k$, q'_{p+1} can be reached from q'_p in $\nu(\mathcal{A}')$ using word RS, q'_k is reachable from q'_{p+1} in $\nu(\mathcal{A}')$ using word $\bar{c}^k$, and p'_2 is reachable from q'_k using word RS. Furthermore, state q_0^{k-1} is reachable from p'_2 in $\nu(\mathcal{A}')$ using word $(\bar{c}^k RS)^{k-3} RS^k R$. Let ν^{k-1} be the restriction of ν over the variables of $\mathcal{A}_{k-1}$. Since w belongs to $L_s(\mathcal{A}_{k-1})$, a final state of $\nu_k(\mathcal{A}_{k-1})$ can be reached from q_0^{k-1} using w. We conclude that a final state of $\nu(\mathcal{A}')$ can be reached from q'_1 using word $(\bar{c}^k RS)^{k-1} RS^k Rw$, and hence that $(\bar{c}^k RS)^{k-1} RS^k Rw$ belongs to $\nu(\mathcal{A}')$.

This finishes the proof of the lemma since ν is an arbitrary valuation for $\mathcal{A}'$. □

We prove the EXPSPACE lower bound of Theorem 6.8, using a generic reduction from the acceptance problem for Turing machines that work in exponential space.

PROOF OF THEOREM 6.8. We have already explained how to obtain the upper bound. We prove the lower bound directly for wildcard automata. But before showing the proof, we need to define some auxiliary notation.

In order to keep the proof readable, we shall assume that NFAs can have edges labeled by regular expressions. That is, NFAs will be of the form $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$, where Q is the set of states, Σ is the alphabet, q_0 is the initial state, F is the set of final states, and the transition relation δ is a finite subset of $Q \times \text{REG}(\Sigma) \times Q$. The notion of acceptance extends to such an automaton in a standard way: if there is a transition (q, e, q'), the automaton is in state q, then, if there is a subword $w \in L(e)$ that starts in the current position, then the automaton skips it and moves to the state q'. Note that for each NFA $\mathcal{A}$ with transitions labeled by regular expressions, one can construct, in polynomial time, a standard NFA $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. This is done by converting each regular expression in a transition into an NFA (in polynomial time) and plugging it in place of the transition. Hence, using these extended transitions is indeed just a matter of convenience.

We shall also abuse the notation and define wildcard automata using transitions as triples in $Q \times \text{REG}(\Sigma \cup \mathcal{W}) \times Q$. But the semantics must not be confused with that of incomplete automata; on the contrary, these automata represent a set of NFAs as defined by valuations $\nu = (\eta, \theta)$ as follows (recall that we allow transitions in NFAs to be labeled over regular expressions). A valuation $\nu = (\eta, \theta)$ for an extended wildcard automata is such that θ maps each regular language $L \in \text{REG}(\Sigma \cup \mathcal{W})$ into $\eta(L) \in \text{REG}(\Sigma)$, and thus a wildcard automaton $\mathcal{A}$ into an NFA $\nu(\mathcal{A}) = (Q, \Sigma, q_0, q_f, \theta(\delta))$, where $(q, \eta(L), q') \in \theta(\delta)$ if and only if $(q, L, q') \in \delta$. The component θ of each valuation is, therefore, idle, and thus for the remainder of the proof we omit its definition, and just refer to valuations as mappings ν from label variables into symbols of the alphabet. Note that for each wildcard automata $\mathcal{A}$ with transitions labeled in $\text{REG}(\Sigma \cup \mathcal{W})$, one can construct, in polynomial time, a standard wildcard automata $\mathcal{A}'$, that is, with transitions labeled in $\Sigma \cup \mathcal{W}$, such that $\nu(\mathcal{A}) = \nu(\mathcal{A}')$, for each valuation ν for the label variables in $\mathcal{W}$. This is done, again, by converting each regular expression in $\text{REG}(\Sigma \cup \mathcal{W})$ in a transition into an NFA over $\Sigma \cup \mathcal{W}$ (in polynomial time), and plugging it in place of the transition. Hence, again, using these extended transitions is indeed just a matter of convenience.

We now start with the reduction for proving the EXPSPACE lower bound. Let $L \subseteq \Sigma^*$ be a language that belongs to EXPSPACE, and let $\mathcal{M}$ be a deterministic Turing machine that decides L in EXPSPACE. Given an input $\bar{a} \in \Sigma^*$ we construct in polynomial time with respect to $\mathcal{M}$ and $\bar{a}$ a wildcard automaton $\mathcal{A}_{\mathcal{M},\bar{a}}$ such that $L_s(\mathcal{A}_{\mathcal{M},\bar{a}}) \neq \emptyset$ if and only if $\mathcal{M}$ accepts $\bar{a}$. This proves that the emptiness problem under strong acceptance is EXPSPACE-complete for wildcard automata.

Assume that the Turing machine $\mathcal{M}$ is defined as $(Q^{\mathcal{M}}, \Sigma \cup \{B\}, s_0, \{s_m\}, \delta^{\mathcal{M}})$, where $Q^{\mathcal{M}} = \{s_0, \ldots, s_m\}$ is the set of states, $\Sigma \cup \{B\}$ is the tape alphabet, with B being the blank symbol, s_0 is the initial state, s_m is the unique final state, and $\delta : (Q^{\mathcal{M}} \setminus \{s_m\}) \times (\Sigma \cup B) \to Q^{\mathcal{M}} \times \Sigma \times \{L, R\}$ is the transition function. Notice that we assume without loss of generality that no transition is defined on the final state s_m. Further, we assume without loss of generality that $\Sigma = \{0, 1\}$. Since $\mathcal{M}$ decides L in EXPSPACE, there is a polynomial $S()$ such that, for every input $\bar{a}$ over Σ, $\mathcal{M}$ decides $\bar{a}$ using space of order $2^{S(|\bar{a}|)}$.

Let $\bar{a} = a_0 a_1 \cdots a_{k-1} \in \Sigma^*$ be an input for $\mathcal{M}$ (that is, each a_i, $0 \leq i \leq k-1$ is a symbol in Σ). For notational convenience, we will assume from now on that $S(|\bar{a}|) = n$. Due to Lemma 6.9, it suffices to construct in polynomial time from $\mathcal{M}$ and $\bar{a}$ a set $\mathbb{A}$ of wildcard automata such that $\bigcap_{\mathcal{A} \in \mathbb{A}} L_s(\mathcal{A})$ is nonempty if and only if $\mathcal{M}$ accepts on input $\bar{a}$. We split $\mathbb{A}$ into four groups: a single wildcard automaton $\mathcal{A}_1$, and three sets $\mathbb{A}_2$, $\mathbb{A}_3$ and $\mathbb{A}_4$ of NFAs.

The main idea of the reduction is to code each finite sequence of configurations of $\mathcal{M}$ as a word w, such that $\bigcap_{\mathcal{A} \in \mathbb{A}} L_s(\mathcal{A})$ is nonempty if and only if w represents an accepting computation of $\mathcal{M}$ on input $\bar{a}$ (i.e., $\mathcal{M}$ accepts $\bar{a}$). The key is that the information of the computation is not only directly coded into the word, but also into the runs of the NFAs in $\mathbb{A}$. This can be better explained as follows.

Although it will be formally defined later, the wildcard automaton $\mathcal{A}_1$ contains n label variables, $X_1, \ldots, X_n$. Furthermore, the alphabet of our automata is $\Sigma = \{0, 1\}$. Thus, the number of different valuations $\nu : \{X_1, \ldots, X_n\} \to \{0, 1\}$ for $\mathcal{A}_1$ is precisely 2^n; we shall enumerate them as $\nu_0, \ldots, \nu_{2^n-1}$, where for each $0 \leq i \leq 2^n - 1$, ν_i corresponds to the valuation $\nu_i : \{X_1, \ldots, X_n\} \to \{0, 1\}$ such that $\nu_i(X_1) \cdots \nu_i(X_n)$ is the n-symbol binary representation of the number i. For example, ν_0 corresponds to the valuation that maps each $X_1, \ldots, X_n$ to the symbol 0, while ν_{2^n-1} maps each variable to the symbol 1. Since $\mathcal{A}_1$ *represents* 2^n NFAs, one way of checking whether a word w belongs to the language $L_s(\mathcal{A}_1)$ is to perform a *parallel* computation of all 2^n NFAs represented by $\mathcal{A}_1$, reading w symbol by symbol, while keeping track of the state of each automata $\nu_i(\mathcal{A}_1)$, for $0 \leq i \leq 2^{n-1}$. We represent a parallel run of $\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1)$ by explicitly stating the sequence of arrays of states of size 2^n, of the form:

$$(q^1_{\nu_0(\mathcal{A}_1)}, \ldots, q^1_{\nu_{2^n-1}(\mathcal{A}_1)}), (q^2_{\nu_0(\mathcal{A}_1)}, \ldots, q^2_{\nu_{2^n-1}(\mathcal{A}_1)}), \ldots,$$

where each array represent the states of $(\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1))$ at a given step of the computation.

The states of $\mathcal{A}_1$ are defined as

$$\{q_a \mid a \in \Sigma \cup \{B\}\} \cup \{q_{a,s} \mid a \in \Sigma \cup \{B\}, s \in Q^{\mathcal{M}}\}.$$

Thus, a parallel run of $\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1)$ can be seen as representing a sequence of configurations of the Turing machine $\mathcal{M}$: On a given instance j of this run, for every $i \in \{0, 2^n - 1\}$, if $q^j_{\nu_i(\mathcal{A}_1)} = q_a$ ($a \in \Sigma_M$), then the ith cell of the tape of $\mathcal{M}$ contains symbol a at step j of the computation. Moreover, if $q^j_{\nu_i(\mathcal{A}_1)} = q_{a,s}$ for some state s of $\mathcal{M}$, then not only the ith cell of the tape contains symbol a at step j of the computation, but also the head of $\mathcal{M}$ is in position i at such step, and $\mathcal{M}$ is in state s.

Naturally, we need to enforce that these configurations represent a valid run of $\mathcal{M}$ with respect to $\bar{a}$ and δ (e.g., we need to enforce that at each step at most one of $\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1)$ is on a state of form $q_{a,s}$). This is achieved with the transitions of $\mathcal{A}_1$ and the NFAs in the sets $\mathbb{A}_2$, $\mathbb{A}_3$ and $\mathbb{A}_4$.

Next, we describe the sets $\mathbb{A}_2$, $\mathbb{A}_3$ and $\mathbb{A}_4$ of NFAs. In what follows, given a number i such that $0 \leq i \leq 2^n - 1$, we use the notation $[i]$ to denote its binary representation as a word of symbols in $\{0, 1\}^n$. For example, the word [3] corresponds to $0^{n-2}11$, and the word $[2^n - 2]$ corresponds to word $1^{n-1}0$.

($\mathbb{A}_2$): The set $\mathbb{A}_2$ consists of the following NFAs.

— The first NFA in $\mathbb{A}_2$ accepts precisely all the words accepted by the regular expression:

$$\big((0|1)^{n+3}\big)^*.$$

Thus, each word w in $\bigcap_{\mathcal{A}\in\mathbb{A}} L_s(\mathcal{A})$ must be of size $0 \mod (n+3)$. The idea is that we find useful to divide each such word into several consecutive subwords of size $n+3$. Given a word w, we say that a subword w' is an $(n+3)$-subword of w if the size of w' is $n+3$, w' is a subword of w and there is an integer $k = 0 \mod (n+3)$ such that there is a match for w' that starts from position k in w. For example, consider the word $w = 0101^n0001^n$. Then, w contains two $(n+3)$-subwords, namely the words 0101^n and 0001^n.

— The set $\mathbb{A}_2$ also contains the NFA that accepts the words represented by the expression

$$\big((000)(0|1)^n\big)^*\big((001|101|100|101)(0|1)^n\big)^*\big(111)(0|1)^n\big)^*.$$

This states that all $(n+3)$-subwords of every word w that belongs to $\bigcap_{\mathcal{A}\in\mathbb{A}} L_s(\mathcal{A})$ are such that their first three symbols are either 000, 001, 010, 100, 101, or 111, followed by an n-bit word. The first three symbols are used to code a particular behavior of the reduction, while the subsequent n-bit words are used to signal one (and only one) of the automata $\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1)$.

Further, all words represented by this NFA will be such that its first $(n+3)$-subword can only have 000 as a prefix, while the last one can only have prefix 111. Intuitively, all $(n+3)$-subwords with prefix 000 are related to the initial configuration of $\mathcal{M}$ on input $\bar{a}$, while all $(n+3)$-subwords with prefix 111 are related to the final configuration of $\mathcal{M}$.

— Next, set $\mathbb{A}_2$ also contains a set of NFAs whose intersection defines the language given by the expression

$$000[0] \cdot 000[1] \cdots 000[2^n - 2] \cdot 000[2^n - 1] \cdot \big((001|101|100|101|111)(0|1)^n\big)^*.$$

In other words, all words accepted by this language contain exactly 2^n $(n+3)$-subwords starting with the prefix 000, one for each different n-bit word, and ordered from [0] to $[2^n - 1]$.

— Finally, $\mathbb{A}_2$ contains an NFA accepting the language given by the expression:

$$\big((000|001|101|100|101)(0|1)^n\big)^*111(0|1)^n,$$

stating that there is only one $(n+3)$ subword of w that starts with the prefix 111.

($\mathbb{A}_3$): Next, we describe $\mathbb{A}_3$. This consists of a set of NFAs such that $\bigcap_{A\in\mathbb{A}_3} L(A)$ is the language described by the expression:

$$\big(000(0 \mid 1)^n\big)^* \Big(001(0 \mid 1)^n \cdot 001(0 \mid 1)^n \cdot (010 \mid 100 \mid 101)[0] \cdots (010 \mid 100 \mid 101)[2^n - 1]\Big)^* 111(0 \mid 1)^n.$$

Thus, any word w in $\bigcap_{A\in\mathbb{A}_3} L(A)$ can be decomposed as $v_i u_1 \cdots u_p v_f$, where v_i is the *initial part* of w, corresponding to a concatenation of words of the form $000(0|1)^n$, v_f is the *final part* of w, of the form $111(0|1)^n$, and each u_m, $1 \le m \le p$ is of the form $\big(001(0 \mid 1)^n 001(0 \mid 1)^n (010 \mid 100 \mid 101)[0] \cdots (010 \mid 100 \mid 101)[2^n - 1]\big)$.

Each of these words will intuitively represent one transition of $\mathcal{M}$. This will be better explained after we define $\mathcal{A}_1$; let us just say for the time being that we need to force each u_m to contain exactly one $(n+3)$-subword that ends with $[j]$, for each $0 \le j \le 2^n - 1$; the easiest way to impose this condition is to force each u_m to be a concatenation of words whose final n symbols form exactly the sequence $[0]\,[1] \cdots [2^n - 1]$.

($\mathbb{A}_4$): Finally, let L_j be the language represented by the following expression:

$$001(0 \mid 1)^n \cdot 001[j] \cdot \big(010(0 \mid 1)^n\big)^* \cdot (100[j-1] \mid 101[j+1]) \cdot \big(010(0 \mid 1)^n\big).$$

Then, $\mathbb{A}_4$ consists of a set of NFAs such that $\bigcap_{A\in\mathbb{A}_4} L(A)$ is the language described by the expression:

$$\big(000(0 \mid 1)^n\big)^* \cdot \Big(\Big(001(0 \mid 1)^n \cdot 001[0] \cdot \big(010(0 \mid 1)^n\big)^* \cdot 101[1] \cdot \big(010(0 \mid 1)^n\big)^*\Big) \mid L_1 \mid L_2 \mid \cdots \mid L_{2n-2} \mid \Big(001(0 \mid 1)^n \cdot 001[2^n - 1] \cdot \big(010(0 \mid 1)^n\big)^* \cdot 100[2^n - 2] \cdot \big(010(0 \mid 1)^n\big)^*\Big)\Big)^* \cdot 111(0 \mid 1)^n.$$

Notice that $\mathbb{A}_4$ intuitively imposes more conditions on the form of the words $u_1, \ldots, u_p$ mentioned before. That is, any word $w \in \bigcap_{A\in\mathbb{A}_3} L(A) \cap \bigcap_{A\in\mathbb{A}_4} L(A)$ is such that $w = v_i u_1 \cdots u_p v_f$, where v_i is a concatenation of words of the form $000(0 \mid 1)^n$, v_f is of the form $111(0 \mid 1)^n$, and each u_m, $1 \le m \le p$, is of the form $\big(001(0 \mid 1)^n 001[j]\,(010 \mid 100 \mid 101)[0] \cdots (010 \mid 100 \mid 101)[2^n - 1]\big)$, for some $0 \le j \le 2^n - 1$, but such that each u_m contains exactly one $(n+3)$-subword w starting with 100 or 101. Further, w is either of the form $100[j-1]$ or $101[j+1]$.

We now have enough ingredients to give some intuition on how the words $u_1, \ldots, u_p$ simulate a transition of $\mathcal{M}$. The first $(n+3)$ symbols on each u_m represent the state into which $\mathcal{M}$ enters after the transition, and the following $(n+3)$ symbols represent the position in the tape that the head of $\mathcal{M}$ is scanning. Furthermore, if u_m contains a $(n+3)$-subword of the form $101[j+1]$, it means that $\mathcal{M}$ is moving its head to the right (into the $j+1$-th position). On the other hand, the presence of a $(n+3)$-subword of the form $100[j-1]$ indicates a transition that moves the head to the left.

For example, consider that for some $1 \le m \le p$, the word u_m is of the form

$$001[\ell]^n \cdot 001[j] \cdot 010[0] \cdots 010[j] \cdot 101[j+1] \cdot 010[j+2] \cdots 010[2^n - 1]$$

for some $0 \leq \ell \leq m$ and $0 \leq j \leq 2^n - 1$. Then this word intuitively represents a transition in which the head of $\mathcal{M}$ is moved from the jth to the $(j+1)$-th position of the tape, that is, to the right, and $\mathcal{M}$ enters state s_ℓ.

We give now the formal definition of $\mathcal{A}_1 = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$.

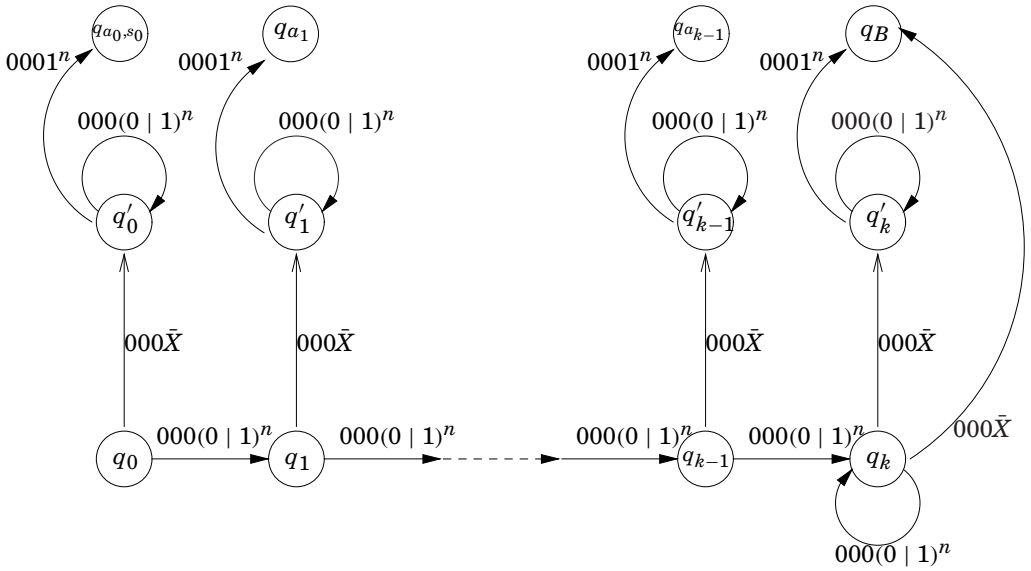
— The set Q of states of $\mathcal{A}_1$ contains state q_a, for each $a \in \Sigma \cup \{B\}$, plus state $q_{a,s}$ for each $a \in \Sigma \cup \{B\}$ and $s \in Q^{\mathcal{M}}$; plus states $q_f, q_0, \ldots, q_k$ and $q'_0, \ldots, q'_k$.
— The initial state is q_0.
— The unique final state in F is q_f.
— The set of label variables: $\mathcal{W}$ is $\{X_1, \ldots, X_n\}$.

In what follows, we denote by $\bar{X}$ the word $X_1 X_2 \cdots X_n$.

The transition function δ is defined next.

— First, we simulate the initial configuration of $\mathcal{M}$ with the following transitions: For each $0 \leq j < k-1$, add transition $(q_j, 000(0 \mid 1)^n, q_{j+1})$ to δ, and for each $0 \leq j \leq k$, add transitions $(q_j, 000\bar{X}, q'_j)$ and $(q'_j, 000(0 \mid 1)^n, q'_j)$ to δ.

In addition, add to δ the following transitions: $(q'_j, 0001^n, q_{a_j})$ for each $1 \leq j \leq k-1$, plus the triples $(q'_0, 0001^n, q_{a_0,s_0})$, $(q'_k, 0001^n, q_B)$, $(q_k, 0001^n, q_k)$, $(q'_k, 0001^n, q'_k)$ and $(q_k, 000\bar{X}, q_B)$. All these transitions are depicted in the following graph:



The idea is that, after reading word $000[0] \cdots 000[2^n - 1]$, each $\nu_i(\mathcal{A}_1)$ is in a state that corresponds to the initial configuration of $\mathcal{M}$; that is, $\nu_0(\mathcal{A}_1)$ is in state q_{a_0,s_0}, the NFA $\nu_1(\mathcal{A}_1), \cdots, \nu_{k-1}(\mathcal{A}_1)$ are in state $q_{a_1}, \ldots, q_{a_{k-1}}$, respectively, and the rest of the NFAs are in state q_B (recall that k is the size of the initial input).

— Next we define the transitions that simulate the final transition of $\mathcal{M}$. Add to δ the triples $(q_a, 111(0 \mid 1)^n, q_f)$, for every $a \in \Sigma$; and $(q_{a,s_m}, 111\bar{X}, q_f)$ for every $a \in \Sigma$ (recall that s_m is the final state of $\mathcal{M}$). The intuition behind these transitions is that they check that the head is on the final state at the end of the configuration. Indeed, since the only transition leaving from a state of the form q_{a,s_m} on input

$111[i]$ is $(q_{a,s_m}, 111\bar{X}, q_f)$, this ensures that it must be the case that $\nu_i(\mathcal{A}_1)$ is on state q_{a,s_m} before reading the last $(n+3)$ symbols of w.

— We continue with the transitions that simulate the run of $\mathcal{M}$. For each transition in $\delta^{\mathcal{M}}$ of the form $\delta(s_j, a) = (s_\ell, b, L)$ $(a \in \Sigma \cup \{B\}, b \in \Sigma, 0 \leq j, \ell \leq m)$, we add the following pair to δ:

$$(q_{a,s_j}, 001[\ell]\ 001\bar{X}\ (010(0 \mid 1)^n)^*\ 100(0 \mid 1)^n\ (010(0 \mid 1)^n)^*, q_b);$$

and for each transition in δ_M of the form $\delta(s_j, a) = (s_\ell, b, L)$ $(a \in \Sigma \cup \{B\}, b \in \Sigma, 0 \leq j, \ell \leq m)$, we add the following pair to δ:

$$(q_{a,s_j}, 001[\ell]\ 001\bar{X}\ (010(0 \mid 1)^n)^*\ 101(0 \mid 1)^n\ (010(0 \mid 1)^n)^*, q_b).$$

— Finally, for every $D \in \{L, R\}$, $a \in \Sigma \cup \{B\}$ and $0 \leq \ell \leq m$, δ contains the following pairs:

$(q_a, 001[\ell]\ 001(0 \mid 1)^n\ (010(0 \mid 1)^n)^*\ (100 \mid 101)\bar{X}\ (010(0 \mid 1)^n)^*, q_{a,s_\ell})$
$(q_a, 001[\ell]\ 001(0 \mid 1)^n\ ((010 \mid 100 \mid 101)(0 \mid 1)^n)^*\ 010\bar{X}\ ((010 \mid 100 \mid 101)(0 \mid 1)^n)^*, q_a)$.

We provide some intuitions regarding this last set of transitions in δ. Recall the first $(n+3)$ symbols of each word of the form

$$001(0 \mid 1)^n \cdot 001(0 \mid 1)^n \cdot (010 \mid 100 \mid 101)[0]\ \cdots\ (010 \mid 100 \mid 101)[2^n - 1]$$

represent the state in which $\mathcal{M}$ enters after a transition, and that the following $(n+3)$ symbols represent the position in the tape that the head of $\mathcal{M}$ is scanning. The idea of this set of transitions is to force each ν_i to behave correctly, according to the new configuration of $\mathcal{M}$ after performing a new computation.

For example, consider a word of the form

$$001[\ell]^n\, 001[j]\ 010[0] \cdots 010[j]\ 101[j+1]\ 010[j+2] \cdots 010[2^n - 1]\,,$$

for $0 \leq \ell \leq m$ and $0 \leq j \leq 2^n - 1$. As we already mentioned, this word intuitively represents a transition in which the head of $\mathcal{M}$ is moved from the jth to the $(j+1)$-th position of the tape, that is, to the right, and $\mathcal{M}$ enters state s_ℓ. But furthermore, assume that such transition is of the form $\delta(s_p, a) = (s_\ell, b, R)$, for $0 \leq p \leq m$, $a \in \Sigma \cup \{B\}$ and $b \in \Sigma$. Then the last set of transitions in δ allow $\nu_j(\mathcal{A}_1)$ to advance from state q_{a,s_p} to state q_b, and force $\nu_{j+1}(\mathcal{A}_1)$ to move from state of the form q'_a to state q_{a',s_ℓ}. The rest of the states are forced to remain unchanged.

This finishes the definition of $\mathbb{A}$. It is clear that $\mathcal{A}_1$ can be constructed in polynomial time from $\mathcal{M}$ and $\bar{a}$. Moreover, while not immediate from the definition, it is important to notice that also $\mathbb{A}_2$, $\mathbb{A}_3$ and $\mathbb{A}_4$ can be constructed in polynomial time.

CLAIM 5. *Each of the sets $\mathbb{A}_2$, $\mathbb{A}_3$, and $\mathbb{A}_4$ can be constructed in polynomial time with respect to $\bar{a}$ and $\mathcal{M}$.*

The proof for Claim 5 is esentially based on the idea that the language that consists of the word $w = [0] \cdot [1] \cdots [2^n - 1]$ can be represented as an intersection of a polynomial number of NFAs. This intersection expresses that for each $i \leq 2^n - 2$, the word $[i]$ has to be followed by the word $[i+1]$, and that the only occurrences of $[0]$ or $[2^n - 1]$ as an n-subword of w are respectively at the beginning and at the end of w. We skip the proof for the sake of presentation, since it is rather cumbersome and, at the same time, based on absolutely standard ideas on how to encode an n-bit counter [Börger et al. 1997; Calvanese et al. 2000b; Kozen 1977].

Finally, we prove that the language $L_s(\mathcal{A}_1) \cap \bigcap_{A \in \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4} L(A)$ is nonempty if and only if $\mathcal{M}$ accepts on input $\bar{a}$.

($\Longrightarrow$) Assume that there is a word w that belongs to $L_s(\mathcal{A}_1) \cap \bigcap_{A \in \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4} L(A)$. We show that $\mathcal{M}$ accepts in input $\bar{a}$ by constructing an accepting run for $\mathcal{M}$, as follows.

Since w belongs to $\bigcap_{A \in \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4} L(A)$, it must be of the form:

$$(000(0 \mid 1)^n)^* \left(001(0 \mid 1)^n 001(0 \mid 1)^n (010 \mid 100 \mid 101)[0] \cdots (010 \mid 100 \mid 101)[2^n - 1]\right)^* 111(0 \mid 1)^n.$$

Let us then divide w into $xu_1 \cdots u_p y$, where each u_i belongs to to the language represented by the expression

$$001(0 \mid 1)^n 001(0 \mid 1)^n (010 \mid 100 \mid 101)[0] \cdots (010 \mid 100 \mid 101)[2^n - 1].$$

It is clear that there is only one possible to split w in this way, as each u_i begins with two $(n+3)$-subwords with prefix 001 and contains no more $(n+3)$-subwords with this prefix.

Since w belongs to $L_s(\mathcal{A}_1)$ it must also belong to $\nu_j(\mathcal{A}_1)$, for each $0 \leq j \leq 2^n - 1$. Let $\rho_0, \ldots, \rho_{2^n-1}$ be accepting runs over the word w for $\nu_0(\mathcal{A}_1), \ldots, \nu_{2^n-1}(\mathcal{A}_1)$, respectively. We are interested in the states assigned by the runs $\rho_0, \ldots, \rho_{2^n-1}$ only to those positions of w that immediately precced one of the subwords of w of the form u_i, for $1 \leq i \leq p$. For simplicity, we denote each such state by $\rho_j(u_i)$; that is, $\rho_j(u_i)$ is the state that is assigned by ρ_j to the subword x, if $i = 1$, and to $xu_1 \cdots u_{i-1}$, if $1 < i \leq p$.

Consider the following sequence $d_1, \ldots, d_p$ of configurations of $\mathcal{M}$: The jth position of the tape in a given d_i contains a symbol $a \in \Sigma \cup \{B\}$ if and only if $\rho_j(u_i) = q_a$. Further, it contains a symbol $a \in \Sigma \cup \{B\}$ and the head is in position j in state s if and only if $\rho_j(u_i) = q_{a,s}$. In any other case, it contains the symbol B. All that is left to do is to show that each d_i is, indeed, a configuration of $\mathcal{M}$ (for instance, that the head is not assigned to two different positions of the tape by d_i), and that $d_1, \ldots, d_p$ represents an accepting computation of $\mathcal{M}$ on input $\bar{a}$. This follows from the following claim (the simple but rather technical proof can be found in the appendix).

CLAIM 6. *The following hold.*

(1) *The configuration d_1 corresponds the initial configuration of $\mathcal{M}$ on input $\bar{a}$.*
(2) *For each $1 \leq i < p$, d_{i+1} is a configuration of $\mathcal{M}$ that can be obtained from d_i by applying the transition rules of $\mathcal{M}$.*
(3) *The configuration d_p is a final configuration for $\mathcal{M}$.*

($\Longleftarrow$) Assume, on the other hand, that there is an accepting computation $d_1, \ldots, d_p$ of $\mathcal{M}$ on input $\bar{a}$. That is, $d_1, \ldots, d_p$ is a sequence of configurations of $\mathcal{M}$ such that: (1) d_1 is the initial configuration of $\mathcal{M}$ on input $\bar{a}$, (2) d_p is a final configuration for $\mathcal{M}$, and (3) for each $1 \leq i < p$, d_{i+1} is a configuration that can be obtained from d_i by applying the transition rules of $\mathcal{M}$. We show that $L_s(\mathcal{A}_1) \cap \bigcap_{A \in \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4} L(A)$ is nonempty.

Construct a word $w = xu_1 \cdots u_p y$ over Σ as follows.

— $x = 000[0] \cdots 000[2^n - 1]$.
— Assume that in the final configuration d_p the head of $\mathcal{M}$ is in position j ($1 \leq j \leq 2^n - 1$). Then, $y := 111[j]$.
— The word u_i, for $1 \leq i \leq p$, is constructed as follows. Assume that in d_i, $1 \leq i \leq p$, $\mathcal{M}$ is in state s_j, $0 \leq j \leq m$, and with its head pointing to the ℓth position of the tape ($0 \leq \ell \leq 2^n - 1$). Further, assume that in d_i the ℓ'-th cell of the tape contains the symbol $c_{\ell'} \in \Sigma \cup \{B\}$, for each $0 \leq \ell' \leq 2^n - 1$, and that d_{i+1} is obtained from d_i by

applying a transition of the form $\delta(s_j, a) = (s_{j'}, b, D)$, where $0 \leq j' \leq m$, $a \in \Sigma \cup \{B\}$, $b \in \Sigma$, and D is either L or R. Then u_i is:
— $001[\ell]\,001[j']\,010[0]\cdots010[\ell-2]\,100[l-1]\,010[\ell]\cdots010[2^n-1]$, if $D = L$, and
— $001[\ell]\,001[j']\,010[0]\cdots010[\ell]\,101[\ell+1]\,010[l+2]\cdots010[2^n-1]$, if $D = R$.

It is not difficult, but rather cumbersome, to show that w belongs to $L_s(\mathcal{A}_1) \cap \bigcap_{A\in\mathbb{A}_2,\mathbb{A}_3,\mathbb{A}_4} L(A)$. The proof can be found in the appendix. □

We now consider problems related to query answering. The first is finding certain paths, or, in automata-theoretic terms, the membership problem under strong acceptance.

PROPOSITION 6.10. *The membership problem under strong acceptance for incomplete automata is* coNP*-complete. It remains* coNP*-hard for wildcard automata and for incomplete automata that do not use any label variables.*

PROOF. We can easily prove coNP-hardness for wildcard automata as follows: The second part of Theorem 5.7 shows that there is a Boolean RPQ Q of the form $Ans() \leftarrow (x, w, y)$, where w is a word over $\{0, 1\}$, such that checking whether $\text{CERTAIN}(Q, \pi) = \texttt{true}$ is coNP-hard over the class of patterns $\pi \in \mathcal{P}^{\text{lv}}_{\text{Codd}}$. Let π be a pattern in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ and let π' be the pattern obtained from π by adding two fresh node ids n_1 and n_2, and adding an outgoing edge labeled 0 from n_1 into every node of π and an ingoing edge labeled 0 from each node of π into n_2. Clearly, $\mathcal{A}_{\pi'}(n_1, n_2)$ is a wildcard automaton. Further, by easily adapting the proof of Theorem 6.6, we can show that $\text{CERTAIN}(Q, \pi) = \texttt{true}$ if and only if $\{0w0\}$ is weakly accepted by $\mathcal{A}_{\pi'}(n_1, n_2)$. But the latter holds if and only if $0w0$ is strongly accepted by $\mathcal{A}_{\pi'}(n_1, n_2)$. Clearly, $\mathcal{A}_{\pi'}(n_1, n_2)$ can be constructed in polynomial time from π, which proves that the membership problem under strong acceptance for wildcard automata is coNP-hard (notice that this is true even for the *fixed* word $0w0$). In order to prove coNP-hardness for patterns without label variables, we only need to convert the pattern $\pi \in \mathcal{P}^{\text{lv}}_{\text{Codd}}$ into an equivalent pattern π^r in $\mathcal{P}^{\text{re}}$ by replacing each label variable X in π with the regular expression $(0|1)$. The rest of the proof uses exactly the same ideas explained above.

For the membership in coNP, let $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$ be an incomplete automaton. Recall that $\mathcal{A}^{fin}$ is the automaton $(Q, \Sigma, \mathcal{W}, q_0, F, \delta^{fin})$, where $\delta^{fin} \subseteq \delta$ is the set of transitions in δ of the form (q_1, L, q_2) such that L defines a finite language over alphabet $\Sigma \cup \mathcal{W}$. Moreover, recall that the proof of Proposition 6.4 shows that each NFA $\nu(\mathcal{A}^{fin})$ is of polynomial size with respect to $\mathcal{A}$, for every possible valuation $\nu = (\eta, \theta)$ for $\mathcal{A}^{fin}$. The same argument can be used to show that every valuation for $\mathcal{A}^{fin}$ can be easily represented in polynomial space with respect to $\mathcal{A}$.

Then, given an incomplete automaton $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$ and a word w, the coNP algorithm first constructs $\mathcal{A}^{fin}$ (which can be done in polynomial time as we only have to remove all those transitions of the form (p, L, q) in $\mathcal{A}$ such that L uses the Kleene-star $*$), then guesses in polynomial time a valuation $\nu = (\eta, \theta)$ for $\mathcal{A}^{fin}$, then checks in polynomial time that ν is a valuation for $\mathcal{A}^{fin}$, and finally, checks in polynomial time that $w \notin L(\nu(\mathcal{A}^{fin}))$. The correctness and soundness of this procedure follow immediately from Lemma 6.5. □

The next question is about the size of automata defining $L_s(\mathcal{A})$. Normally large size bounds are easy to obtain for deterministic automata, while NFAs could be exponentially smaller. Here we use techniques from Glaister and Shallit [1996] to show that even the smallest NFAs capturing certain paths in the answer to an RPQ could be doubly exponential, matching the upper bound of Proposition 6.4.

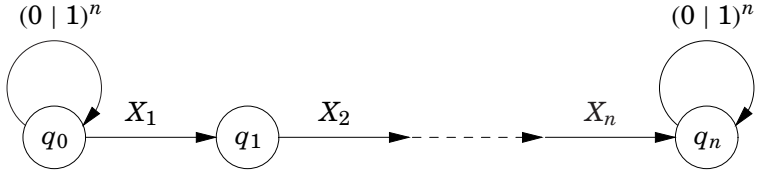
THEOREM 6.11. *There exists a polynomial p and a family $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of wildcard automata such each $\mathcal{A}_n$ is of size at most $p(n)$ and uses n wildcards, and every NFA $\mathcal{A}'_n$ satisfying $L(\mathcal{A}'_n) = L_s(\mathcal{A}_n)$ has $2^{2^{\Omega(n)}}$ states.*

There also exists a family of incomplete automata without label variables with the same property.

PROOF. Let $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ be the family of automata containing, for each $n \in \mathbb{N}$, the incomplete automata $\mathcal{A}_n = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$, where:

— $Q = \{q_0, q_1, \ldots, q_n, p_1, \ldots, p_n, r_1, \ldots, r_n\}$,
— $\Sigma = \{0, 1\}$ and $\mathcal{W} = \{X_1, \ldots, X_n\}$,
— F contains only q_n, and
— δ contains (q_{i-1}, X_i, q_i) for every $1 \leq i \leq n$, plus the triples $(q_0, 0, p_1)$, $(q_0, 1, p_1)$, $(q_n, 0, r_1)$, $(q_n, 0, r_1)$, and $(p_{j-1}, 0, p_j)$, $(p_{j-1}, 1, p_j)$, $(r_{j-1}, 0, r_j)$, $(r_{j-1}, 1, r_j)$ for each $1 \leq j \leq n$.

The automaton is depicted by the following figure, where the transitions labeled by $(0 \mid 1)^n$ represent the sections of the automata corresponding to states $p_1, \ldots, p_n$ and $r_1, \ldots, r_n$; the intended meaning of these sections of $\mathcal{A}_n$ is to allow the runs of $\mathcal{A}_n$ to do an n-symbol loop on states q_0 and q_n.



It is clear that the size of A_n is polynomial with respect to n. In order to show that every NFA $\mathcal{A}'_n$ satisfying $L(\mathcal{A}'_n) = L_s(\mathcal{A}_n)$ has $2^{2^{\Omega(n)}}$ states, we use the following result.

THEOREM 6.12 [GLAISTER AND SHALLIT 1996]. *Let $L \subset \Sigma^*$ be a regular language, and suppose that there exists a set of pairs $P = \{(u_i, v_i) \mid 1 \leq i \leq n\}$ such that*

(1) *$u_i v_i \in L$, for every $1 \leq i \leq n$,*
(2) *$u_j v_i \notin L$, for every $1 \leq i, j \leq n$ and $i \neq j$.*

Then, any NFA accepting L has at least n states.

Given a collection S of words over $\{0, 1\}$, let w_S denote the concatenation, in lexicographical order, of all the words that belong to S, and let $w_{\bar{S},n}$ denote the concatenation of all words in $\{0, 1\}^n$ that are not in S.

Then, define a set of pairs $P_n = \{(w_S, w_{\bar{S},n}) \mid S \subset \{0, 1\}^n \text{ and } |S| = 2^{n-1}\}$. Since there are 2^n binary words of length n, the are $\binom{2^n}{2^{n-1}}$ different subsets of $\{0, 1\}^n$ of size 2^{n-1}, and thus P_n contains $\binom{2^n}{2^{n-1}}$ pairs, which clearly belongs to $2^{2^{\Omega(n)}}$.

Next we show that P_n satisfies the conditions of (1) and (2) of the aforementioned result by Glaister and Shallit [1996].

(1) We need to show that for every set $S \subset \{0, 1\}^n$ of size 2^{n-1}, the word $w_S w_{\bar{S},n}$ belongs to $L_s(\mathcal{A}_n)$. Let then S be an arbitrary subset of $\{0, 1\}^n$ of size 2^{n-1}, and let $\nu = (\eta, \theta)$ be an arbitrary valuation for $\mathcal{A}_n$. Notice that η maps each element from $\{X_1, \ldots, X_n\}$ into Σ. Define $u = \eta(X_1) \cdots \eta(X_n)$. Then, u is a subword of either w_S or $w_{\bar{S},n}$. Assume the former is true (the other case is analogous). Then, the word

$w_S w_{\bar{S},n}$ can be decomposed in $v \cdot u \cdot v' \cdot w_{\bar{S},n}$, with $v', v'', w_{\bar{S},n} \in L((0 \mid 1)^n)^*)$. Then, to build an accepting run for $\nu(\mathcal{A}_n)$ on the word $w_S w_{\bar{S},n}$, just let the automaton $\nu(\mathcal{A}_n)$ loop on q_0 until the beginning of u appears, at which point we advance to state q_n using u, and then continue in q_n until the end of the word is reached.

(2) Assume for the sake of contradiction that there are distinct subsets S_1, S_2 of $\{0,1\}^n$ of size 2^{n-1} such that $w_{S_1} w_{\bar{S}_2,n}$ belongs to $L_s(\mathcal{A}_n)$. Since S_1 and S_2 are distinct, proper subsets of $\{0,1\}^n$ (they are of size 2^{n-1}), there must be a word in $\{0,1\}^n$ that belongs to S_2 but not to S_1. Let s be such word. Moreover, let η be a valuation from $\{X_1, \ldots, X_n\}$ into Σ such that $\eta(X_1) \cdots \eta(X_n) = s$. It is straightforward to show the following: Let $u \in \{0,1\}^n$ be a word of size n. Then, u is a subword of every $w \in L_s(\mathcal{A}_n)$. Moreover, there is a match for u in w that starts in a position j of w ($1 \leq j \leq |w|$), and such that $j = 0 \mod n$. Since we have assumed that the word $w_{S_1} w_{\bar{S}_2,n}$ belongs to $L_s(\mathcal{A}_n)$, by this claim we have that s must be a subword of $w_{S_1} w_{\bar{S}_2,n}$ that matches $w_{S_1} w_{\bar{S}_2,n}$ in a position j such that $j = 0 \mod n$. Then, from the construction of w_{S_1} and $w_{\bar{S}_2,n}$, it must be that s either belongs to S_1 or does not belong to S_2. This is a contradiction.

For the family of automata without label variables, notice that each $\mathcal{A}_n$ can be easily transformed into an equivalent incomplete automaton without label variables, by replacing each label variable in $\mathcal{A}_n$ with the regular expression $(0 \mid 1)$. The proof is analogous. □

This gives a lower bound on the size of automata for representing certain paths in answers to RPQs.

COROLLARY 6.13. *There exists a polynomial p, a family $\{\pi_n\}_{n \in \mathbb{N}}$ of $\mathcal{P}^{\text{lv}}$ graph patterns, each with two distinguished nodes n_1 and n_2, and an RPQ Q such that the size of π_n is at most $p(n)$, and every NFA defining* $\text{CERTAIN}^{\text{path}}(Q; \pi_n, n_1, n_2)$ *has $2^{2^{\Omega(n)}}$ states.*

The same holds for $\mathcal{P}^{\text{re}}$ patterns.

Note there is an exponential gap between the complexity of the membership problem and the size of a representation of all words strongly accepted by an incomplete automaton. There is no contradiction, of course, between Theorem 6.11 and Proposition 6.10 as smallest NFAs accepting even finite languages L can be of size exponential in the maximum length of a word in L.

Remark. By closely inspecting proofs, one notices that lower bounds in Theorem 6.11, Proposition 6.10, and Corollary 6.13 remain true even for the Codd interpretation of patterns and wildcard automata (i.e., each label variable is used in at most one transition). The same is true of Theorem 6.8, but at the cost of a much more involved reduction. For the sake of presentation, we have decided to leave it to the appendix.

7. TRACTABILITY RESTRICTIONS AND HEURISTICS

While many results of Sections 5 and 6 point to a rather high complexity of query answering, they still leave a few routes for finding tractable classes, or providing heuristics that – at least based on the experience of other areas – may be useful.

If we look at data complexity, results of Section 5.2 show that one possibility of getting tractable cases is to impose further restrictions on underlying graphs of patterns. Being DAGs, as we saw, is not enough, which suggests trees. We shall in fact get a more general result, replacing trees with graphs of bounded treewidth.

Combined complexity results in Section 5.1 point to $\mathcal{P}^{\text{nv,lv}}$ as the largest class with acceptable combined complexity (i.e., not exceeding that of FO; in fact staying in the 2nd level of the polynomial hierarchy). The data complexity for the class, although intractable, drops to the 1st level of the polynomial hierarchy. This suggests using techniques from a field that has achieved great success in solving problems of this complexity, namely constraint satisfaction [Dechter 2003; Kolaitis and Vardi 2007]. The field has identified many tractable restrictions and, what is equally important, provided many practical heuristics that help solve intractable problems. The connection between RPQs on graph databases and constraint satisfaction was already established in Calvanese et al. [2000c]. As the second contribution of this section, we show how to cast the query answering problem for RPQs over graph patterns as a constraint satisfaction problem, with a particularly simple translation for several classes.

7.1. Tractability Restrictions

Recall the standard definition of tree decompositions and treewidth of a graph $G = (N, E)$, with $E \subseteq N \times N$ (see, e.g., Diestel [2005]). A tree decomposition is a pair (T, f) where T is a tree and $f : T \to 2^N$ assigns to each node t in T a set of nodes $f(t)$ of G such that every edge of G is contained in one of the sets $f(t)$, and each set $\{t \mid n \in f(t)\}$ is a connected subset of T for all $n \in N$. The width of such a decomposition is $\max_t |f(t)| - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G. The treewidth of a connected graph G equals 1 if and only if G is a tree.

A class of graph patterns is of *bounded treewidth* if there is a fixed $k \in \mathbb{N}$ so that for every pattern π in the class, the treewidth of its underlying graph G_π is at most k.

We saw that label variables and regular expressions lead to intractable data complexity of query answering. We now show that bounded treewidth guarantees tractability for large classes of patterns with these features.

THEOREM 7.1. *The data complexity of finding certain answers to CRPQs over classes of graph patterns of bounded treewidth in* $\mathcal{P}^{\text{nv,re}}$ *and* $\mathcal{P}^{\text{nv,lv}}_{\text{Codd}}$ *is in* PTIME.

PROOF. It is sufficient to prove the theorem for the case of patterns in $\mathcal{P}^{\text{nv,re}}$. This is because each pattern $\pi \in \mathcal{P}^{\text{nv,lv}}$ that does not repeat label variables is equivalent to the pattern $\pi' \in \mathcal{P}^{\text{nv,re}}$ that is obtained from π by replacing each label variable X mentioned in π by the regular expression $\bigcup_{a \in \Sigma} a$ (i.e., $[\![\pi]\!] = [\![\pi']\!]$). Clearly, the underlying graphs of π and π' are the same.

We start by proving some auxiliary but necessary results. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two NFAs over the same alphabet Σ. Assume that the set of states of $\mathcal{A}_1$ is S, its transition function is given by $\delta : S \times \Sigma \to 2^S$, s_0 is the initial state, and $F \subseteq S$ is the set of final states. Let f be a function from S into 2^S, S' be a subset of S, and $\mathfrak{S}$ be a subset of 2^S. We say that the tuple $(f, S', \mathfrak{S})$ is *realized* in $\mathcal{A}_2$ whenever $\mathcal{A}_2$ accepts a word w such that (1) $\delta(\{s\}, w) = f(s)$, for each $s \in S$, (2) $S' \subseteq S$ consists of exactly those states s such that for some prefix w' of w, $\delta(\{s\}, w')$ contains at least one final state, and (3) $\mathfrak{S}$ consists of exactly those $S'' \subseteq S$ such that for some suffix w'' of w it is the case that $\delta(\{s_0\}, w'') = S''$. The following claim will be useful for the rest of the proof. The proof can be found in the appendix.

CLAIM 7. *Assume that the size of* $\mathcal{A}_1$ *is considered to be fixed. Then the set of tuples of the form* $(f, S', \mathfrak{S})$ *that are realized in* $\mathcal{A}_2$ *can be computed in polynomial time.*

Now we prove the proposition. In order to do this, we use the following idea. Given a pattern π in $\mathcal{P}^{\text{nv,re}}$, whose underlying undirected graph is of fixed treewidth, and a

fixed CRPQ Q (that we assume, without loss of generality, to be Boolean), we do the following.

— First, from π and Q, we construct in polynomial time a first-order structure $\mathcal{B}_{\pi,Q}$ over vocabulary σ (as defined here) such that the tree-width of $\mathcal{B}_{\pi,Q}$ is fixed.
— Second, from Q, we construct in constant time a sentence ϕ_Q in monadic second-order logic (MSO) over vocabulary σ such that $\text{CERTAIN}(Q,\pi) = \texttt{false}$ if and only if ϕ_Q holds in $\mathcal{B}_{\pi,Q}$.

It follows from Courcelle's theorem that the fixed MSO sentence ϕ_Q can be evaluated in polynomial-time over $\mathcal{B}_{\pi,Q}$ (since $\mathcal{B}_{\pi,Q}$ is of fixed tree-width). Since ϕ_Q can be constructed in constant time from Q, and $\mathcal{B}_{\pi,Q}$ can be constructed in polynomial time from π and Q, we conclude that there is a polynomial-time algorithm that evaluates fixed CRPQs over the class of patterns in $\mathcal{P}^{\text{nv,re}}$ such that its underlying undirected graph is of fixed treewidth.

Let π be a pattern in $\mathcal{P}^{\text{nv,re}}$ over Σ, such that its underlying undirected graph is of fixed treewidth $k > 0$, and let Q be a Boolean CRPQ. We assume that Q is an RPQ of the form (x,R,y), where R is a regular expression over Σ. We later explain how to extend the argument to arbitrary CRPQs with constants. This case, although much more cumbersome, uses essentially the same ideas that we use to solve the problem for RPQs.

We start by constructing an NFA $\mathcal{A}$ that is equivalent to R. Clearly, this can be done in constant time since Q itself is constant. Assume that the set of states of $\mathcal{A}$ is S, that its transition function is $\delta : S \times \Sigma \to 2^S$, the initial state is $s_0 \in S$, and $F \subseteq S$ contains the final states. Let $s_1, \ldots, s_p$ be an arbitrary enumeration of the states in S. Further, let $\mathcal{F}$ be the set of all functions $f : S \to 2^S$, let $W_1, \ldots, W_t$ an arbitrary enumeration of the elements in $\mathcal{F} \times 2^S \times 2^{2^S}$ (i.e., $t = |\mathcal{F}| \times 2^p \times 2^{2^p}$), and $Z_1, \ldots, Z_{2^t}$ an arbitrary enumeration of the subsets of $\mathcal{F} \times 2^S \times 2^{2^S}$. Then, we construct, for each e of the form (p,L,q) in π, the set $C_e \subseteq \mathcal{F} \times 2^S \times 2^{2^S}$ that contains exactly those tuples of the form $(f,S',\mathfrak{S})$, where $f : S \to 2^S$, $S' \subseteq S$, and $\mathfrak{S} \subseteq 2^S$, that are realized by the NFA that is equivalent to L. Using Claim 7, and the fact that for each regular expression an equivalent NFA can be constructed in polynomial time, one can easily prove that the set C_e can be constructed in polynomial time, for each edge e in π.

Construction of $\mathcal{B}_{\pi,Q}$. Now we show how to construct $\mathcal{B}_{\pi,Q}$ from π and Q. First, we define the vocabulary σ. This consists of a ternary relation *Edges* and unary predicates $U_1, \ldots, U_{2^t}$. Next, we define the domain of $\mathcal{B}_{\pi,Q}$. In order to do that, we associate, with each edge e in π a constant c_e that works as an identifier for e (i.e., if e and e' are distinct edges in π, then c_e and $c_{e'}$ are also distinct constants). Then, the domain of $\mathcal{B}_{\pi,Q}$ consists of each node p mentioned in π plus all constants of the form c_e such that e is an edge in π.

The interpretation of *Edges* in $\mathcal{B}_{\pi,Q}$ contains all tuples of the form (p,c_e,q) such that e is an edge from p to q in π. The interpretation of predicate U_i, $1 \leq i \leq 2^t$, contains exactly those constants of the form c_e such that $C_e = Z_i$. (Thus, the interpretations of $U_1, \ldots, U_{2^t}$ define a partition of the set of elements of the form c_e in $\mathcal{B}_{\pi,Q}$.) It easily follows from previous remarks that $\mathcal{B}_{\pi,Q}$ can be constructed in polynomial time from π and Q (recall that Q is fixed). The next claim proves that the treewidth of $\mathcal{B}_{\pi,Q}$ is fixed.

CLAIM 8. *The treewidth of* $\mathcal{B}_{\pi,Q}$ *is at most* $6k^2$.

PROOF. Since $\mathcal{B}_{\pi,Q}$ consists of several unary predicates and one ternary relation symbol, it is sufficient to prove that the restriction $\mathcal{B}'_{\pi,Q}$ of $\mathcal{B}_{\pi,Q}$ to the relation symbol *Edges* has treewidth bounded by $6k^2$. Take an arbitrary tree decomposition $(T, (B_t)_{t\in T})$, of the underlying undirected graph G of π, that witnesses that the treewidth of G is at most k. Recall that $(T, (B_t)_{t\in T})$ satisfies the following: (1) T is a tree; (2) Each B_t, $t \in T$, is a subset of the nodes in G, and every node of G belongs to at least some B_t, $t \in T$; (3) For every node p in G, the set $\{t \mid p \in B_t\}$ is connected; (4) If (p, q) is an edge of G, then, for some $t \in T$, it is the case that $\{p, q\} \subseteq B_t$; (5) $|B_t| \leq k + 1$, for each $t \in T$. From $(T, (B_t)_{t\in T})$, we construct the following tree decomposition of $\mathcal{B}'_{\pi,Q}$: For each edge (p, q) in G, we choose an arbitrary B_t, $t \in T$, that contains both p and q. Assume that there are exactly m edges $e_1, \ldots, e_m$ that go from p to q in π. Then, we replace t in T with a path of m new nodes $t_1, \ldots, t_m$, and define $B_{t_i} := B_t \cup \{c_{e_1}\}$, for each $1 \leq i \leq m$. It is not hard to see that the resulting tuple $(T', (B'_t)_{t\in T'})$ is a tree decomposition of $\mathcal{B}'_{\pi,Q}$, and that $|B'_t| \leq (k + 1) + (k + 1)^2 \leq 6k^2$, for each $t \in T'$. We conclude that the treewidth of $\mathcal{B}_{\pi,Q}$ is at most $6k^2$. □

Construction of ϕ_Q. The MSO formula ϕ_Q is defined as follows:

$$\phi_Q := \exists Y_1 \cdots \exists Y_t \left(\alpha(Y_1, \ldots, Y_t) \wedge \beta(Y_1, \ldots, Y_t) \wedge \neg\exists x \exists y \gamma(x, y, Y_1, \ldots, Y_t)\right),$$

where x and y are first-order variables and each Y_j $(1 \leq j \leq t)$ is a monadic second-order order variable. Intuitively, with ϕ_Q we try to "guess" a graph database in $[\![\pi]\!]$ that does not satisfy Q. This is done as explained here.

In the Y_j's, we try to guess an assignment (i.e., a graph database) that replaces each element of the form c_e in $\mathcal{B}_{\pi,Q}$ (i.e., each edge e in π) with a word w in the regular language L, assuming that the edge e is labeled with L in π. Notice, however, that it is impossible with the power of MSO to guess an entire word for an edge. Nevertheless, we do not need to guess all the information contained in w, and, indeed, for the sake of query answering with respect to Q, it is enough to guess only the tuple in $\mathcal{F} \times 2^S \times 2^{2^S}$ that is witnessed by w. This is precisely what formulas α and β do. Formula α guesses in the Y_j's the tuples in $\mathcal{F} \times 2^S \times 2^{2^S}$ that are witnessed by the words that replace edges in the graph database represented by π that we are trying to construct to falsify Q, and formula β checks, for each edge e, that such an assignment is consistent with the tuples in C_e (i.e., that we have guessed for c_e a tuple in $\mathcal{F} \times 2^S \times 2^{2^S}$ that is witnessed by L, assuming that L is the regular language that labels e in π). On the other hand, $\neg\exists x \exists y \gamma$ checks that Q does not hold in the graph database $G \in [\![\pi]\!]$ that is represented by the Y_j's; that is, G is any graph database that is obtained from π by replacing each edge e in π such that $c_e \in Y_j$ with a word w that realizes the tuple W_j in $\mathcal{F} \times 2^S \times 2^{2^S}$.

The formulas α, β and γ are defined as follows.

— Formula $\alpha(Y_1, \ldots, Y_t)$ establishes that the interpretations of $Y_1, \ldots, Y_t$ form a partition of the elements of the form c_e in $\mathcal{B}_{\pi,Q}$ (i.e., the elements that appear in the second coordinate of the interpretation of the relation *Edges* in $\mathcal{B}_{\pi,Q}$). Further, only elements of the form c_e belong to the interpretation of Y_j, for each $1 \leq j \leq t$. (Notice that elements of the form c_e are easily definable with the formula $\exists z_1 \exists z_3 \mathit{Edges}(z_1, z_2, z_3)$).

— Formula $\beta(Y_1, \ldots, Y_t)$ establishes that, for each edge e in π, if the constant c_e belongs to the interpretation of Y_j, $1 \leq j \leq t$, then the tuple $(f, S', \mathfrak{S})$ that corresponds to W_j belongs to C_e. This can be easily expressed by a formula that states that if an element y belongs to Y_j, $1 \leq j \leq t$, then it also belongs to the interpretation of some U_i, $1 \leq i \leq 2^t$, such that $W_j \in Z_i$.

— Assume that $W_j = (f^j, S^j, \mathfrak{S}^j)$, for $1 \le j \le t$. Let $X_1, \ldots, X_p$ be fresh monadic second-order variables and u_1, v_1, u_2, v_2 be fresh first-order variables. Then the formula $\gamma(x, y, Y_1, \ldots, Y_t)$ is defined as $\exists X_1 \cdots \exists X_p \theta$, where θ is the disjunction of the following formulas:
  — $\theta_1(x, y, Y_1, \ldots, Y_t, X_1, \ldots, X_p)$,
  — $\exists u \exists v\, \theta_2(x, y, u, v, Y_1, \ldots, Y_t, X_1, \ldots, X_p)$,
  — $\exists u \exists v\, \theta_3(x, y, u, v, Y_1, \ldots, Y_t, X_1, \ldots, X_p)$,
  — $\exists u_1 \exists v_1 \exists u_2 \exists v_2\, \theta_4(x, y, u_1, v_1, u_2, v_2, Y_1, \ldots, Y_t, X_1, \ldots, X_p)$,
and the MSO formulas θ_i, $1 \le i \le 4$, are as explained here.

First, for $S' \subseteq S$ and $s \in S$, we define an MSO formula $\mu_{s,S'}(x, y, X_1, \ldots, X_p, Y_1, \ldots, Y_t)$ that establishes the following.
  — The interpretations of $X_1, \ldots, X_p$ contain exactly the least fixpoints defined as follows: (1) x belongs to X_i, for each $1 \le i \le p$ such that $s_i \in S'$; (2) For each nodes z, z' and w, if (a) z belongs to the interpretation X_j, $1 \le j \le p$, (b) $Edges(z, w, z')$ holds, and (c) w belongs to Y_i, $1 \le i \le t$, then z' belongs to the interpretation of X_ℓ, for each $1 \le \ell \le p$ such that $s_\ell \in f^i(s_j)$.
  — The element y belongs to the interpretation of X_i, assuming that $s = s_i$.

It is standard, although rather cumbersome, to construct explicitly the MSO formula $\mu_{s,S'}(x, y, X_1, \ldots, X_p, Y_1, \ldots, Y_t)$. For the sake of readability, we omit it here. Intuitively, this formula checks the following on a pair of nodes x and y from $\mathcal{B}_{\pi,Q}$: If G is a graph database defined by the Y_j's (as described previously), then the X_i's contain exactly the nodes of $\mathcal{B}_{\pi,Q}$ (and, hence, of π) that are assigned state s_i by some "run" of $\mathcal{A}$ over the paths of G, that is initialized by assigning state s' to x, for each $s' \in S'$. This is done as follows: First, assign x to X_i for each $1 \le i \le p$ such that $s_i \in S'$. Then recursively proceed as follows. If node p of $\mathcal{B}_{\pi,Q}$ is assigned to X_i (i.e., state s_i of $\mathcal{A}$), there is an edge e from node p to q in π, and c_e belongs to the interpretation of Y_j (i.e., c_e has been replaced in G by a word that realizes, in particular, the function $f^j : S \to 2^S$), then q has to be assigned to each state $s_\ell \in f^j(s_i)$, that is, to the set X_ℓ. The formula $\mu_{s,S'}(x, y, X_1, \ldots, X_p, Y_1, \ldots, Y_t)$ checks, in addition, that y is assigned state s (i.e., that y belongs to X_i assuming that $s = s_i$).

Then we define the following.
  — $\theta_1 := \bigvee_{s' \in F} \mu_{s',\{s_0\}}(x, y, Y_1, \ldots, Y_\ell, X_1, \ldots, X_p)$.
  — $\theta_2 := \bigwedge_{1 \le j \le \ell} \big(Edges(u, v, x) \wedge Y_j(v) \rightarrow \bigvee_{S' \in \mathfrak{S}^j} \bigvee_{s' \in F} \mu_{s',S'}(x, y, Y_1, \ldots, Y_\ell, X_1, \ldots, X_p)\big)$.
  — $\theta_3 := \bigwedge_{1 \le j \le \ell} \big(Edges(y, v, u) \wedge Y_j(v) \rightarrow \bigvee_{s \in S^j} \mu_{s,\{s_0\}}(x, y, Y_1, \ldots, Y_\ell, X_1, \ldots, X_p)\big)$.
  — Formula θ_4 is:

$$Edges(u_1, v_1, x) \wedge Edges(y, v_2, u_2) \wedge \bigwedge_{1 \le j, \ell \le t} \big(Y_j(v_1) \wedge Y_\ell(v_2) \rightarrow \bigvee_{S' \in \mathfrak{S}^j, s \in S^\ell} \mu_{s,S'}(x, y, Y_1, \ldots, Y_t, X_1, \ldots, X_p)\big).$$

The meaning of tehse formulas will become clear when we prove the soundness and correctness of the construction of $\mathcal{B}_{\pi,Q}$ and π_Q (i.e., that $\textsc{certain}(Q, \pi) = \texttt{false}$ if and only if $\mathcal{B}_{\pi,Q} \models \phi_Q$).

Clearly, ϕ_Q can be constructed in constant time from Q. Next, we show that $\textsc{certain}(Q, \pi) = \texttt{false}$ if and only if $\mathcal{B}_{\pi,Q} \models \phi_Q$.

Soundness and Correctness. Assume first that $\mathcal{B}_{\pi,Q} \models \phi_Q$. This means that there exists a partition $P_1, \ldots, P_t$ of the elements of the form c_e that belong to $\mathcal{B}_{\pi,Q}$, such that $\mathcal{B}_{\pi,Q} \models \beta(P_1, \ldots, P_t) \wedge \neg \exists x \exists y \gamma(x, y, P_1, \ldots, P_t)$. Since $\mathcal{B}_{\pi,Q} \models \beta(P_1, \ldots, P_t)$, it is

the case that if an element of the form c_e belongs to P_i, $1 \leq i \leq t$, then the tuple $(f, S', \mathfrak{S})$ that corresponds to W_i belongs to C_e. With this in mind, we prove next that $\text{CERTAIN}(Q, \pi) = \texttt{false}$. In order to do that, we construct a graph $G \in [\![\pi]\!]$ such that $Q(G) = \texttt{false}$.

Let σ be an assignment from the nodes of π into the set $\mathcal{N}$ of node ids that (1) is the identity map on node ids, and (2) assigns a distinct node id n_x, that does not appear in π, to each node variable x. Then, the graph database G is obtained from π by replacing each node p by $\sigma(p)$, and then replacing each edge e of the form (p, L, q) with a path ρ_e of fresh node ids that goes from $\sigma(p)$ to $\sigma(q)$ that satisfies the following: Assume that c_e belongs to P_i, $1 \leq i \leq t$, and that $W_i = (f, S', \mathfrak{S})$. Then, $\lambda(\rho_e)$ is a word w that belongs to L and such that (1) $\delta(s, w) = f(s)$, for each $s \in S$, (2) S' is precisely the set of states s such that, for some prefix w' of w, it is the case that $\delta(s, w')$ contains at least one final state, and (3) $\mathfrak{S}$ consists of exactly those $S'' \subseteq S$ such that for some suffix w'' of w it is the case that $\delta(\{s_0\}, w'') = S''$. Notice that w exists since $\mathcal{B}_{\pi,Q} \models \beta(P_1, \ldots, P_t)$ and hence $(f, S', \mathfrak{S})$ is realized by the NFA $\mathcal{A}'$ that is equivalent to L. It is immediately clear then that $G \in [\![\pi]\!]$.

Now we prove that $Q(G) = \texttt{false}$. Assume, for the sake of contradiction, that there are two node ids n and n' in G such that there is a path ρ from n to n' that satisfies that $\lambda(\rho) \in R$. Notice that ρ is either of the form $\rho_1 \rho_{e_1} \rho_{e_2} \cdots \rho_{e_m} \rho_2$ or $\rho_{e_1} \rho_{e_2} \cdots \rho_{e_m} \rho_2$ or $\rho_1 \rho_{e_1} \rho_{e_2} \cdots \rho_{e_m}$ or $\rho_{e_1} \rho_{e_2} \cdots \rho_{e_m}$, where each ρ_{e_i}, $1 \leq i \leq m$, is the path associated with an edge e_i of π in G, ρ_1 is a suffix of the path ρ_{e_0} in G that is associated with an edge e_0 of π, and ρ_2 is a prefix of the path $\rho_{e_{m+1}}$ in G that is associated with an edge e_{m+1} of π. We assume in the following that ρ is of the form $\rho_1 \rho_{e_1} \rho_{e_2} \cdots \rho_{e_m} \rho_2$, all other cases being similar.

Assume that c_{e_0} belongs to Y_j, for $1 \leq j \leq t$, and that $W_j = (f^j, S^j, \mathfrak{S}^j)$. Thus, if $\delta(\{s_0\}, \lambda(\rho_1)) = S' \subseteq S$, then $S' \in \mathfrak{S}^j$. Further, assume that $c_{e_{m+1}}$ belongs to Y_ℓ, for $1 \leq \ell \leq t$, and that $W_\ell = (f^\ell, S^\ell, \mathfrak{S}^\ell)$. Thus, if $\delta(S', \lambda(\rho_{e_1} \rho_{e_2} \cdots \rho_{e_m})) = S'' \subseteq S$, then S'' contains at least some state s' in S^ℓ (otherwise, it would not be the case that $\delta(S'', \lambda(\rho_2))$ contains at least some state in F, and, thus, that $\lambda(\rho) \in R$). Further, it is clear that the following holds for each state $s \in S''$: Assume that $e_1 = (p_1, L_1, q_1)$ and that $e_m = (p_m, L_m, q_m)$. Also, assume that $U_1, \ldots, U_p$ contain exactly the least fixpoints defined as follows over the nodes of π: (1) p_1 belongs to U_i, for each $1 \leq i \leq p$ such that $s_i \in S'$; (2) For each nodes z, z' and w, if (a) z belongs to the interpretation U_j, $1 \leq j \leq p$, (b) $\mathit{Edges}(z, w, z')$ holds, (c) w belongs to P_i, $1 \leq i \leq t$, then z' belongs to the interpretation of U_ℓ, for each $1 \leq \ell \leq p$ such that $s_\ell \in f^i(s_j)$. Then, the node q_m belongs to the interpretation of U_i, assuming that $s = s_i$.

Assume that $e_i = (p_i, L_i, q_i)$, for each $1 \leq i \leq m + 1$. Then, clearly $\mathcal{B}_{\pi,Q} \models \mathit{Edges}(p_0, c_{e_0}, q_0) \wedge \mathit{Edges}(p_{m+1}, c_{e_{m+1}}, q_{m+1})$. Further, it is clear from the previous remarks that

$$\mathcal{B}_{\pi,Q} \models Y_j(c_{e_0}) \wedge Y_\ell(c_{e_{m+1}}) \wedge \mu_{s',S'}(p_1, q_m, P_1, \ldots, P_t, U_1, \ldots, U_p).$$

But then $\mathcal{B}_{\pi,Q} \models \exists x \exists y \gamma(x, y, P_1, \ldots, P_t)$, since $s' \in S_\ell$ and $S' \in \mathfrak{S}^j$, which is a contradiction.

Assume now that $\text{CERTAIN}(Q, \pi) = \texttt{false}$. Thus, from Claim 2, there is a graph database G in $[\![\pi]\!]$ such that G is σ-canonical for π and $Q(G) = \texttt{false}$. For each edge $e \in \pi$, let ρ_e be the path that is associated with e in G. We first construct a partition $P_1, \ldots, P_t$ for the elements of the form c_e in $\mathcal{B}_{\pi,Q}$ as follows: For each edge e in π, if the NFA that only accepts the word $\lambda(\rho_e)$ realizes the tuple W_i, then c_e belongs to P_i. We show next that $\mathcal{B}_{\pi,Q} \models \alpha(P_1, \ldots, P_t) \wedge \beta(P_1, \ldots, P_t) \wedge \neg \exists x \exists y \gamma(x, y, P_1, \ldots, P_t)$, which implies, in turn, that $\mathcal{B}_{\pi,Q} \models \phi_Q$.

Clearly, since G is canonical for π, $\mathcal{B}_{\pi,Q} \models \alpha(P_1,\ldots,P_t) \wedge \beta(P_1,\ldots,P_t)$. It just rests to show that $\beta_{\pi,Q} \models \neg\exists x\exists y\gamma(x,y,P_1,\ldots,P_t)$. Assume, on the contrary, that $\beta_{\pi,Q} \models \exists x\exists y\gamma(x,y,P_1,\ldots,P_t)$. In particular, assume that $\beta_{\pi,Q} \models \exists x\exists y\exists X_1,\ldots,X_p\exists u_1\exists v_1\exists u_2\exists v_2\,\theta_4(x,y,u_1,v_1,u_2,v_2,P_1,\ldots,P_t,X_1,\ldots,X_p)$, all other cases being similar.

Since $\beta_{\pi,Q} \models \exists x\exists y\exists X_1,\ldots,X_p\exists u_1\exists v_1\exists u_2\exists v_2\,\theta_4(x,y,u_1,v_1,u_2,v_2,P_1,\ldots,P_t,X_1,\ldots,X_p)$, there exist elements p, p', q, q', c_e and $c_{e'}$ in $\mathcal{B}_{\pi,Q}$ such that the following holds: (1) $Edges(p',c_e,p)$ and $Edges(q,c_{e'},q')$ holds in $\mathcal{B}_{\pi,Q}$; (2) If $c_e \in P_j$ and $c_{e'} \in P_\ell$, then it is the case that the following holds: Assume that $W_j = (f^j,S^j,\mathfrak{S}^j)$ and $W_\ell = (f^\ell,S^\ell,\mathfrak{S}^\ell)$. Then for some $S' \in \mathfrak{S}^j$ and $s \in S^\ell$ it is the case that $\exists X_1\cdots\exists X_p\mu_{s,S'}(p,q,P_1,\ldots,P_t,X_1,\ldots,X_p)$ holds in $\mathcal{B}_{\pi,Q}$. From the two previous facts, one can easily conclude the following: (1) There is a suffix ρ_1 of ρ_e such that $\delta(\{s_0\},\lambda(\rho_1)) = S'$; (2) There is a prefix ρ_2 of $\rho_{e'}$ such that $\delta(\{s\},\lambda(\rho_2))$ contains at least some final state; (3) There is a path ρ in G from $\sigma(p)$ to $\sigma(q)$ such that $\delta(S',\lambda(\rho))$ contains the state s. We conclude that $\rho_1\rho\rho_2$ is a path in G such that $\lambda(\rho) \in R$. This concludes this part of the proof.

Extension to Arbitrary CRPQs. A procedure that computes certain answers in polynomial time for arbitrary conjunctions of RPQs is more cumbersome to describe, but relies essentially on the same proof ideas. First of all, when constructing $\mathcal{B}_{\pi,Q}$ from π and Q, we have to be more careful, and provide in advance the necessary information to constants of the form c_e, in order to be able to recognize later when it is possible for a join between two node variables to occur in a node that belongs to a path that witnesses the edge e. In the same way, formula ϕ_Q has to be changed accordingly, in order to allow for this kind of joins to occur in the graph database. The addition of constants to queries only makes things easier, as then we precisely know where an element has to be witnessed in the graph database. □

The Codd interpretation of label variables is essential, since without it the problem is already coNP-hard for treewidth 1 (see Theorem 5.7). For $\mathcal{P}^{\mathrm{re}}$ patterns, coNP-hardness results of Theorem 5.8 used classes of DAGs of unbounded treewidth.

7.2. Certain Answers via Constraint Satisfaction

We now demonstrate the potential of using techniques from constraint satisfaction for answering queries over graph patterns, in the spirit of Calvanese et al. [2000c]. We shall concentrate on patterns in $\mathcal{P}^{\mathrm{nv,lv}}$, for which data complexity is in coNP. Of course, pure complexity-theoretic argument tells us that (the complement of) query answering can be cast as a constraint satisfaction problem; what we show here is that the translation for RPQs is very transparent, opening up the possibility of bringing the huge arsenal of tools from constraint satisfaction [Dechter 2003].

We adopt the standard view of the constraint satisfaction problem (CSP) as checking for the existence of a homomorphism from a relational structure $\mathcal{M}_1$ to another structure $\mathcal{M}_2$ of the same vocabulary [Kolaitis and Vardi 2007], referring to this problem as CSP($\mathcal{M}_1,\mathcal{M}_2$). Often this problem is considered with $\mathcal{M}_2$ fixed; in which case, one refers to nonuniform CSP.

Consider a pattern $\pi = (N,E)$ in $\mathcal{P}^{\mathrm{nv,lv}}$, that is, $E \subseteq N \times (\Sigma \cup \mathcal{W}) \times N$ for a finite set $\mathcal{W}$ of label variables. Let Q be an RPQ given by $Ans(x,y) \leftarrow (x,L,y)$, where $L \subseteq \Sigma^*$ is a regular language. We now define logical structures $\mathcal{M}_\pi(n,n')$ and $\mathcal{M}_Q$ over vocabulary

$$(Nodes, Expr, (Lab_a)_{a\in\Sigma}, Src, Sink, Edge),$$

where $Edge$ is a ternary relation and other relations are unary. Here, n and n' are two node ids of π.

Structure $\mathcal{M}_\pi(n,n')$. The domain is the disjoint union of N, Σ, and $\mathcal{W}$, the set of label variables used in π. The interpretation of the predicates is as follows.

$$\begin{array}{rlrl} \mathit{Nodes} & := N & \mathit{Edge} & := E \\ \mathit{Lab}_a & := \{a\} & \mathit{Src} & := \{n\} \\ \mathit{Expr} & := \mathcal{W} & \mathit{Sink} & := \{n'\} \end{array}$$

Structure $\mathcal{M}_Q$. Assume that L is recognized by an NFA (S,Σ,q_0,F,δ) with $\delta : S\times\Sigma \to 2^S$ (extended, as usual, to a transition function on sets $\delta(S',a)=\bigcup_{s\in S'}\delta(s,a)$). The domain of $\mathcal{M}_Q$ is the disjoint union of 2^S and Σ. The interpretation of the predicates is the following.

$$\begin{array}{ll} \mathit{Nodes}{:=}2^S & \mathit{Edge}{:=}\{(S',a,S'') \in 2^S\times\Sigma\times 2^S \mid \delta(S',a)\subseteq S''\} \\ \mathit{Lab}_a{:=}\{a\} & \mathit{Src}{:=}\{S'\in 2^S \mid q_0\in S'\} \\ \mathit{Expr}{:=}\Sigma & \mathit{Sink}{:=}2^{S-F} \end{array}$$

THEOREM 7.2. *For patterns* $\pi \in \mathcal{P}^{\mathrm{nv,lv}}$, *under these translations,* $(n,n') \in \mathrm{CERTAIN}(Q,\pi)$ *if and only if there is no solution to* $\mathrm{CSP}(\mathcal{M}_\pi(n,n'),\mathcal{M}_Q)$.

PROOF. Assume first that $(n,n') \notin \mathrm{CERTAIN}(Q,\pi)$. Then, there is a graph database G over Σ such that $G\in[\![\pi]\!]$ but $(n,n')\notin Q(G)$. Since $G\in[\![\pi]\!]$, there exists a homomorphism $h : (h_1,h_2)$ from π into G, where h_1 maps nodes of π into nodes of G, and h_2 maps label variables used in π into symbols from Σ.

Let $\mathcal{A}=(S,\Sigma,q_0,F,\delta)$ be the NFA that recognizes L, where we assume, without loss of generality, that $\delta(q,a)$ is defined, for each $q\in S$ and $a\in\Sigma$. Further, let $\mathcal{A}'$ be the NFA $\mathcal{A}\times G$. Recall that $\pi=(N,E)$ and that $\mathcal{W}$ is the set of label variables used in π. Then, let $f : N \to 2^S$ be the mapping defined as $f(p)=S'$, where S' is the subset of S that consists of exactly those states q such that there is a run of $\mathcal{A}'$ from state (q_0,n) to state $(q,h_1(p))$. Further, let f' be the mapping from the domain of $\mathcal{M}_\pi(n,n')$ into the domain of $\mathcal{M}_Q$ that is defined as follows:

— For each $p\in\mathcal{M}_\pi(n,n')\cap N$, it is the case that $f'(p)=f(p)$;
— For each $a\in\mathcal{M}_\pi(n,n')\cap\Sigma$, it is the case that $f'(a)=a$;
— For each $X\in\mathcal{M}_\pi(n,n')\cap\mathcal{W}$, it is the case that $f'(X)=h_2(X)$.

We prove next that f' is a homomorphism from $\mathcal{M}_\pi(n,n')$ into $\mathcal{M}_Q$.

Clearly, for each element c in the domain of $\mathcal{M}_\pi(n,n')$, it is the case that $c\in T \Rightarrow f'(c)\in T$, for each $T\in\{\mathit{Nodes},\mathit{Expr},(\mathit{Lab}_a)_{a\in\Sigma}\}$. Further, it is clear from the definition of f' and f, that $f'(n)$ contains the state q_0, and thus, that for each c in the domain of $\mathcal{M}_\pi(n,n')$ it is the case that $c\in\mathit{Source}\Rightarrow f'(c)\in\mathit{Source}$. Moreover, since $(n,n')\notin Q(G)$, there is no run of $\mathcal{A}'$ from (q_0,n) to a state (q,n') such that $q\in F$. Thus, $f'(n')=f(n')$ satisfies that $f'(n')\cap F=\emptyset$, and, therefore, we can conclude that for each c in the domain of $\mathcal{M}_\pi(n,n')$ it is the case that $c\in\mathit{Sink}\Rightarrow f'(c)\in\mathit{Sink}$.

It just remains to show that, for each triple of the form (p,D,q), where $p,q\in N$ and $D\in\Sigma\cup\mathcal{W}$, it is the case that $(p,D,p')\in\mathit{Edges}\Rightarrow(f'(p),f'(D),f'(p'))\in\mathit{Edges}$. Assume that $(p,D,p')\in\mathit{Edges}$. Consider an arbitrary state $q\in f'(p)$. Then there exists a run of $\mathcal{A}'$ from state (q_0,n) to state $(q,h_1(p))$. Since (p,D,p') is an edge of π, it must be the case that $(h_1(p),h_2(D),h_1(p'))$ is an edge of G. Thus, there is a run of $\mathcal{A}'$ from state (q_0,n) to state $(\delta(q,h_2(D)),h_1(p'))$. (We assume $h_2(D)=D$ if $D\in\Sigma$.) Since q was arbitrarily chosen in $f'(p)$, we conclude that $\bigcup_{q\in f'(p)}\delta(q,f'(D))\subseteq f'(p')$, and, therefore, that $(f'(p),f'(D),f'(p'))\in\mathit{Edges}$.

We conclude that there is a solution for $\mathrm{CSP}(\mathcal{M}_\pi(n,n'),\mathcal{M}_Q)$.

Assume, on the other hand, that there is a solution for CSP($\mathcal{M}_\pi(n,n'), \mathcal{M}_Q$). Thus, there is a homomorphism f from $\mathcal{M}_\pi(n,n')$ into $\mathcal{M}_Q$. We define G as the graph database over Σ that can be obtained from π by replacing each node variable x with a fresh node id n_x, and each label variable $X \in \mathcal{W}$ with the symbol $f(X) \in \Sigma$. (Notice that $f(X)$ is, indeed, a symbol in Σ, since f is a homomorphism from $\mathcal{M}_\pi(n,n')$ into $\mathcal{M}_Q$.) It is clear that $G \in [\![\pi]\!]$. We prove next that $(n,n') \notin Q(G)$.

Assume that the set of node ids mentioned in G is $N' \supseteq N$. Consider again the NFA $\mathcal{A}' := \mathcal{A} \times G$. Define a function $f' : N' \to 2^S$ such that for each $n_0 \in N'$, $f'(n_0)$ is the subset S' of S that consists of exactly those states q such that there is a run of $\mathcal{A}'$ from state (q_0,n) to state (q,n_0). We claim that $f'(n') \cap F = \emptyset$, which implies that $(n,n') \notin Q(G)$.

First of all, we prove that $f'(n_0) \subseteq f(n_0)$, for each $n_0 \in N'$. Assume, for the sake of contradiction, that for some $n_0 \in N'$ there is a state $q \in f'(n_0)$ such that $q \notin f(n_0)$. Since $q \in f'(n_0)$, there is a run of $\mathcal{A}'$ that is of the form

$$(q_0,n)(q_1,n_1)\cdots(q_t,n_t)(q,n_0)$$

on some word $a_1a_2a_t\cdots a_{t+1}$ over Σ. But since $q_0 \in f(n)$, it must be the case that $q_j \in f(n_j)$, for each $1 \le j \le t$. This is because f is a homomorphism from $\mathcal{M}_\pi(n,n')$ into $\mathcal{M}_Q$, and, thus, $\bigcup_{q' \in f(n)} \delta(q', a_1) \subseteq f(n_1)$ and $\bigcup_{q' \in f(n_j)} \delta(q', a_{j+1}) \subseteq f(n_{j+1})$, for each $0 \le j < t$. For the same reason, $q \in f(n_0)$, which is a contradiction.

Notice that $f(n') \cap F = \emptyset$ (since f is a homomorphism from $\mathcal{M}_\pi(n,n')$ into $\mathcal{M}_Q$), and hence $f'(n') \cap F = \emptyset$ (this is because we have just proved that $f'(n') \subseteq f(n')$). The latter implies that $(n,n') \notin Q(G)$. Further, since $G \in [\![\pi]\!]$, we conclude that $(n,n') \notin \text{CERTAIN}(Q,\pi)$. □

Many algorithmic techniques for constraint satisfaction for CSP($\mathcal{M}_1, \mathcal{M}_2$) are based on exploiting properties of the structure $\mathcal{M}_1$, so the extremely simple construction of $\mathcal{M}_\pi(n,n')$ indeed opens up the possibility of using a large body of heuristics developed in that area.

The case of data complexity corresponds to the nonuniform version of CSP, with $\mathcal{M}_Q$ fixed. In that case one can immediately conclude (using known results on CSP [Dechter 2003; Kolaitis and Vardi 2007]) that if we have a class of patterns $\pi \in \mathcal{P}^{\text{nv,lv}}$ which, when viewed as ternary relations E, has bounded treewidth, then the data complexity of RPQs over such a class is in PTIME (note that this is incompatible with Theorem 7.1 which gives a PTIME result for a larger class of queries, but under the restriction of the Codd interpretation of label variables).

An analog of Theorem 7.2 for patterns in $\mathcal{P}^{\text{nv,re}}$ was shown in Calvanese et al. [2000c], which implies tractability of RPQ evaluation in data complexity over patterns in $\mathcal{P}^{\text{nv,re}}$ whose underlying graph is of bounded treewidth. The idea of the proof in such case is the following: The domain of the structure $\mathcal{M}_Q$ will contain, in addition to 2^S, where S is the set of states of the NFA $\mathcal{A}$ that recognizes L, each function $\tau : S \to S$. We then enlarge $\mathcal{M}_Q$ by adding a unary predicate P_Γ, for each set Γ of functions from S to S. The interpretation of P_Γ consists of all the functions $\tau \in \Gamma$. The domain of $\mathcal{M}_\pi(n,n')$ will contain, in addition to the set N of node ids of π, each regular expression L' that labels an edge of π (and if two edges are labeled by the same regular expression, we see them as different objects in the domain). We then ensure that L' is mapped into some function $\tau : S \to S$, such that there is a word $w \in L'$ that *realizes* $\tau : S \to S$; that is, that there is a run of $\mathcal{A}$ over w from state s to $\tau(s)$, for each $s \in S$. In order to do that, we add L' to the interpretation of P_Γ in $\mathcal{M}_\pi(n,n')$, where Γ is the set of all functions $\tau : S \to S$, such that there is a word $w \in L'$ that realizes $\tau : S \to S$. It is not hard to prove that $\mathcal{M}_\pi(n,n')$ can be constructed in polynomial time (since L, and, thus, $\mathcal{A}$, is

fixed). A homomomorphism from $\mathcal{M}_\pi(n, n')$ to $\mathcal{M}_Q$ assigns a function $\tau : S \to S$ to each regular expression L' in the domain of $\mathcal{M}_\pi(n, n')$. Intuitively, this represents the *type* with respect to $\mathcal{A}$ of the word that will replace the regular expression L' in a graph database G that belongs to $[\![\pi]\!]$. This is all the information we need to know about that word in order to check whether $(n, n') \in Q(G)$.

We have not been able to extend these techniques to the most expressive class of patterns in $\mathcal{P}^{\text{nv,lv,re}}$. A possible explanation is that data complexity of RPQ evaluation is intractable even over extremely simple patterns in the class $\mathcal{P}^{\text{lv,re}}$, which makes the search for well-behaved fragments difficult.

PROPOSITION 7.3. *There is an RPQ Q such that* DATA COMPLEXITY*(Q) is intractable even over input patterns in $\mathcal{P}^{\text{lv,re}}$ with exactly one edge.*

The proof of Proposition 7.3 is by a mild modification of the proof of the second part of Theorem 9 in Barceló et al. [2013].

8. CONCLUSIONS

We studied structural properties and querying of graph patterns. We looked at three main features of patterns: node variables, label variables, and regular expressions specifying paths. We showed that each of these features strictly increases the expressiveness of patterns. We looked at data and combined complexity of answering CRPQs and other queries (both extensions and restrictions of CRPQs). We developed a model of automata that capture query answering, both for returning nodes and paths, and studied their properties. Finally, we identified tractable restrictions, as well as classes of reasonable combined complexity for which query answering is naturally viewed as a constraint satisfaction property.

The main conclusion is that, without carefully chosen restrictions, querying graph patterns is computationally harder than querying relational or XML patterns. In particular, this has implications for ongoing work on defining schema mappings as well as integration and exchange techniques for graph-structured data. However, we can identify rather robust classes with either tractable query answering, or for which one can hope to find good heuristics by using techniques from other fields. Developing such techniques is a natural continuation of this work. Another line for further work is to study tractable restrictions for integrating and exchanging graph data.

ELECTRONIC APPENDIX

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

ACKNOWLEDGMENTS

We thank Wenfei Fan and Peter Wood for their comments.

REFERENCES

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

Angles, R. and Gutierrez, C. 2008. Survey of graph database models. *ACM Comput. Surv. 40,* 1.

Arenas, M., Barceló, P., Libkin, L., and Murlak, F. 2010. *Relational and XML Data Exchange*. Morgan & Claypool.

Barceló, P., Hurtado, C., Libkin, L., and Wood, P. 2010a. Expressive languages for path queries over graph-structured data. In *Proceedings of the 29th ACM Symposium on Principles of Database Systems (PODS)*. 3–14.

Barceló, P., Libkin, L., Poggi, A., and Sirangelo, C. 2010b. XML with incomplete information. *ACM 58,* 1, 1–62.

Barceló, P., Libkin, L., and Reutter, J. 2013. Parameterized regular expressions and their languages. *Theoret. Comput. Sci. 474*, 21–45.

Björklund, H., Martens, W., and Schwentick, T. 2007. Conjunctive query containment over trees. In *Proceeding of the 11th International Symposium on Database Programming Languages (DBPL)*. 66–80.

Bonatti, P. A., Lutz, C., Murano, A., and Vardi, M. Y. 2008. The complexity of enriched mu-calculi. *Log. Meth. Comput. Sci. 8,* 4.

Börger, E., Gräedel, E., and Gurevich, Y. 1997. *The Classical Decision Problem*. Perspectives in Mathematical Logics, Springer-Verlag.

Buneman, P., Davidson, S. B., Hillebrand, G. G., and Suciu, D. 1996. A query language and optimization techniques for unstructured data. In *Proceedings of the SIGMOD Conference*. 505–516.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2000a. Answering regular path queries using views. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*. 389–398.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2000b. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 176–185.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2000c. View-based query processing and constraint satisfaction. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science (LICS)*. 361–371.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2002. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci. 64,* 3, 443–465.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. 2011. Simplifying schema mappings. In *Proceedings of the 14th International Conference on Database Theory (ICDT)*. 114–125.

Cheng, J., Yu, J. X., Ding, B., Yu, P. S., and Wang, H. 2008. Fast graph pattern matching. In *Proceedings of the 24th International Conference on Data Engineering (ICDE)*. 913–922.

Cohen, S. and Sagiv, Y. 2005. An abstract framework for generating maximal answers to queries. In *Proceedings of the 10th International Conference on Database Theory (ICDT)*. 129–143.

Consens, M. and Mendelzon, A. 1990. Graphlog: A visual formalism for real life recursion. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems (PODS)*. 404–416.

Cruz, I., Mendelzon, A., and Wood, P. 1987. A graphical query language supporting recursion. In *Proceedings of the ACM Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD)*. 323–330.

De Giacomo, G. and Lenzerini, M. 1997. A uniform framework for concept definitions in description logics. *J. Artif. Intell. Res. (JAIR) 6*, 87–110.

Dechter, R. 2003. *Constraint Processing*. Morgan-Kauffman.

Deutsch, A. and Tannen, V. 2001. Optimization properties for classes of conjunctive regular path queries. In *Proceedings of the 8th International Workshop on Database Programming Languages (DBPL)*. 21–39.

Diestel, R. 2005. *Graph Theory*. Springer.

Fagin, R., Kolaitis, P., Miller, R., and Popa, L. 2005. Data exchange: Semantics and query answering. *Theoret. Comput. Sci. 336,* 1, 89–124.

Fan, W., Li, J., Ma, S., Tang, N., and Wu, Y. 2010a. Graph pattern matching: From intractable to polynomial time. In *Proc. VLDB Endow. 3,* 1, 264–275.

Fan, W., Li, J., Ma, S., Wang, H., and Wu, Y. 2010b. Homomorphism revisited for graph matching. In *Proc. VLDB Endow. 3,* 1, 1161–1172.

Fan, W., Li, J., Ma, S., Tang, N., and Wu, Y. 2011. Adding regular expressions to graph reachability and pattern queries. In *Proceedings of the 27th International Conference on Data Engineering (ICDE)*. 39–50.

Glaister, I. and Shallit, J. 1996. A lower bound technique for the size of nondeterministic finite automata. *Inf. Process. Lett. 59,* 2, 75–77.

Gottlob, G., Koch, C., and Schulz, K. 2006. Conjunctive queries over trees. *J. ACM 53,* 2, 238–272.

Gutierrez, C., Hurtado, C., Mendelzon, A. O., and Pérez, J. 2011. Foundations of semantic web databases. *J. Comput. Syst. Sci. 77,* 3, 520–541.

Gyssens, M., Paredaens, J., Van den Bussche, J., and Van Gucht, D. 1994. A graph-oriented object database model. *IEEE Trans. Knowl. Data Eng. 6,* 4, 572–586.

Imielinski, T. and Lipski, W. 1984. Incomplete information in relational databases. *J. ACM 31,* 4, 761–791.

Johnson, D. and Klug, A. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput Syst. Sci. 28,* 1, 167–189.

Kanza, Y., Nutt, W., and Sagiv, Y. 2002. Querying incomplete information in semistructured data. *J. Comput. Syst. Sci. 64,* 3, 655–693.

Kolaitis, P. and Vardi, M. 2007. A logical approach to constraint satisfaction. In *Finite Model Theory and Its Applications*, Springer, 339–370.

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

Kupferman, O., Vardi, M. Y., and Wolper, P. 2001. Module checking. *Inf. Computat. 164,* 2, 322–344.

Lakshmanan, L., Ramesh, G., Wang, W. H., and Zhao, Z. 2004. On testing satisfiability of tree pattern queries. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*. 120–131.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS)*. 233–246.

Leser, U. 2005. A query language for biological networks. *Bioinformatics 21,* 2, ii33–ii39.

Libkin, L. 2004. *Elements of Finite Model Theory*. Springer.

Libkin, L. 2011. Incomplete information and certain answers in general data models. In *Proceedings of the 30th ACM Symposium on Principles of Database Systems (PODS)*. 59–70.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. 2002. Network motifs: Simple building blocks of complex networks. *Science 298,* 5594, 824–827.

Natarajan, M. 2000. Understanding the structure of a drug trafficking organization: A conversational analysis. *Crime Prevention Studies 11*, 273–298.

Olken, F. 2003. Graph data management for molecular biology. *OMICS: A Journal of Integrative Biology 7,* 1, 75–78.

Pérez, J., Arenas, M., and Gutierrez, C. 2009. Semantics and complexity of SPARQL. *ACM Trans. Datab. Syst. 34,* 3.

Ronen, R. and Shmueli, O. 2009. Soql: A language for querying and creating data in social networks. In *Proceedings of the 25th International Conference on Data Engineering (ICDE)*. 1595–1602.

San Martín, M. and Gutierrez, C. 2009. Representing, querying and transforming social networks with RDF/SPARQL. In *Proceedings of the 6th European Semantic Web Conference (ESWC)*. 293–307.

Tong, H., Faloutsos, C., Gallagher, B., and Eliassi-Rad, T. 2007. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 737–746.

Weikum, G., Kasneci, G., Ramanath, M., and Suchanek, F. 2009. Database and information-retrieval methods for knowledge discovery. *Commun. ACM 52,* 4, 56–64.

Received November 2011; revised May 2013; accepted November 2013