Graph Patterns: Structure, Query Answering and Applications in Schema Mappings and Formal Language Theory

Juan L. Reutter

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2013

Abstract

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, i.e., graph patterns. Queries need to be posed against such data, but techniques for querying patterns are generally lacking, and even simple properties of graph patterns, such as the languages needed to specify them, are not well understood.

In this dissertation we present several contributions in the study of graph patterns. We analyze how to query them and how to use them as queries. We also analyze some of their applications in two different contexts: schema mapping specification and data exchange for graph databases, and formal language theory. We first identify key features of patterns, such as node and label variables and edges specified by regular expressions, and define a classification of patterns based on them. Next we study how to answer standard graph queries over graph patterns, and 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 that guarantee tractability. We then turn to the the study of schema mappings for graph databases. As for relational and XML databases, our mapping languages are based on patterns. They subsume all previously considered mapping languages for graph databases, and are capable of expressing many data exchange scenarios in the graph database context. We study the problems of materializing solutions and query answering for data exchange under these mappings, analyze their complexity, and identify relevant classes of mappings and queries for which these problems can be solved efficiently. We also introduce a new model of automata that is based on graph patterns, and define two modes of acceptance for them. We show that this model has applications not only in graph databases but in several other contexts. We study the basic properties of such automata, and the key computational tasks associated with them.

Acknowledgements

I want to thank my supervisor Leonid Libkin for his support, collaboration, advice, lunch facts and old russian jokes. But specially when I had to attend different personal situations, covering my back when it was needed. Many thanks as well to Pablo Barceló and Jorge Pérez. The preliminary ideas for this dissertation started with a conversation with Leonid and Pablo, and together with Jorge they participated in many discussions that helped improving this work, leading to its completion. Finally, I want to thank all the people that helped me during my studies. To Pancha, who flew more than 10.000 kilometers, leaving many things behind to join me in Edinburgh, and to my coauthors Marcelo, Cristian, Rada, Tony, Egor, Andras, Amelie and Domagoj.

This work and my studies were supported by Conicyt BecasChile PhD Grant, EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Juan L. Reutter*)

Para Isabel y Sam

Table of Contents

List of Figures

Chapter 1

Introduction

Querying and mining graph-structured data is currently one of the most active research topics in the database community, fueled by the adoption of graph models in several new application domains. To name a few examples, the standard data model for Semantic Web applications [W3C Consortium, 2013] is a graph model called Resource Description Framework (RDF) [G. Klyne, 2004], and a similar model is used by several biological databases such as the Kyoto Encyclopedia of Genes and Genomes (KEGG) [Kanehisa and Goto, 2000] or parts of the National Centre for Biotechnology Information [Belleau et al., 2008]. Furthermore, commercial vendors of graph database management systems such as Neo4j [Neo4j, 2013] and InfiniteGraph [InfiniteGraph, 2013] claim that their products are used for a broad range of applications that includes social networks, fraud detection, geographical models, telecommunications and astronomical databases. In all these 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].

Graph patterns. 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], 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., 2010b, Fan et al., 2010a, Gutierrez et al., 2011, Gyssens et al., 1994, Leser, 2005, Milo et al., 2002, Natarajan, 2000, Pérez et al., 2009, Ronen and Shmueli, 2009, San Martín and Gutierrez, 2009, Tong et al., 2007, Weikum et al., 2009]; see also the surveys [Angles and Gutierrez, 2008, Angles, 2012].

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. Simple patterns are used, for example, in [Ogata et al., 2000, Matono et al., 2003] to extract *functionally related enzyme clusters* from the KEGG pathway database [Kanehisa and Goto, 2000], a

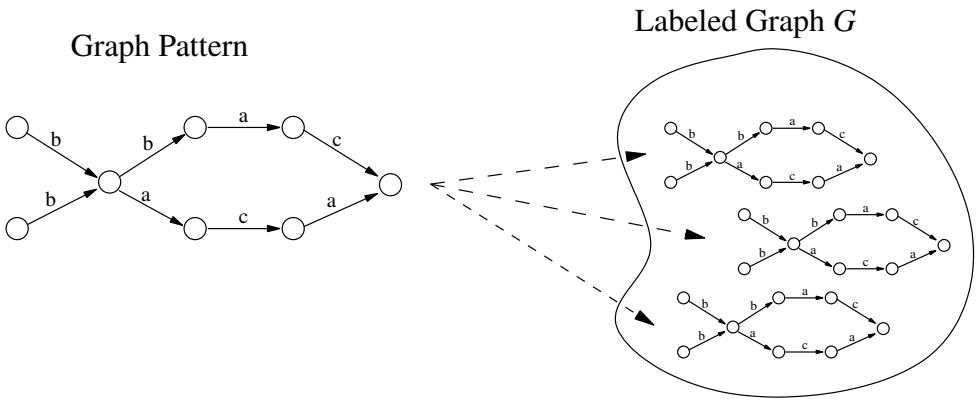


Figure 1.1: A simple graph pattern matching three subgraphs of G. Matching patterns of this form is common in metabolic pathway databases.

biological database that contains metabolic pathways, which are series of chemical reactions reactions that occur as part of the cellular processes of microorganisms. Biological networks are also used to model other processes such as gene transcription or protein interactions. Here it is also common to search for recurring or statistically significant patterns, which are called network motifs [Milo et al., 2002].

But many applications of graph databases raise the need for much more complex patterns, that specify information not only about the data itself, but also about the underlying *structure* of the graph database. This is specified by means of the so-called *reachability queries*, in which one can look for connection between elements in a network, and *regular path queries* (RPQs), a special case of reachability queries in which one can also specify restrictions on the path between the connected elements. Reachability queries are now a part of every major commercial developments of graph databases [Angles, 2012], and are now included in the standard specification for querying RDF databases [Harris and Seaborne, 2013].

Querying patterns instead of graphs. The notion of *querying* graphs with graph patterns has also evolved with time. In the simplest form, posing pattern queries against graph databases is nothing more that the traditional NP-complete subgraph isomorphism problem (this is still used, nonetheless, in practical applications, e.g., in [Cheng et al., 2008, Tong et al., 2007]). This is depicted in Figure 1.1, that shows a simple pattern on the left side, and how it matches two subgraphs of a much larger graph G (on the right of the figure). From a relational point of view, these simple graph patterns can be seen as analogs of (relational) conjunctive queries, which are the building block for most relational query languages.

But lately this simple notion of querying has evolved from graph isomorphism into notions based on graph homeomorphisms (i.e., mapping edges to paths) and simulation relations between patterns and graphs [Fan et al., 2010b, Fan et al., 2010a]. Outputs of matching queries

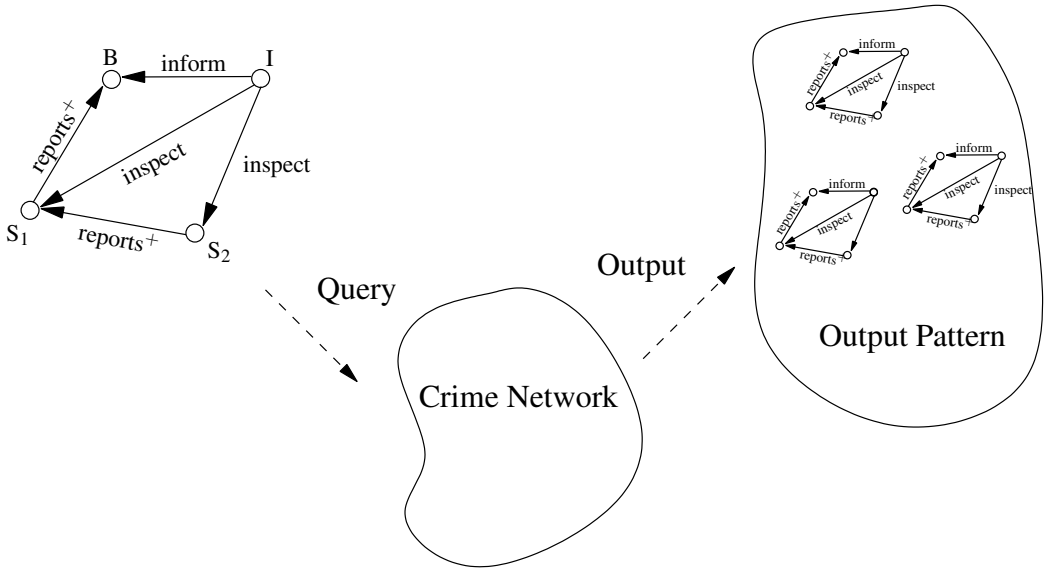


Figure 1.2: The output of the query is a pattern itself

are patterns themselves: their nodes are those that are involved in the simulation relation, and relationships between them are those specified in the pattern.

As an example, consider the structure of a drug crime organization, modeled as a graph database [Natarajan, 2000]. As figure 1.2 shows, we use a query to extract more detailed information about the structure of this network. This query is represented as graph pattern whose edges can be labelled by *regular expressions*, describing the structure of the paths connecting two individuals. Intuitively, the pattern in Figure 1.2 looks for information about the nature of the communication system of different bands in the network. It is interpreted as follows. The node B represents the *boss* of the crime network, and there are two *subordinates* S_1 and S_2 of the boss B. S_1 reports to B via a chain of *one or more* subordinates, represented with the edge labelled with the regular expression $reports^+$, and S_2 reports to S_1 via a similar chain. The pattern also describes that B uses a single informant I to keep both subordinates in check at the same time. The information in this pattern might be relevant, for instance, when trying to infiltrate the network, since one needs to take into account not only the command chain of the organization, but also the different informants used by the boss. As noted in e.g. [Fan et al., 2010b], the notion of subgraph isomorphism is meaningless for this pattern; one needs to use instead notions based on graph homeomorphisms or simulation relations. When using such semantics to query the crime network, the output is precisely the set of all the groups of individuals whose information structure coincides with the pattern, together with descriptions of paths specifying their relationships. In other words, the output of the query is a graph pattern itself

Similar scenarios arise in other applications that rely on graph structured data, and the need to output incomplete query results has been already pointed out for some of them, such

as the World Wide Web [Kanza et al., 2002]. When such matching and query results require extracting additional information from them, one ends up querying patterns rather than graphs. This shows a double use of graph patterns, since they not only serve as queries, but also as *data*, by representing partial information about graph-structured data.

The *duality* between data and queries is often present when dealing with incomplete information, where partial databases are typically viewed as patterns. For example, relational naive tables are tableaux of conjunctive queries [Imielinski and Lipski, 1984], and in XML, typical query languages are based on tree patterns, i.e., incomplete descriptions of documents [Barceló et al., 2010b, Björklund et al., 2007]. However, unlike other database scenarios such as relational databases or XML, that have received considerable attention in the literature, the research in graph patterns and graph incomplete information is currently lacking.

1.0.1 Main goals

The first and central goal of this dissertation is to study graph patterns, and the basics of querying them. We study them from two different point of views: as a query language for graph databases, and as a mean of representing partially defined graph data. In other words, we study how to *query* graph patterns, and how to *use them as queries*. We also analyze their applications in various scenarios, from data exchange to formal language theory.

To begin with, we need to formalize the notion of graph patterns. In order to do this we identify key features of patterns, such as node and label variables and edges specified by regular expressions, and define a classification of patterns based on these features.

We then turn to the topic of query answering, or how to query graph patterns. As for other data models that represent partial information [Arenas et al., 2010, Barceló et al., 2010b, Fagin et al., 2005a, 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, i.e., certain answers. In this dissertation we provide a thorough analysis of the problem of query evaluation: we give precise characterizations of both combined and data complexity of the problem of answering queries over graph patterns. For cases when complexity is high, we do further analysis of features that lead to intractability, as well as identifying several lower-complexity restrictions.

The study of the basics of graph patterns, and how to query them, allows us to apply our research in several database scenarios where the need for querying patterns naturally arises. In this dissertation we focus on two of these applications: schema mappings and data exchange for graph databases, and formal language theory.

Applications in schema mappings and data exchange. Querying partial information is commonly present in integrating and exchanging (or translating) data [Arenas et al., 2010, Fagin et al., 2005a, Lenzerini, 2002]. In such applications one starts with one or several data

repositories, which need to be integrated or translated, and a set of rules specifying how one is to carry these transformations, that are usually denoted as *schema mappings*. The whole process normally results in partially specified databases, which then needs to be queried.

After years of persistent research, the foundations have been laid for both relational and XML database data exchange and integration, and we now understand many algorithmic issues related to these tasks (see [Kolaitis, 2005, Barceló, 2009, Arenas et al., 2010] for surveys in the area). But, at the same time, the study of interoperability issues among graph-structured data sources remain almost unexplored from a theoretical point of view. Some remarkable exceptions can be found in the work by Calvanese et al. regarding rewriting of views for graph databases [Calvanese et al., 2000b], and more recently the study of schema mapping simplification in the same context [Calvanese et al., 2011a].

One of the lessons learned from the research of relational and XML data exchange and integration is that patterns have been a key ingredient in the formalization and study of these tasks. Indeed, patterns are at the core of most of the schema mappings studied in the literature, and even commercial data exchange tools are based on them [Fagin et al., 2005b, Kolaitis, 2005]. It is thus natural to use our framework based on graph patterns to study tasks such as data exchange, data integration and, more generally, schema mapping management in the context of graph database.

As the second main goal of this dissertation, we embark on the theoretical study of schema mapping specification and data exchange for graph databases, the latter being currently unexplored. As we have mentioned, we shall base our mappings on graph patterns, thus adapting the usual notion of a source-to-target dependency into the graph database context. Afterwards, we apply our mappings to study the problem of data exchange, which can be described as follows: Given a mapping $\mathcal{M}$ from a *source* schema to a *target* schema and a source database D, compute a target database that better reflects the source data in D under $\mathcal{M}$ [Fagin et al., 2005b, Barceló, 2009, Arenas et al., 2010]. Such target database is said to be a *solution* for D under $\mathcal{M}$. In relational data exchange one normally computes a *universal* solution [Fagin et al., 2005b], which is a database with incomplete information that represents the entire space of solutions for D under $\mathcal{M}$. In graph data exchange we show that such a solution always exists for our mappings, and it corresponds to our well studied notion of graph patterns.

Applications in language theory. The connection between formal language theory and graph databases starts with the observation that graph databases can be seen as non-deterministic finite state automata (NFAs) without distinguished initial and final states [Cruz et al., 1987, Consens and Mendelzon, 1990]. In the same sense, a graph pattern representing partial graph information can be seen as some sort of *incomplete automaton*, in which the precise relationships defining the transition function, or the structure of the automaton, is missing. The third main topic of this dissertation is to study graph patterns from a language-theoretic point of

view or, more precisely, incomplete automata. We show that there are several possibilities of semantics for incomplete automata; some of this semantics have strong relationships with the task of querying graph patterns, but others relate to algorithmic tasks of completely different applications, in areas such as program analysis.

1.1 Contributions of this Dissertation

This dissertation present contributions in the definition, study and application of graph patterns as a paradigm for specifying partially defined graph structured data. We detail these contributions below.

1.1.1 Classifying and querying graph patterns

Chapter 3 and 4 form the first part of this dissertation, and are dedicated to study the basics of incomplete information in graph databases, in the forms of graph patterns.

The first task is to formally define the notion of graph patterns we use in this dissertation. Chapter 3 begins with a discussion on what types of features should be added to patterns, in order to represent incomplete information. Based on examples from a wide range of applications – from social network to crime databases – we settle for objects in which partially defined information can be defined in the following ways. First, as is usual in relational naive tables and XML patterns, we model incompleteness with node variables, that may appear in patterns instead of node id's. These variables represent the lack of information about the *data* that is represented by the pattern. Node variables are usually enough to represent incomplete relational information, but in semistructured data we also have the need to represent the lack of *structural* information i.e., missing information regarding the edges and labels of the graph. To model it we add two extra features to patterns: label variables, that may replace some of the edge labels in the pattern, and regular expressions in the edges, representing missing information about the connection between two nodes.

The foundations of our work are laid in Section 3.2. Here we provide a formal definition of graph patterns, and we divide them into different classes, depending on which of the key features listed above – node variables, label variables, and edges labeled with regular expressions – they use. Afterwards, in Section 3.3 we provide a complete classification of the expressiveness of each of these classes of patterns, showing that adding each new feature to graph patterns strictly increases their expressiveness.

Graph queries. As we have mentioned before, when dealing with incomplete data one often encounters a *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, i.e., incomplete descriptions of documents.

We also investigate this duality for the case of graph patterns, and define the general notion of *graph queries* as those queries which are represented by a graph pattern. We compare these queries with some of the most typical query languages for graphs. Usually these languages specify the existence of paths between nodes, with the restriction that the labels of such path belong to regular languages [Abiteboul et al., 1999, Cruz et al., 1987, Consens and Mendelzon, 1990, Gyssens et al., 1994, Calvanese et al., 2002]. 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. We shall see that queries such as RPQs and CRPQs arise as special cases of graph patterns, continuing the analogy with relational and XML cases.

Query answering. Chapter 4 deals with the problem of querying graph patterns. In this chapter we study the complexity of the problem of finding certain answers to queries over graph patterns, or just query answering, for short (recall that certain answers are those answers that are independent of the way in which the missing parts of patterns are interpreted).

In its more general definition, the problem can be specified as follows: Given a graph pattern π, a graph query ξ of arity k and a tuple $\bar{t}$ of nodes of size k, decide whether $\bar{t}$ is in the certain answers of ξ over π. We study this problem in its general definition, and also investigate what happens when one restricts its inputs. For patterns we range from arbitrary graph patterns to graph patterns using none of the features mentioned above (in other words, just graph databases); and for queries we analyze the upper bounds for queries given by arbitrary graph patterns , and provide a full analysis for CRPQs. We also study the *data complexity* of the problem, that is, the problem when the query is considered to be fixed, and only the pattern comes as input.

Our results shows that the problem is always in EXPSPACE, and in coNP for data complexity. The full analysis for CRPQs shows that data complexity ranges from NLOGSPACE to coNP, and combined complexity (when both query and pattern are inputs) ranges from EXPSPACE to NP. As for the techniques, we will show that most of our results regarding classes of patterns that only use node variables can be obtained based on previous results in incomplete databases, but that much more involved techniques are needed for more expressive patterns using any of the other two features (label variables or regular expressions in the edges). Here most of our results are based on the relationship between certain answers and implication of patterns, as noted in [Calvanese et al., 2000b].

Most of the results in Chapters 3 and 4 were published in PODS 11 [Barceló et al., 2011b], but some of the results in Section 4.2 are presented here for the first time.

1.1.2 Tractability restrictions and heuristics for querying graph patterns

The results in Chapter 4 point to a rather high complexity of querying graph patterns. If we focus in the data complexity of the problem, we have a general coNP-hardness lower bound, which applies to both arbitrary graph queries or CRPQs over any class of patterns that use label variables or regular expressions in the edges. This contrasts results from relational incomplete information, where answering patterns, or conjunctive queries over typical relational databases with incomplete information (usually called naive tables) is in PTIME [Imielinski and Lipski, 1984].

For patterns that do not use any of these features, the complexity of querying CRPQs drops to NLOGSPACE, which is expected, since patterns without label variables or regular expressions are precisely those that can be expressed as relational naive tables, and thus for this class we can reuse all the machinery that was developed in the relational scenario.

But the question of tractability remains for patterns using either label variables or regular expressions, since relational tools will not work for them. It is therefore natural to ask for restrictions - or heuristics - to improve the complexity of the problem. This is the subject of Chapter 5.

Lower bounds. The first thing we need to do is to analyze the coNP lower bound for querying patterns, and see how the complexity of the problem behaves under various restrictions on both patterns and queries. We do this in Section 5.1, first for the case when the input to the problem is a pattern with label variables, and then for patterns with regular expression on the edges. In both cases, the results show that the coNP bound is quite resilient. To name a few of the results, we prove that, if the input is a pattern with label variables, then query answering is coNP-hard even if the underlying graph (i.e., the structure of the pattern) resembles a path and over very limited queries, and for regular expressions the coNP lower bound remains even if the structure of the pattern is a directed acyclic graph (DAG), and even if these expressions are very simple – of form $a_1|a_2$ for some letters a_1, a_2 –.

Tractable restrictions. Our refined lower bounds still allow some room for tractable cases. Since the lower bound remains for DAGs, it is natural to look for querying patterns whose structure resemble trees, or, more generally, graphs with *bounded treewidth*. We show in Section 5.2 that indeed an interesting tractable fragment can be found on these lines: the data complexity of answering CRPQs over patterns is in PTIME if the structure of the input pattern is a graph of bounded treewidth, as long as the pattern does not make use of arbitrary label variables.

Another possibility is to look for fragments that involve restricting both patterns and queries at the same time, or the interaction between them. Indeed, most of the coNP-hardness reductions we show on Section 5.1 make use of patterns that interact with queries in a rather unnatural

way. But what if one could restrict the problem to disallow only those patterns that have these complex interactions with the queries? We answer this question in Section 5.3. More precisely, we formalize the notion of a complex interaction between patterns and queries, in terms of the behavior of all the potential witnesses for the query over a given pattern. For some classes of patterns, this notion can be computed efficiently, and gives rise to a second, novel tractable fragment for the data complexity of query answering. We also show that our restrictions are, in a sense, optimal, since lifting any one of them leads to intractability.

Query answering as constraint satisfaction. We finish Chapter 5 with a rather different approach. Instead of looking for further restrictions, we show that the problem of answering queries over patterns is tightly related to a well known area of research: the constraint satisfaction problem. In order to take advantages of all the advances in solving constraint satisfaction problems, we show how to cast the query answering problem as a constraint satisfaction problem, with a particularly simple translation for patterns in that use both node and label variables. The consequences of the results of this section are twofold. On one hand this enables us to use the machinery - and heuristics- that have been developed through many years in solving constrain satisfaction problems, but on the other hand, it tells us that it will be probably difficult to obtain a precise characterization of tractability for query answering, since characterizing the instances of CSP that are in PTIME is a longstanding open problem [Dechter, 2003, Kolaitis and Vardi, 2007].

The results from this chapter have been previously published in PODS11 [Barceló et al., 2011b] and ICDT 2013 [Barceló et al., 2013a].

1.1.3 Schema Mappings and Data Exchange for graph databases

In Chapter 6 we embark on the theoretical study of schema mapping specification and data exchange for graph databases, the latter being currently unexplored.

Schema mappings. Schema mappings are high-level specifications that permit to define relationships between two different schemas [Bernstein, 2003, Lenzerini, 2002, Arenas et al., 2010]. Schema mappings have received considerable attention in the context of relational databases [Nash et al., 2005, Fagin et al., 2005c, Fagin, 2007], see also [Arenas et al., 2009]. Regrettably, the possibility of using relational mappings tools for solving interoperability tasks is not suitable for graph databases. We delve into this issue below.

Relational mappings are typically defined in terms of conjunctive queries, and hence they lack any form of recursion which is a crucial feature for querying graph databases [Abiteboul et al., 1999, Wood, 2012]. Therefore, additional features would need to be included in this class of mappings in order to specify more complex navigational properties. However, this would imply leaving relational data exchange tools behind, since the study of relational

data exchange has been carried out mostly in terms of mappings defined by non-recursive queries [Barceló, 2009]. The problem is that even when navigational properties can sometimes be expressed in SQL or in other extensions of relational calculus, such as DATALOG, it is not clear how to define schema mappings based on those languages. Moreover, given the high complexity cost associated with performing simple static analysis tasks for them, such as equivalence or containment, using mappings based in unrestricted SQL or DATALOG may leave us without practical algorithms for even the most simple data exchange tasks.

We thus require a class of mappings that is specifically tailored for graph databases. As it has been done in relational and XML settings, our first candidate as a formalism for defining a mapping language is again graph patterns. As we shall see on Chapter 6, these types of mappings are capable of expressing a wide variety of interesting exchange properties in the graph database context. Section 6.2 deals with the formalization of graph schema mappings, and then in Section 6.3 we explore some properties of our class of mappings. Notably, we show in this section how these mappings can express not only usual exchange properties based on exporting tuples of elements, but also complex navigational properties, such as exporting entire paths satisfying some regular conditions.

Data Exchange. We then apply our mappings to study data exchange in the graph database context. Recall that the data exchange problem is the following: Given a mapping $\mathcal{M}$ from a *source* schema to a *target* schema and a source database D, compute a target database that best reflects the source data in D under $\mathcal{M}$ [Fagin et al., 2005b, Barceló, 2009, Arenas et al., 2010]. Such target database is said to be a *solution* for D under $\mathcal{M}$. In relational data exchange one normally computes a *universal* solution [Fagin et al., 2005b], which is a database with incomplete information that represents the entire space of solutions for D under $\mathcal{M}$. In graph data exchange we face the analog problem of computing a *universal representative*, which is an incomplete graph database, with missing information both at the data and at the structural level, with the same good properties.

This suggest, once again, the need for graph patterns. And indeed in Section 6.4 we show how a universal representative can be obtained, in the form of a graph pattern, for every graph data exchange setting. We also show that universal representatives enjoy many of the good properties of universal solutions. In particular, the usual techniques for computing universal solutions in relational data exchange can be applied to computing universal representatives in graph data exchange. The procedure works in exponential time in combined complexity (that is, assuming mappings and databases to be part of the input), and in polynomial time in data complexity (that is, assuming mappings to be fixed).

Another important problem in data exchange is query answering. As for the case of graph patterns, here one is interested in computing the *certain* answers of a query [Lenzerini, 2002, Barceló, 2009, Arenas et al., 2010], which are defined in this case as those answers that

hold in all possible solutions. Just as in relational data exchange [Fagin et al., 2005b, Arenas et al., 2011b], we do this in a two-step fashion: First, we compute a universal representative (a graph pattern), and then we evaluate the query over the universal representative.

This two step fashion almost immediately gives us upper bounds for computing the certain answers of queries in a data exchange setting, as it essentially reduces to the problem of querying graph patterns. Unfortunately, it also tells us that the problem of computing certain answers in graph data exchange is inherently difficult in combined complexity, and that even in data complexity (that is, assuming queries and mappings to be fixed) we easily face intractability. This motivates the search for relevant classes of mappings and queries for which the problem of computing certain answers is tractable. Since Chapter 5 deals with the issue from the point of view of data complexity, we devote the rest of Chapter 6 to find tractable scenarios in combined complexity.

Tractable data exchange settings. Finding classes of data exchange settings in which both computing representatives and query answering are tractable in combined complexity appears to be specially relevant for graph databases. Based on the assumption that mappings and queries are often much smaller than the data (and hence a meaningful complexity analysis should not locate databases and specifications at the same level), the analysis of traditional data exchange settings has been mostly carried out in terms of data complexity. However, for the case of graph databases and their massive data applications (such as, for example, social networks or scientific databases) this assumption is not longer valid. For instance, the procedure mentioned above for constructing universal representatives in graph data exchange works in time $|G|^{O(|\mathcal{M}|)}$, for a source graph database G and a mapping $\mathcal{M}$, which can be considered infeasible for big graph databases, even for small $\mathcal{M}$. It is thus important for the study of graph data exchange to identify relevant classes of mappings for which basic computational tasks can be solved efficiently, not only in data but also in combined complexity.

It follows from the observations above that most pattern-based mappings will be doomed from a combined complexity point of view, including those based in CRPQs. For this reason, we turn instead to mappings defined with binary queries, such as the well known language of RPQs. Defining our mappings using RPQs, however, would limit our expressive power to a great extent. Instead, we use *nested regular expressions* [Barceló et al., 2012], to define *nested path queries* (NPQs), a class of queries that extends RPQs with the ability to traverse edges in both directions and to perform branching while navigating the data. Using these mappings, which we call NPQ-restricted mappings, we show in Sections 6.5 and 6.6 how one can perform all the aforementioned data exchange tasks in polynomial time in combined complexity, without sacrificing too much expressive power (in comparison with mappings defined by arbitrary patterns). Interestingly enough, one can even apply query rewriting techniques to obtain linear-time algorithms for query answering, under some further restrictions on mappings.

Composition of mappings. Section 6.7 presents a brief discussion about the good metadata management of NPQ-restricted mappings, in particular, on how to compose them, and whether the composition of two of them can be expressed in some language. Our results show that, unlike the relational case, it is very unlikely that even the composition of two NPQ-restricted mappings may be expressed with any sort of natural graph mappings, but, on the other hand, we identify a smaller class of mappings that is *closed* under composition, i.e., the composition of any two mappings of this class can be expressed in the same class. We believe that this discussion is the starting point for the research of metadata management of graph mappings, in the spirit of relational mappings (see [Arenas et al., 2009] for a survey).

Most of the results in Chapters 6 have already appeared in ICDT'13 [Barceló et al., 2013a]. The results from the expressibility of nested regular expressions were published in AMW'12 [Barceló et al., 2012]. Chapter 6 also contains results that are presented here for the first time.

1.1.4 Applications in Formal Language Theory

Just as graph databases can be viewed as finite automata, graph patterns in turn can be viewed as some form of automata where the precise information of the transition relation is missing. We denote these as *incomplete automata*, which formally are nothing more than graph patterns with distinguished initial and (a set of) final nodes. If we restrict our patterns to contain only label variables (no regular expression in the edges), then we arrive at the notion of *parameterized automata*, and its corresponding notion of regular expressions, that we call *parameterized regular expressions*, a subclass of incomplete automata that is of independent interest.

As the final main contribution of this dissertation, in Chapters 7 and 8 we study these automata from a language-theoretic point of view, and show how this research has applications in a great range of different scenarios.

Semantics and applications. We start in Section 7.1 with the definition of the semantics of incomplete and parameterized automata. Intuitively, it resembles the semantics of graph patterns: just as a graph pattern represents a set of graph databases, each incomplete automaton represents a (possible infinite) set of *complete* automata (this is, NFA's).

We can then define two semantics for incomplete automata. Under the *certainty* semantics, a word w belongs to the language of an incomplete automaton $\mathcal{A}$ if w is in the language of all the automata represented by $\mathcal{A}$. This semantics is inspired by the idea of certain answers, and it is therefore natural to ask whether the notion of querying graph patterns is related to the notion of acceptance of incomplete automata under certainty semantics. The answer is positive: we show in Section 7.2 that, for RPQs of form $\varphi = (u, w, v)$, it is possible to cast the problem of answering φ over a graph pattern π as a purely language-theoretical problem: whether w belongs to the language of the incomplete automata generated by π. Interestingly, we also show that the certainty semantics for incomplete automata provides the correct framework to

study the problem of computing certain answers over patterns, but for queries that can also return *paths*, apart from tuples of nodes, such as the language of ECRPQs that was presented in [Barceló et al., 2010a].

Our second semantics for incomplete automata is the *possibility* semantics, which is built using union instead of intersection: a word w belongs to the language of an incomplete automaton $\mathcal{A}$ if w is in the language of *any* of the automaton represented by $\mathcal{A}$. While not directly related to databases, these semantics of incomplete automata arise in a variety of applications, such as static analysis of programs [Liu et al., 2004, Liu and Stoller, 2006, de Moor et al., 2003], in phase-sequence prediction for dynamic memory allocation [Shen et al., 2007], or as a compact way to express a family of legal behaviors in hardware verification [Bhadra et al., 2005], or as a tool to state regular constraints in constraint satisfaction problems [Pesant, 2004].

Basic Properties. We have mentioned several applications for both of our semantics of incomplete automata. At the same time, however, very little is known about the basic properties of these languages. Incomplete automata have not been studied before under certainty semantics, and while there are several papers on the possibility semantics (see e.g. [Grumberg et al., 2010, Kaminski and Zeitlin, 2010]), these are concentrated in the context of *infinite* alphabets. The motivation of [Grumberg et al., 2010] comes from the study of infinite-state systems with finite control (e.g., software with integer parameters). In contrast, for the applications outlined before, finite alphabets are more appropriate [Liu et al., 2004, Liu and Stoller, 2006].

Thus, our main goal is to determine the exact complexity of the key problems related to languages generated by both certainty and possibility semantics of incomplete automata.

Although somewhat surprising at a first glance, we show in Section 7.3 that the language generated by certainty semantics over incomplete automata are indeed regular. For possibility semantics, languages need not be regular, but parameterized automata always yield regular languages under possibility semantics.

Whenever these languages are regular we establish upper bounds on the running time of algorithms for constructing NFAs, and then prove matching lower bounds for the sizes of NFAs representing both certainty and possibility semantics. These lower bounds are much stronger: they even apply for parameterized regular expressions.

Decision problems. In Chapter 8 we discuss several standard language-theoretic decision problems, such as membership of a word in the language, language nonemptiness, universality, and containment. We fully determine the complexity of all these decision problems. All of them end up being complete for various complexity classes, from NLOGSPACE to EXPSPACE. We also show that some of these problems may be undecidable for the case of possibility semantics. Moreover, we show that, save for a few exceptions, the complexity of these problems is not dependent on a particular model of automata: usually the upper bounds that we present in

this section apply to the more general notion of incomplete automata, while the lower bounds are shown for the restricted parameterized automata, and even for parameterized regular expressions.

The results from these chapters have been previously published in PODS11 [Barceló et al., 2011b], FSTTCS 2011 [Barceló et al., 2011a] and TCS [Barceló et al., 2013b]. Most of the upper bounds in Sections 8.2 to 8.6 appear for the first time in this dissertation, since they had to be extended from previously published results for parameterized regular expressions, in order to manage the case when the input is any arbitrary incomplete automaton.

Chapter 2

Background

In this chapter we present the notation that is used thorough this dissertation, and review some key results in the areas of graph databases and incomplete information that will be useful for the presentation of our ideas. We assume familiarity with first order logic and with the basics of formal language theory, specifically regular languages, finite automata and regular expressions.

Section 2.1 deals with graph databases. Here we present key definitions and introduce the languages of regular path queries (RPQs) and conjunctive regular path queries (CRPQs) that will take a prominent role in this dissertation. Continuing with the topic of query answering, we also review how the connection between graph databases and automata leads us to query answering algorithms that match those of conjunctive queries in relational databases.

Afterwards, in Section 2.2, we review some of the basics ideas behind the theory of incomplete information in relational databases, in particular, the definition of incomplete relational databases and the algorithms for querying them. These notions are also crucial for our dissertation. For example, even though relational querying algorithms are not directly related to graph databases, some of the graph based algorithms that we present in this dissertation rely on the same underlying ideas as those for incomplete information in relational databases.

2.1 Graph Databases and Queries

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 $\mathbf{V}$ 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 $\mathbf{V}$), and E is the set of edges, i.e., $E \subseteq N \times \Sigma \times N$. 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), $(n_1, a_1, n_2), \cdots, (n_{m-1}, a_{m-1}, n_m)$, 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.

Note that each graph database $G = (N, E)$ over Σ can be naturally viewed as a nondeterministic finite automaton (NFA) over alphabet Σ without initial and final states. Its states are the nodes in N, and its transitions are edges in E. This basic idea is key for many results in graph databases, and we use it for several results in this dissertation.

2.1.1 Regular path queries

The basic querying mechanism for graph databases is provided by means of *regular path queries*, or *RPQs* [Abiteboul et al., 1999, Cruz et al., 1987, Calvanese et al., 2002]. 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 over Σ 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., [Cruz et al., 1987, Consens and Mendelzon, 1990, Abiteboul et al., 1999, Calvanese et al., 2000b]) that analogs of conjunctive queries whose atoms are RPQs are much more useful in practice than simple RPQs. This motivated in [Florescu et al., 1998] the study of *conjunctive regular path queries*, or *CRPQs*. In such queries, multiple RPQs can be combined, and some variables can be existentially quantified. Formally, a CRPQ Q over a finite alphabet Σ is an expression of the form:

$$Q(\bar{z}) = \bigwedge_{1 \leq i \leq m} (x_i, L_i, y_i), \tag{2.1}$$

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 $Q(\bar{z})$ is the *head* of the query, the expression on the right of the equality is its *body*. A query with the head $Q()$ (i.e., no variables in the output) is called a *Boolean* query.

Intuitively, such a query 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 (2.1) 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, i.e., 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 (that is, as usual, the singleton set with the empty tuple models `true`, and the empty set models `false`).

There are other extensions of regular paths queries that have been studied in the literature. Cruz et al. studied in [Cruz et al., 1987] the language G, a query language that resembles the patterns that we study in this dissertation, albeit with a different semantics, and Consens and

Mendelzon defined *Graphlog* in [Consens and Mendelzon, 1990], a query language that augmented patterns with recursion. More recently, *2-way* regular path queries, or 2RPQs, and the corresponding *conjunctions* of 2RPQs (C2RPQs), as introduced in [Calvanese et al., 2000b], extend RPQs and CRPQs with the possibility of traversing an edge of the graph *backwards*, and ECRPQs [Barceló et al., 2010a] include the possibility of returning paths as outputs, as well as allowing some synchronization between these paths. We shall briefly discuss 2RPQs and C2RPQs in Chapter 6 as a possibility of expressing schema mappings, and ECRPQs will be analyzed in Chapter 7 when studying queries that can return paths. But for the rest of this dissertation we prefer to maintain RPQs and CRPQs as our de-facto language for graph databases, as they form the backbone of all aforementioned query languages.

2.1.2 Answering regular path queries

We now present the main ideas behind some results on the complexity of answering RPQs and CRPQs over graph databases [Cruz et al., 1987, Consens and Mendelzon, 1990]. The main conclusion is that one can obtain algorithms for answering CRPQs that match the complexity of relational conjunctive queries. We do this by exploiting the connection between graph databases and automata. To be more precise, we want to solve the following problem:

PROBLEM:	QUERY ANSWERING
INPUT:	A CRPQ $Q(\bar{x})$ with $\lvert\bar{x}\rvert = k$, a graph database G over Σ and a tuple $\bar{v} \in N^k$.
QUESTION:	Is $\bar{v} \in Q(G)$?

Let us now review how to solve this problem when the input is an RPQ. The idea is the following. Given a graph database $G = (N, E)$ over Σ, an RPQ of form $Q(x,y) = (x,R,y)$ and a pair (u,v) of nodes from N, in order to decide wether (u,v) belongs to $Q(G)$ one constructs from G the automaton $A_G(u,v) = (N, \Sigma, u, \{v\}, E)$ and the automaton A_R that accepts the language given by R. Then one can show that (u,v) belong to $Q(G)$ if and only if the language of the product automaton $A_G(u,v) \times A_R$ is nonempty.

The algorithm above automatically gives us an NLOGSPACE upper bound for answering RPQs over graph databases, by performing a standard on-the-fly verification algorithm. For CRPQs, it also gives us an NLOGSPACE upper bound for data complexity of query answering (i.e., assuming that the query Q is fixed), but with a little bit of effort one can use this technique to show that the combined complexity is in NP, and therefore NP-complete, since the problem is NP-hard already for conjunctive queries.

2.2 Incomplete Information in Relational Databases

Next we briefly recall the basics of incomplete information in relational databases, as studied in [Imielinski and Lipski, 1984, Abiteboul et al., 1991, Grahne, 1991, Abiteboul et al., 1995].

A *relational schema* $\mathbf{S}$ is a finite set $\{R_1,\ldots,R_k\}$ of relation symbols, with each R_i having a fixed arity $n_i \geq 0$. Let $\mathbf{D}$ be a countably infinite domain. An instance I of $\mathbf{S}$ assigns to each relation symbol R_i of $\mathbf{S}$ a finite relation $R_i^I \subseteq \mathbf{D}^{n_i}$. $\mathrm{Inst}(\mathbf{S})$ denotes the set of all instances of $\mathbf{S}$. The *domain* $\mathrm{dom}(I)$ of instance I is the set of all elements that occur in any of the relations R_i^I. We say that $R_i(t)$ is a fact of I if $t \in R_i^I$. Note that every instance can therefore be denoted by its set of facts.

2.2.1 Incomplete Information

In relational databases, incomplete information is modeled by means of *naive tables*. These are sets of facts over a schema $\mathbf{S}$ that use both elements of $\mathbf{D}$ as well as *variables*, the latter to represent missing information. We always assume that variables come from an infinite set $\mathcal{W}$ that is disjoint with $\mathbf{D}$.

Naive tables represent a set of instances over $\mathbf{S}$, the semantics is usually given in terms of homomorphisms. Formally, a homomorphism from a table T to an instance I is a mapping $h : \mathrm{dom}(T) \to \mathrm{dom}(I)$ such that (1) h is the identity on elements from $\mathbf{D}$ and (2) for each fact $R(t_1,\ldots,t_n)$ in T the interpretation $R^I$of R in I contains tuple $h(t_1),\ldots,h(t_n)$ (in other words, the fact $R(h(t_1),\ldots,h(t_n))$ is in I). We then say that an instance I of $\mathbf{S}$ is represented by a naive table T if there is a homomorphism from T to I [1]. We say that $\mathrm{Rep}(T)$ is the set of all instances represented by T.

Example 2.2.1 *Consider a schema* $\mathbf{S}$ *with binary relations* R *and* S. *Then* $T = \{R(a,x),R(x,y),S(y,b),S(a,b)\}$ *is a naive table. We usually adopt the convention that variables are represented by lowercase letters such as* $x,y,z,x_1,\ldots$. *The following instance belong to* $\mathrm{Rep}(T)$*:* $I_1 = \{R(a,a),R(a,b),S(a,b),S(b,b)\}$*, which can be shown using the homomorphism that maps* x *to* a *and* y *to* b *(and is the identity on constants* a *and* b*). The instance* $I_2 = \{R(a,a),S(a,b)\}$ *and* $I_3 = \{R(a,a),S(a,b),S(c,c)\}$ *also belongs to* $\mathrm{Rep}(T)$*, via the homomorphism that maps both variables* x *and* y *to the constant* a.

Note that naive tables are really nothing more than conjunctive queries. This duality between queries and incomplete databases is common in the area of incomplete information, and will also take a prominent role in our dissertation.

Other types of tables have been proposed in the literature, apart from naive tables. For example, *Codd* tables are naive tables in which all variable occurrences are distinct; and there are other models which extend naive tables, such as *conditional* tables

[1] This semantics is usually described as *open world assumption* in the literature

[Imielinski and Lipski, 1984], where one can impose much more complex conditions on the appearance of tuples.

2.2.2 Query Answering and Naive Evaluation

One of the most important computational problems related to incompleteness is query answering. Here one is typically interested in computing the *certain answers* of queries. To define these, let Q be a query and T a naive table, both over schema $\mathbf{S}$. Then the certain answers of Q over T corresponds to the set $\text{CERTAIN}(Q,T) = \bigcap\{Q(R) \mid R \in \text{Rep}(T)\}$.

The problem of deciding wether a tuple $\bar{t}$ of values belongs to the certain answers of Q over T has been shown to be undecidable if Q is an arbitrary relational algebra expression[Abiteboul et al., 1991]. However, one of the most important results from [Imielinski and Lipski, 1984] tells us that query answering over naive tables is tractable if Q is a union of conjunctive queries. In this case, certain answers are computed by a process called *naive evaluation*. Under it, we first compute the set $Q(T)$ as if T was a relational instance, treating variables as values, and then obtain $\text{CERTAIN}(Q,T)$ by removing from $Q(T)$ all tuples that contain variables (i.e., only variable-free tuples are kept in the output). Thus, computing the certain answers of conjunctive queries over naive table has the same complexity as the problem of computing the answers of conjunctive queries over relational instances: it is in LOGSPACE if the query is assumed to be fixed, and NP-complete in combined complexity.

Chapter 3

Graph Patterns

In this chapter we formally define the notion of graph patterns that is used thorough this dissertation. As in the case of tree-structured data, e.g. 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 chapter 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].

We begin this chapter with an analysis, through various examples, of the key features that need to be addressed when dealing with partial information in the graph database context.

Once we have identified these features, we proceed to formalize the notion of graph patterns, and define their semantics. In particular, we show that patterns satisfy our desiderata, as they can be used to define many interesting query languages, and, in particular, RPQs and CRPQs. This query language will be the base of our query answering algorithms in the following chapters, and also of the schema mappings for graph databases that we define in Chapter 6.

We finish this chapter by proving that each feature of patterns strictly increases its expressiveness, so that for example adding node variables to patterns always result in a more expressive class of patterns. We conduct this study for general patterns, and also for a restriction based on the notion of *Codd* tables from relational databases.

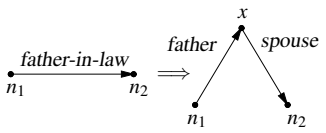
3.1 Forms of Incompleteness in Graph Databases

We now analyze types of features that need to be addressed in the study of graph patterns. Our account is based on several examples of applications where the need to deal with partial information arises in a natural way, many of them –from crime detection to extracting and transforming graph data– where presented in the introduction.

Recall that in the relational case, one has the need to deal with variables in place of missing data values [Imielinski and Lipski, 1984]. In the case of XML, one may also have 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 by the following figure:

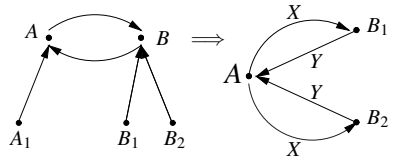


There are several other scenarios where node variables are important. For example, variables are key when modeling graph queries as patterns [Consens and Mendelzon, 1990], and they can also be used to model blank nodes in RDF [Pérez et al., 2009].

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

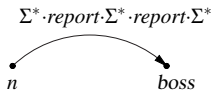
$$(A_1,X,B),\ldots,(A_n,X,B),\ (B_1,Y,A),\ldots,(B_m,Y,A)$$

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. The following figure illustrates the case for followers B_1 and B_2 of B.



Regular languages We can use regular expressions to model scenarios in which one loses information about the certain relationship between two nodes. Returning to the example with

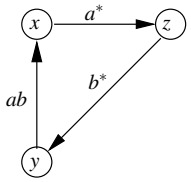
crime detection in a network of people, where the result of a matching may contain facts like "there is a path between x and the boss that goes via a chain of at least one intermediary", we showed how to use a regular language to express this relationship, namely the regular expression $report^+$, where the label *report* indicates subordinacy in the hierarchy. More complex relationships may arise that require more complex regular expressions. For example, the property "there is a path between x and the boss that goes via at least two intermediaries" will be expressed by a regular expression $\Sigma^* \cdot report \cdot \Sigma^* \cdot report \cdot \Sigma^*$, where Σ is the set of all labels, which would result in an edge of form:



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,R,B), where R is a regular expression.

Combining these features We have analyzed each of these features in isolation, but of course they can be combined. A typical example of patterns that combine these features are CRPQs, that intuitively can be represented as a graph with node variables and where the edges between nodes can be labelled with regular expressions.

Example 3.1.1 *Consider the CRPQ* $Q(x,y) = \exists z\,(x,a^*,z) \wedge (z,b^*,y) \wedge (y,ab,x)$*, over alphabet* $\Sigma = \{a,b\}$*. Its representation as a graph pattern with variables in the nodes and regular expressions in the edges is as follows:*



3.2 Formal Definition

As explained in the previous section, 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, i.e., marked nulls for graph nodes;
- label variables, i.e., 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 (finite) labeling alphabet Σ was defined as a labeled graph, (N,E), where $N \subseteq \mathbf{V}$ is the finite 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 3.2.1 (Graph Pattern) *A* graph pattern *over finite alphabet Σ is a pair $\pi = (N,E)$ where*

- $N \subseteq \mathbf{V} \cup \mathcal{V}_{\text{node}}$ *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.* □

3.2.1 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 $L \in \text{REG}(\Gamma)$ 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 graph 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 $n \in \mathbf{V}$; 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.

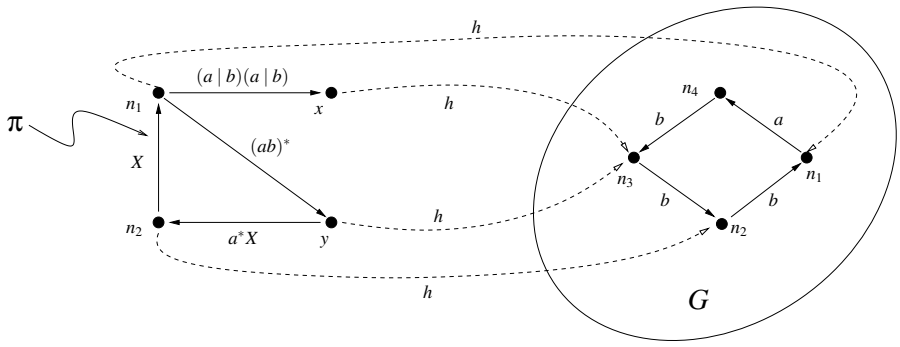


Figure 3.1: A homomorphism $h : \pi \to G$

Example 3.2.2 *An illustration is given in Fig. 3.1: a homomorphism is defined by mapping label variable X to label 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* $(n_1,(ab)^*,y)$ *can also be 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) *can be mapped into* (n_3,b,n_2)*, since b is in the language denoted by* a^*b*.* □

Remark. We have defined homomorphisms as pairs (h_1,h_2), where h_2 maps label variables to labels in the alphabet Σ of a graph. However, we could also have defined h_2 so that it maps label variables to any regular language in Σ^*. Constructs similar to this alternative definition have been studied in the context of programming languages (see e.g. [Aho, 1990]). We have chosen our simpler definition for two reasons. First, it is more natural for database applications, and follows the line of previous work in the area (c.f. [Cruz et al., 1987]). But second, having this extended definition would undoubtedly make graph patterns more difficult to work with, as even augmenting RPQs with this functionality leads to several undecidability issues [Freydenberger, 2013].

3.2.2 Graph patterns as queries

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, i.e., incomplete descriptions of documents. In our case, we have seen how to use graph patterns to represent incomplete graph data, but these patterns can of course be viewed as standard graph database queries, that return tuples of node variables. This is explained as follows.

We adopt the convention that patterns used in 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 $\varphi(\bar{z}) = \bigwedge_{i\leq m}(x_i,L_i,y_i)$, 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, in Chapter 4, we shall extend it to graph patterns). Given a graph database $G = (N, E)$ with $N \subset \mathbf{V}$, 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.

Example 3.2.3 *Consider again the example in Fig. 3.1 and the homomorphism described in Example 3.2.2. 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, when viewed as the graph query (ξ, x, y), corresponds to the CRPQ :*

$$\varphi(x,y) = (z_1, (a|b)(a|b), x) \wedge (z_1, (ab)^*, y) \wedge (y, a^*b, z_2) \wedge (z_2, b, z_1)$$

If it is evaluated in graph G shown in Fig. 3.1, 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. □

3.3 Classification

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 $\mathcal{P}^{\mathrm{nv,lv,re}}$ (all are present).

Of course $\mathcal{P}$ is the class of graph databases (N, E) with $N \subseteq \mathbf{V}$ and $E \subseteq N \times \Sigma \times N$, and $\mathcal{P}^{\mathrm{nv,lv,re}}$ is the class of all graph patterns as in Definition 3.2.1 with $N \subseteq \mathbf{V} \cup \mathcal{V}_{\mathrm{node}}$ and $E \subseteq N \times \mathrm{REG}(\Sigma \cup \mathcal{V}_{\mathrm{lab}}) \times N$. We now examine some others.

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

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

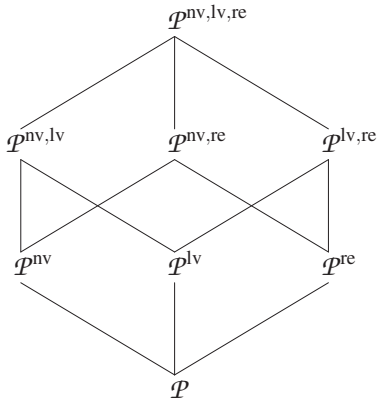


Figure 3.2: Relationships between classes of graph patterns

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

Since patterns from $\mathcal{P}^{\text{nv}}$ can be represented by relational naive tables, one can ask whether it is possible to reuse all the machinery developed for naive tables to the study of these patterns, and in particular, whether naive query evaluation should work for them. We will see in Chapter 4 that this is indeed true. However, this will turn out to be the largest class for which such naive evaluation works, as we shall demonstrate later.

3.3.1 Comparing features of graph patterns

Coming back to our three features of graph patterns, it is natural to ask whether all are necessary, or some are expressible with others. In this section we compare them in terms of their expressive power, and 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 3.3.1 *Adding each new feature to graph patterns strictly increases their expressiveness: in other words, $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\sigma'}$ is equivalent to $\sigma \subseteq \sigma'$.*

The relationships mentioned in Theorem 3.3.1 are summarized in Figure 3.2.

The proof of this Theorem relies on the following result. It provides examples of patterns in each of $\mathcal{P}^{\text{nv}}$, $\mathcal{P}^{\text{lv}}$, $\mathcal{P}^{\text{re}}$ such that the set of all graphs represented by these patterns cannot be represented by any pattern using any of the remaining two features.

Lemma 3.3.2 *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.*

Proof: We begin by proving **statement 1**. 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: $\bigcap\{N_G \mid G = (N_G,E_G) \text{ and } G \in [\![\pi']\!]_\Sigma\} = \emptyset$. 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 node, and such a node must belong to every graph G in $[\![\pi']\!]_\Sigma$.

Now we prove **statement 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 below the following property: 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 $|\rho|$. Note that this property immediately yields a contradiction, since 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.

  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 $\pi' \in \mathcal{P}^{\mathrm{nv,lv}}$, 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 3**, we prove that there exists a pattern π in $\mathcal{P}^{\mathrm{lv}}$ over alphabet $\Sigma = \{a, b\}$, such that $[\![\pi]\!]_\Sigma \neq [\![\pi']\!]_\Sigma$ for all patterns π' in $\mathcal{P}^{\mathrm{nv,re}}$ over Σ. We use the following claim:

Claim 3.3.3 *Let π be a pattern in $\mathcal{P}^{\mathrm{nv,re}}$ 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 $\mathcal{P}^{\mathrm{nv,re}}$, 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 now show that h_1'' is a homomorphism from π into G''. It is clear that h_1'' maps nodes 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 above, 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.

- 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 3.3.3, 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 3.3.3 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. □

Proof of Theorem 3.3.1: Using this lemma, we can now present the proof of the Theorem.

($\Leftarrow$): From the definition, it is clear that $\sigma \subseteq \sigma'$ implies $\mathcal{P}^\sigma \preceq \mathcal{P}^{\sigma'}$.

($\Rightarrow$): To prove that $\mathcal{P}^\sigma \preceq \mathcal{P}^{\sigma'}$ implies $\sigma \subseteq \sigma'$, assume for the sake of contradiction that for some σ, σ' it is the case that $\mathcal{P}^\sigma \preceq \mathcal{P}^{\sigma'}$, but it is not the case that $\sigma \subseteq \sigma'$. Then there exists an element of $\{\text{nv}, \text{lv}, \text{re}\}$ that belongs to σ but not to σ'. It follows from statements (1, 2 and 3) of Lemma 3.3.2 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 shows that $\mathcal{P}^\sigma \not\preceq \mathcal{P}^{\sigma'}$, which is a contradiction.

3.3.2 Codd Patterns and their classification

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 ones, and describe classes of patterns for which adding variables under Codd interpretation increases expressiveness.

Proposition 3.3.4

- *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 $[\![\pi]\!]_\Sigma$.

Finally 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 3.3.2, 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 above 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 $\{\text{lv}\}$ or $\{\text{lv}, \text{re}\}$, 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 $\mathcal{P}^{\sigma} \preceq \mathcal{P}^{\text{re}}$, and then either $\mathcal{P}^{\sigma} \prec \mathcal{P}^{\text{re}}$, or $\mathcal{P}^{\sigma} \sim \mathcal{P}^{\text{re}}$. However, any of these two facts contradicts Theorem 3.3.1.

Next, assume that σ contains nv. To prove that $\mathcal{P}^{\sigma}_{\text{Codd}}$ is not equally expressive as $\mathcal{P}^{\sigma}$ we shall prove a more general statement: There exists a pattern π in $\mathcal{P}^{\text{nv}}$ over alphabet $\Sigma = \{a\}$, such that $[\![\pi]\!]_\Sigma \neq [\![\pi']\!]_\Sigma$ for all π' in $\mathcal{P}^{\text{nv,lv,re}}_{\text{Codd}}$ 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 $\mathcal{P}^{\text{nv,lv,re}}_{\text{Codd}}$ 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 $\mathcal{P}^{\text{nv,lv,re}}_{\text{Codd}}$ 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 3.3.4.

□

Chapter 4

Answering Queries Over Graph Patterns

Having studied the basics of Graph Patterns, the next step is to analyze how to query them. The goal of this chapter is to study the complexity – both data and combined – of query answering over graph patterns. As for our query language, apart from the usual RPQs and CRPQs we take the study one step further, and provide complexity bounds for the (more general) language of queries given by graph patterns. Let us begin with some terminology.

4.1 Key Definitions

Graph patterns generally represent an infinite set of graph databases, and thus we cannot pose queries directly over them. As for other data models [Arenas et al., 2010, Barceló et al., 2010b, Fagin et al., 2005a, Imielinski and Lipski, 1984, Lenzerini, 2002], one looks instead for answers that are independent of the way in which the missing parts of patterns are interpreted, the so-called certain answers.

4.1.1 Certain Answers

To formally define certain answers in our context, 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, and so are queries based on graph patterns. For them, we can define their certain answers on graph patterns in the standard way:

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

Although somewhat unintuitive at the first glance, the labeling alphabet can make a difference in finding certain answers. For this reason we have explicitly included the alphabet

in our definition of certain answers. As usual, if Σ is clear from the context, we write simply $\text{CERTAIN}(Q,\pi)$.

Example 4.1.1 *The labeling alphabet can make a difference in finding certain answers. Consider a pattern π 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 $\varphi() = (x,ab,y)$. Then* $\text{CERTAIN}_{\{a,b\}}(Q,\pi) = \texttt{true}$*: whether X is a or b, there is a path labeled ab. However,* $\text{CERTAIN}_{\{a,b,c\}}(Q,\pi) = \texttt{false}$ *(by setting $X = c$).* □

4.1.2 Relationship between certain answers and implication of patterns

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

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

Proof: Let $Q = (\xi,\bar{x})$ be a graph query over Σ, and π a graph pattern. Assume that for some tuple $\bar{v}$ we have that $\pi \models \xi[\bar{v}/\bar{x}]$. Then $[\![\pi]\!] \subseteq [\![\xi[\bar{v}/\bar{x}]]\!]$. This means that for every graph G such that there is a homomorphism $h : \pi \to G$ there must also be a homomorphism from $\xi[\bar{v}/\bar{x}]$ to G. In other words, $\bar{v}$ belongs to $Q(G)$. Since G was arbitrarily chosen, we have that $\bar{v}$ belongs to $\text{CERTAIN}(Q,\pi)$. On the other hand, if a tuple $\bar{v}$ belongs to $\text{CERTAIN}(Q,\pi)$, then for every graph G in $[\![\pi]\!]$ (i.e., that there is a homomorphism from π to G), it must be the case that $\bar{v}$ belongs to $Q(G)$, which by definition means that there must be a homomorphism from $\xi[\bar{v}/\bar{x}]$ to G. This entails that $[\![\pi]\!] \subseteq [\![\xi[\bar{v}/\bar{x}]]\!]$, which was to be shown. □

For Boolean graph queries $Q = (\xi,())$ with the empty tuple of output variables (i.e., `true`/`false` queries), Lemma 4.1.2 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 for patterns that use regular expressions in their edge labels.

4.1.3 Problem definition

We are now ready to formally state our object of study in this chapter: the combined complexity of query answering. The problem we deal with is as follows:

PROBLEM:	PATTERN CERTAIN ANSWERS
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 $\bar{v} \in \text{CERTAIN}(Q, \pi)$?

We also study the *data* complexity of certain answers, 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 $\lvert\bar{x}\rvert = k$.

PROBLEM:	PATTERN CERTAIN ANSWERS(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.

4.1.4 Organization of this chapter

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.

We begin (Section 4.2) by examining up to what extent can results from relational databases be used in our context. We shall see that some results can be obtained for query answering in this fashion, but that relational techniques fall short when providing bounds for the general problem. We then devote Section 4.3 to show general upper bounds for both combined and data complexity, for the case when both the query and the input can be arbitrary patterns. Next, in order to provide a finer analysis on the impact of various features of graph patterns in query answering, we adopt CRPQs as our benchmark language (recall that CRPQs can be viewed as graph queries $(\xi, \bar{x})$ with $\xi \in \mathcal{P}^{\text{nv,re}}$), and provide exact complexity bounds for CRPQs over all classes of patterns. This is done in Section 4.4.

4.2 Using Naive Evaluation: The Relational Case

Some classes of patterns can be represented as naive tables, perhaps with constraints. The schema of these relational representations consists of binary relation symbols E_a, for each $a \in \Sigma$. Then, for instance, a pattern π in $\mathcal{P}^{\text{nv}}$ over Σ can be interpreted as a naive table I_π of this schema: The interpretation of symbol E_a in this structure contains all pairs (p, q) of nodes in π such that there is an edge labeled a from p to q in π.

Example 4.2.1 *Consider the graph pattern π over alphabet $\Sigma = \{a,b\}$, that consist of edges (n_1,a,n_2), (n_2,b,y), (n_2,b,x) and (n_1,a,x). The following is the relational representation of π:*

E_a

n_1	n_2
n_1	x

E_b

n_2	x
n_2	y

Note that this is, in fact, a naive table.

Other class of patterns can be represented, in a similar fashion, if we add constraints to the relational representations. For example, patterns from $\mathcal{P}^{\text{nv,lv}}$ are represented as relational naive tables with an additional constraint that the interpretation for label variables must come from the labeling alphabet Σ, which can easily be coded as an inclusion constraint. The goal of this section is to see up to what extent it is possible to reuse algorithms and bounds from relational incomplete databases to compute certain answers over graph patterns.

4.2.1 Patterns in $\mathcal{P}^{\text{nv}}$, or naive tables

We have mentioned that patterns in $\mathcal{P}^{\text{nv}}$ are nothing more than relational naive tables, assuming the standard relational representation of graph databases. This representation automatically gives us tractable bounds for data complexity of answering CRPQs over patterns in $\mathcal{P}^{\text{nv}}$, since it is well known that CRPQs can be expressed in datalog, and computing the certain answers of a datalog program over a naive table is known to be in PTIME (see e.g. [Abiteboul and Duschka, 1998]). However, with a little bit more of effort, one can show that in fact all graph patterns, when viewed as queries, can be expressed in datalog, which gives us the following result.

Proposition 4.2.2 PATTERN CERTAIN ANSWERS(Q) *is in* PTIME *when restricted to patterns in $\mathcal{P}^{\text{nv}}$.*

Proof: It is clear that every graph query in $\mathcal{P}^{\text{nv,re}}$ can be transformed into an equivalent datalog program over the relational representation that we have previously discussed. Let now $(\xi,\bar{x})$ be an arbitrary graph query, and let $\mathcal{W}$ be the set of label variables in ξ. note then that $(\xi,\bar{x})$ is equivalent to the union of the following set of graph queries: $(\nu(\xi),\bar{x})$, for every valuation $\nu : \mathcal{W} \to \Sigma$. We can then construct a datalog program P_ξ that is equivalent to $(\xi,\bar{x})$ by taking the union of all datalog programs $P_{\nu(\xi)}$ that are equivalent to $(\nu(\xi),\bar{x})$, in a way such that the answers for P_ξ are exactly the union of the answers of each $P_{\nu(\xi)}$. □

Since even the standard evaluation of datalog programs over complete relational databases is known to be EXPTIME-complete, the same approach adopted for proving Proposition 4.2.2 gives us an exponential upper bound for the combined complexity of certain answers. This is,

as expected, not optimal, and in fact much lower bounds have been found already for CRPQs. For the case of arbitrary graph queries, it turns out that one can also match the complexity bounds for querying CRPQs over standard graph databases.

The key idea to show this fact is to disregard using relational representations, but instead perform naive evaluation directly over the graph pattern. Formally, let Q be a graph query and π a pattern in $\mathcal{P}^{\text{nv}}$ over Σ. We show next that the set $\text{CERTAIN}_\Sigma(Q,\pi)$ can be computed by directly evaluating Q over π, treating node variables as if they were ordinary node ids, and then discarding from the answers all tuples containing node variables.

Lemma 4.2.3 *Let Q be a graph query and π a pattern in $\mathcal{P}^{\text{nv}}$, both over an alphabet Σ. Then the naive evaluation of Q over π corresponds to* $\text{CERTAIN}_\Sigma(Q,\pi)$.

Proof: First, by definition, the relational representation of every graph pattern π in $\mathcal{P}^{\text{nv}}$ is homomorphically contained in the relational representation of any graph database in $\text{Rep}(\pi)$. Second, since datalog queries are preserved under relational homomorphisms, and from Proposition 4.2.2 every graph query can be expressed in datalog, it follows that graph queries are preserved under relational homomorphisms, and thus naive evaluation works even in the graph database scenario. □

It is not difficult to show that the problem of evaluating a graph query over a graph database G is in NP (one just chooses a proper graph homomorphism), and hence performing a naïve evaluation (and, therefore, computing certain answers) of a graph query over a pattern in $\mathcal{P}^{\text{nv}}$ is in NP. The problem is clearly also NP-hard, even over patterns in $\mathcal{P}$, as it contains as a subinstance the problem of conjunctive query evaluation over graphs.

Proposition 4.2.4 PATTERN CERTAIN ANSWERS *is* NP*-complete when restricted to graph queries over patterns in $\mathcal{P}^{\text{nv}}$.*

Thus, for the case of patterns in $\mathcal{P}^{\text{nv}}$ the complexity of querying patterns is, remarkably, not greater than the complexity of evaluating conjunctive queries over naive tables. As we have explained, this is due to the fact that naive evaluation works for this classes of patterns. We shall see in the remainder of this Chapter that as soon as we add regular expressions or label variables to our patterns, then the possibility of doing naive evaluation appears to be lost. Consequently, this carries over an increase in combined complexity, and, for the case of data complexity, we immediately loose tractability.

Other relational representations. Unfortunately, for other classes of patterns, 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 $\mathcal{P}^{\text{nv,lv}}$ patterns. Same applies for data complexity.

The results in this section will be revisited in Chapter 6 when studying how to perform query answering in the context of data exchange: indeed, we see that patterns in $\mathcal{P}^{\text{nv}}$ will be the natural candidates to materialize when exchanging information under some interesting classes of data exchange settings, allowing us to reuse the techniques visited here in order to compute queries in a data exchange context.

4.3 General Upper Bounds

Our next task is to analyze what happens in the general scenario. The main results of this section are the general upper bounds for the certain answers problem, for arbitrary graph queries over arbitrary graph patterns, for both combined and data complexity. We have discussed that relational representations can only provide meaningful bounds for patterns in $\mathcal{P}^{\text{nv}}$, and we have argued that new techniques are needed in order to solve this problem for more expressive classes of patterns. Therefore, instead of using relational techniques, we exploit the connection between certain answers and pattern implication stated in Lemma 4.1.2, and look at the results of pattern implication by [Calvanese et al., 2000b].

4.3.1 Combined Complexity

We begin with combined complexity. It was shown in [Calvanese et al., 2000b] that the pattern implication problem for CRPQs is EXPSPACE-complete. By Lemma 4.1.2 this gives us with little effort that answering queries in $\mathcal{P}^{\text{nv,re}}$ over patterns in $\mathcal{P}^{\text{nv,re}}$ is EXPSPACE-complete. Using essentially the same techniques as in [Calvanese et al., 2000b], we can prove that the previous upper bound extends beyond CRPQs.

Proposition 4.3.1 PATTERN CERTAIN ANSWERS *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 (that is, 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 $\mathcal{A}_\pi^\nu$, 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 $\mathcal{A}_\xi^{\nu'}$, 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 $\mathcal{B} = \mathcal{A}_\pi^\nu \cap \bigcap_{\nu' : \mathcal{W} \to \Sigma} \mathcal{A}_\xi^{\nu'}$ 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. □

4.3.2 Data Complexity

Next we turn to data complexity of certain answers. In contrast to the relational case, where answering positive queries over naive tables is usually tractable, the data complexity of query answering jumps to coNP-hard as soon as regular expressions or label variables are involved in patterns, i.e., as soon as we loose the possibility of performing naive evaluation.

Let us quickly show the proof for the case of patterns in $\mathcal{P}^{\text{re}}$. We extend this result to $\mathcal{P}^{\text{lv}}$ in the next section. These will also be tightened significantly in Chapter 5

Proposition 4.3.2 *There is a CRPQ Q such that* PATTERN CERTAIN ANSWERS(Q) *is* coNP*-hard over patterns in $\mathcal{P}^{\text{re}}$.*

Proof: We use 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, i.e., (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 $\{a,r,g,b\}$ 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 $Q() = (x,rar|gag|bab,y)$ over π_G is `true` if and only if G is not 3-colorable. □

In order to prove that the upper bound again extends to arbitrary queries, 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, and essentially the same ideas explored for answering XML patterns over XML incomplete information [Barceló et al., 2010b]. More precisely, to show that a particular tuple of nodes does not belong to the certain answers of a graph query Q over a graph pattern π, one can just guess a graph G in $\text{Rep}(\pi)$, and then evaluate Q directly over G (which is polynomial in data complexity). All that one really needs to show is that one can always find a suitable graph G of polynomial size w.r.t Q. This is what we do next, but first we need some definitions.

4.3.2.1 Canonical graph databases

Let π be a graph pattern, and σ an assignment from the nodes of π into $\mathbf{V}$ 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 $\text{REG}(\Sigma \cup \mathcal{V}_{\text{lab}})$ $(1 \leq i \leq m)$. 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 $\nu : \mathcal{V}_{\text{lab}} \to \Sigma$ such that the following holds:

- G consists of m simple paths, one for each edge in π, which are node and edge disjoint, i.e. 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 4.3.3 *For each graph query Q and tuple $\bar{n}$ of node ids in π, it is the case that $\bar{n} \notin \text{CERTAIN}(Q,\pi)$ if and only if there is a a graph database G over Σ that is canonical for π and such that $\bar{n} \notin Q(G)$.*

4.3.2.2 Proof of coNP upper bound

Finally we have the appropriate tools to prove the upper bound for data complexity.

Proposition 4.3.4 PATTERN CERTAIN ANSWERS(Q) *is in* coNP *for arbitrary graph queries over arbitrary graph patterns.*

Proof: 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 G. (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 $\nu : \mathcal{V}_{\text{lab}} \to \Sigma$, for some canonical assignment σ (Claim 4.3.3). 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 $\bigvee_{\{\nu \mid \nu:\mathcal{V}_{\text{lab}} \to \Sigma\}} Q_\nu$, 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 $\mathcal{P}^{\text{nv,re}}$. 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 union of CRPQs 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, see [Libkin, 2004] a precise definition – 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 presented in Section 4.2) 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 $G \models Q' \Leftrightarrow \mathcal{S}_G \models \varphi$, 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* (c.f., [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 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 $\lambda(\rho_{\rightarrow}^i)$, where $\rho_{\rightarrow}^i = n_i a_i n_{i+1} \cdots a_\ell n_{\ell+1}$; and
- α_2^i is the rank-k type of the word $\lambda(\rho_{\leftarrow}^i)$, where $\rho_{\leftarrow}^i = n_0 a_0 \cdots a_{i-2} n_{i-1} a_{i-1}$.

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 $\rho' = n_0 a_0 n_1 \cdots a_{i-1} n_i a_j n_{j+1} \ldots n_\ell a_\ell n_{\ell+1}$, 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 $(G,\bar{n}) \models \varphi$ if and only if $(G',\bar{n}) \models \varphi$ (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 $\text{CERTAIN}(Q,\pi) = \texttt{false}$. Let π be a graph pattern over Σ. If Q contains some node id that does not appear in π then clearly $\text{CERTAIN}(Q,\pi) = \texttt{false}$. 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]. □

4.4 Full Complexity Analysis

So far we have provided upper bounds for the general problem of query answering, looking at both combined and data complexity; and we have also provided bounds for querying patterns in $\mathcal{P}^{\text{nv}}$, since in this case most of these result follow rather swiftly from previous work on incomplete information over relational databases.

For combined complexity, the picture shows that the problem is EXPSPACE-complete as soon as regular expressions are allowed in the patterns that are to be queried, but the question of the impact of label variables is still open. On the other hand, for data complexity we have argued that the problem becomes CONP-hard on data complexity as soon as any of regular expressions or label variables are allowed in the input patterns.

In this section we set up to complete the picture in full detail, showing upper and lower bounds for querying all of the 8 classes of graph patterns. We use CRPQs, instead of graph queries, as our benchmark query language. This will allow us to compare with most of the previous results in the literature, and at the same time it helps to maintain the readability of the dissertation.

4.4.1 Combined Complexity

The general EXPSPACE upper bound from Proposition 4.3.1 obviously translate into a general EXPSPACE upper bound for CRPQs. A matching lower bound was given by Calvanese et al. for patterns in $\mathcal{P}^{\text{nv,re}}$ (see Theorem 6 in [Calvanese et al., 2000b], which proves that containment of CRPQs is EXPSPACE-complete). By slightly adapting the reduction in that paper one can also show that the problem remains EXPSPACE-hard over the class of patterns in $\mathcal{P}^{\text{re}}$.

On the other hand, in Section 4.2 we established NP upper bounds for the combined complexity of query answering. 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.

To complete the picture it remains to see what happens when one restricts from using regular expressions, but add label variables, i.e. the cases when our data is a pattern in $\mathcal{P}^{\text{lv}}$ or $\mathcal{P}^{\text{nv,lv}}$.

Lemma 4.4.1 PATTERN CERTAIN ANSWERS *is* Π_2^p*-complete, when restricted to patterns in* $\mathcal{P}^{\text{lv}}$ *or* $\mathcal{P}^{\text{nv,lv}}$

Proof: 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 above, the latter can be solved in coNP.

For the proof of Π_2^p-hardness for the class $\mathcal{P}^{\text{lv}}$ we reduce from the problem of $\forall\exists$ POSITIVE 1-3 3-SAT, which is known to be Π_2^p-hard [Björklund et al., 2007]. This problem is defined as follows: A set of clauses $\{C_1,\ldots,C_p\}$ is given, each of which has exactly 3 distinct Boolean variables from the disjoint union of $\{x_1,\ldots,x_m\}$ and $\{y_1,\ldots,y_t\}$. No variable is negated. The problem asks whether for each assignment for $\{x_1,\ldots,x_m\}$, there exists an assignment for $\{y_1,\ldots,y_t\}$ such that each clause C_i contains exactly one true variable.

Let $\varphi := \forall x_1 \cdots \forall x_m \exists y_1 \ldots \exists y_t \{C_1,\ldots,C_p\}$ be an instance of $\forall\exists$ positive 1-3 3-SAT. From φ we construct in polynomial time an alphabet Σ, a pattern $\pi_\varphi \in \mathcal{P}^{\text{lv}}$ and a CRPQ Q_φ, both over Σ, such that for each assignment for $\{x_1,\ldots,x_m\}$, there exists an assignment for $\{y_1,\ldots,y_t\}$ such that each clause C_i contains exactly one true variable if and only if $\text{CERTAIN}(Q_\varphi,\pi_\varphi) = \texttt{true}$.

We assume, without loss of generality, that φ contains clauses neither of the form $(x_j \vee x_k \vee x_\ell)$, for $1 \le j < k < \ell \le m$, nor of the form $(x_j \vee x_k \vee y_\ell)$, for $1 \le j < k \le m$ and $1 \le \ell \le t$. Indeed,

it is clear that if φ contains a clause of any of these forms, then there exists an assignment for $\{x_1,\ldots,x_m\}$ such that for no assignment for $\{y_1,\ldots,y_t\}$ it is the case that each clause C_i contains exactly one true variable (this is the case, in particular, for the assignment that makes true each variable in $\{x_1,\ldots,x_m\}$). This means that if φ contains clauses of any if these forms, then it does not belong to $\forall\exists$ positive 1-3 3-SAT. Further, it is easy to verify in polynomial time whether φ contains clauses of these forms.

The alphabet Σ over which π_φ is defined is:

$$\{C_1,\ldots,C_p,(++),(+-),(-+),(--),P,N,V,S,0,1\}.$$

The pattern π_φ uses label variables in the set $\{X_1,\ldots,X_m\}$, and it is defined as follows.

First, π_φ contains node ids $n_\bot$ and $n_\top$ that represent, respectively, the Boolean values `true` and `false`. Intuitively, the fact that the node variable y_i ($1 \leq i \leq t$) of Q_φ, as defined below, is assigned to node $n_\bot$ (resp., $n_\top$) in a graph database $G \in [\![\pi_\varphi]\!]$, represents a valuation of φ that assigns value `false` (resp., `true`) to y_i.

In order to identify $n_\bot$ and $n_\top$ in π_φ, we mark the first node id with a self-loop labeled N (for *Negative* value) and the second one with a self-loop labeled P (for *Positive* value).

Second, for each clause C_i, $1 \leq i \leq p$, the pattern π_φ contains a subpattern π_{C_i} that is defined by cases.

1. Assume first that $C_i = (y_j \vee y_\ell \vee x_k)$, where $1 \leq j < \ell \leq t$ and $1 \leq k \leq m$. Then π_{C_i} consists of pairwise distinct node ids

   $$n^i_\bot, n^i_\top, m^{i,1}_\bot, m^{i,2}_\bot, m^{i,1}_\top, m^{i,2}_\top, t_i.$$

   Intuitively, the fact that the variable z^1_i of Q_φ, as defined below, is mapped into node id $n^i_\bot$ (resp., $n^i_\top$) in some graph database $G \in [\![\pi_\varphi]\!]$, represents a valuation for φ that assigns value `false` (resp., `true`) to the first variable of clause C_i (that is, to y_j).

   In the same way, the fact that the variable z^2_i of Q_φ, as defined below, is mapped into node id $m^{i,1}_\bot$ (resp., $m^{i,2}_\top$) in some graph database $G \in [\![\pi_\varphi]\!]$, represents a valuation for φ that assigns value `false` (resp., `true`) to both the first and the second variable of clause C_i (that is, to y_j and y_ℓ). And, analogously, the fact that the variable z^2_i of Q_φ, as defined below, is mapped into node id $m^{i,2}_\bot$ (resp., $m^{i,1}_\top$) in some graph database $G \in [\![\pi_\varphi]\!]$, represents a valuation for φ that assigns value `true` to the first variable, y_j, of clause C_i, and value `false` to the second one, y_ℓ (resp., value `false` to the first variable of clause C_i and value `true` to the second one).

   The following edges are the only edges that exist in between the node ids of π_{C_i} plus $\{n_\bot, n_\top\}$:

- Both $n^i_\bot$, $n^i_\top$ and t_i have a self-loop labeled C_i. This self-loop permits identifying these nodes as part of the pattern π_{C_i}.
- There are edges labeled V from $n^i_\bot$ into both $m^{i,1}_\bot$ and $m^{i,1}_\top$. These edges represent the fact that $n^i_\bot, m^{i,1}_\bot, m^{i,1}_\top$ are node ids that encode valuations for φ that assign value `false` to the first variable, y_j, of clause C_i.
- There are edges labeled V from $n^i_\top$ into both $m^{i,2}_\bot$ and $m^{i,2}_\top$. These edges represent the fact that $n^i_\top, m^{i,2}_\bot, m^{i,2}_\top$ are node ids that encode valuations for φ that assign value `true` to the first variable, y_j, of clause C_i.
- There are edges labeled S from $n^i_\bot, m^{i,1}_\bot$ and $m^{i,2}_\bot$ into $n_\bot$. These edges represent the fact that $n^i_\bot$ (resp., $m^{i,1}_\bot$ and $m^{i,2}_\bot$) are node ids that encode valuations for φ that assign value `false` to the first (resp., second) variable of clause C_i.
- There are edges labeled S from $n^i_\top, m^{i,1}_\top$ and $m^{i,2}_\top$ into $n_\top$. These edges represent the fact that $n^i_\top$ (resp., $m^{i,1}_\top$ and $m^{i,2}_\top$) are node ids that encode valuations for φ that assign value `true` to the first (resp., second) variable of clause C_i.
- The node id $m^{i,1}_\bot$ has a self-loop labeled $(--)$, that represents that $m^{i,1}_\bot$ encodes valuations for φ that assign the value `false` to the first two variables of C_i (that is, to y_j and y_ℓ).
- The node id $m^{i,2}_\bot$ has a self-loop labeled $(+-)$, that represents that $m^{i,2}_\bot$ encodes valuations for φ that assign value `true` to the first variable, y_j, of C_i and value `false` to the second one, y_ℓ.
- The node id $m^{i,1}_\top$ has a self-loop labeled $(-+)$, that represents that $m^{i,1}_\top$ encodes valuations for φ that assign value `false` to the first variable, y_j, of C_i and value `true` to the second one, y_ℓ.
- The node id $m^{i,2}_\top$ has a self-loop labeled $(++)$, that represents that $m^{i,2}_\top$ encodes valuations for φ that assign the value `true` to the first two variables of C_i (that is, to y_j and y_ℓ).
- There are edges labeled X_k from each one of $m^{i,1}_\bot, m^{i,2}_\bot, m^{i,1}_\top$ and $m^{i,2}_\top$ into t_i. These edges represent the fact that the last variable of C_i is x_k.

Figure 4.1 shows how this pattern (together with all the edges that link this pattern to $n_\bot$ and $n_\top$) looks.

2. Assume now that $C_i = (y_j \vee y_\ell \vee y_k)$, where $1 \leq j < k < \ell \leq t$. Then π_{C_i} is defined exactly as in the previous case, except that now the node id t_i is removed, together with all the edges that point to it, and the 2 new node ids $p^i_\bot, p^i_\top$ are added, together with the edges that link them to the rest of π_{C_i} and the nodes $n_\bot$ and $n_\top$ that we mention below.

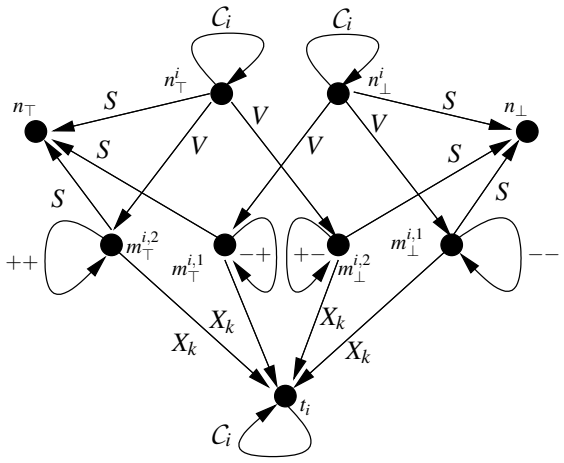


Figure 4.1: Pattern π_{C_i} for a clause C_i of the form $(y_j \vee y_\ell \vee x_k)$.

In this case, the fact that the variable z^3_i of Q_φ, as defined below, is mapped into node id $p^i_\bot$ (resp., $p^i_\top$) in some graph database $G \in [\![\pi_\varphi]\!]$, represents a valuation for φ that assigns value `false` (resp., `true`) to the third variable of clause C_i (that is, to y_k).

The pattern π_{C_i} contains the following edges linking the node ids in $\{p^i_\bot, p^i_\top\}$ to the rest of π_{C_i} and to the node ids $n_\bot$ and $n_\top$:

- The node ida $p^i_\bot$ and $p^i_\top$ and have a self-loop labeled C_i. This self-loop permits identifying these nodes as part of the pattern π_{C_i}.
- There is an edge labeled S from $p^i_\bot$ into $n_\bot$. This edge represents the fact that $p^i_\bot$ is the node id that encodes valuations for φ that assign value `false` to the third variable, y_k, of clause C_i.
- There is an edge labeled S from $p^i_\top$ into $n_\top$. This edge represents the fact that $p^i_\top$ is the node id that encodes valuations for φ that assign value `true` to the third variable, y_k, of clause C_i.
- There are edges labeled N from $m^{i,1}_\bot$, $m^{i,2}_\bot$, $m^{i,1}_\top$ and $m^{i,2}_\top$ into $p^i_\bot$. This edge also represents the fact that $p^i_\bot$ is the node id that encodes valuations for φ that assign value `false` to the third variable, y_k, of clause C_i.
- There are edges labeled P from $m^{i,1}_\bot$, $m^{i,2}_\bot$, $m^{i,1}_\top$ and $m^{i,2}_\top$ into $p^i_\top$. This edge also represents the fact that $p^i_\top$ is the node id that encodes valuations for φ that assign value `true` to the third variable, y_k, of clause C_i.

Figure 4.2 shows how this pattern (together with all the edges that link this pattern to $n_\bot$ and $n_\top$) looks.

Clearly, π_φ belongs to $\mathcal{P}^{\text{lv}}$ and can be constructed in polynomial time from φ.

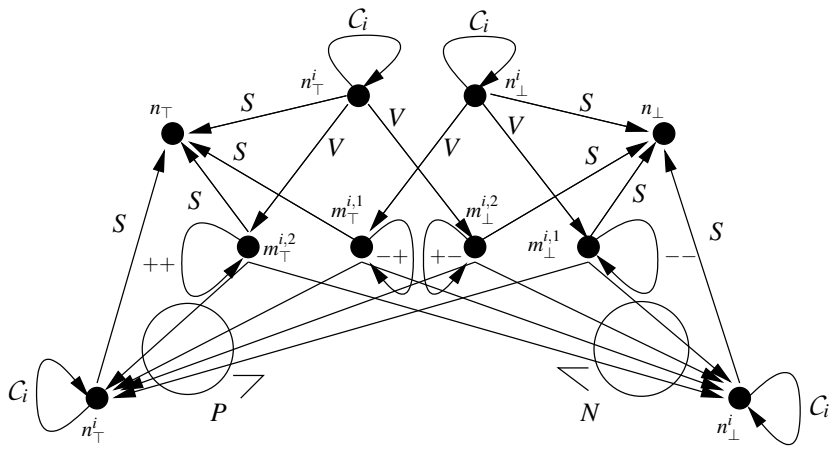


Figure 4.2: Pattern π_{C_i} for a clause C_i of the form $(y_j \vee y_\ell \vee x_k)$.

Now we construct, from φ, a CRPQ Q_φ over alphabet Σ that is defined as the existential closure of the following conjuncts: First, for each $1 \leq i \leq t$ a conjunct $(y_i, (P \cup N), y_i)$ that states that the node variable y_i is mapped into either $n_\bot$ or $n_\top$. Second, for each $1 \leq i \leq p$, a conjunction θ_{C_i} that is again defined by cases:

1. Assume first that $C_i = (y_j \vee y_\ell \vee x_k)$, where $1 \leq j < \ell \leq t$ and $1 \leq k \leq m$. Then θ_{C_i} is

$$(z_i^1, S, y_j), (z_i^2, S, y_\ell), (z_i^1, C_i \cdot V, z_i^2), (z_i^2, \mathcal{R}, z_i^3),$$

   where the regular language $\mathcal{R}$ is defined as follows:

$$\mathcal{R} := \Big(\bigcup_{a \in \Sigma \setminus \{0,1\}} a \Big) \cdot C_i \cup \big((--) \cdot 1 \cup (-+) \cdot 0 \cup (+-) \cdot 0\big) \cdot C_i.$$

2. Assume second that $C_i = (y_j \vee y_\ell \vee y_k)$, where $1 \leq j < \ell < k \leq t$. Then θ_{C_i} is

$$(z_i^1, S, y_j), (z_i^2, S, y_\ell), (z_i^3, S, y_k), (z_i^1, C_i \cdot V, z_i^2), (z_i^2, \mathcal{R}, z_i^3),$$

   where the regular language $\mathcal{R}$ is defined as follows:

$$\mathcal{R} := \big((--) \cdot P \cup (-+) \cdot N \cup (+-) \cdot N\big) \cdot C_i.$$

Clearly, Q_φ can be constructed in polynomial time from φ.

We prove next that φ belongs to $\forall\exists$ POSITIVE 1-3 3SAT if and only if $\text{CERTAIN}(Q_\varphi, \pi_\varphi) =$ `true`. Assume first that $\varphi = \forall x_1 \cdots \forall x_m \exists y_1 \ldots \exists y_t \{C_1, \ldots, C_p\}$ belongs to $\forall\exists$ POSITIVE 1-3 3SAT. Consider an arbitrary graph database $G \in [\![\pi_\varphi]\!]$. Then there exists a homomorphism $h : \pi_\varphi \to G$. Since π_φ does not contain node variables, we assume without loss of generality that h is a mapping from the label variables of π_φ, that is, $\{X_1, \ldots, X_m\}$, into Σ. We prove

next that Q_φ holds in G, which shows, in turn, that $\text{CERTAIN}(Q_\varphi, \pi_\varphi) = \texttt{true}$ (since G was arbitrarily chosen).

Let $\mathcal{X}$ be the set of label variables in $\{X_1, \ldots, X_m\}$ such that $h(X) = 1 \Leftrightarrow X \in \mathcal{X}$. (Notice that $\mathcal{X}$ can be empty). Consider the assignment κ for the propositional variables $\{x_1, \ldots, x_m\}$ of φ into Boolean values true and false, such that κ assigns value true to variable x_i if and only if $X_i \in \mathcal{X}$ ($1 \leq i \leq m$). Since φ belongs to $\forall\exists$ POSITIVE 1-3 3SAT, there is an assignment κ' for the propositional variables $\{x_1, \ldots, x_m, y_1, \ldots, y_t\}$ of φ into Boolean values true and false, such that κ' coincides with κ on $\{x_1, \ldots, x_m\}$ and κ' assigns value true to exactly one propositional variable in each clause C_i, $1 \leq i \leq p$.

Recall that the set of node variables of Q_φ is $\{y_1, \ldots, y_t, z_1^1, z_1^2, z_1^3, \ldots, z_p^1, z_p^2, z_p^3\}$. We define σ to be any mapping from $\{y_1, \ldots, y_t, z_1^1, z_1^2, z_1^3, \ldots, z_p^1, z_p^2, z_p^3\}$ into the node ids of G that satisfies the following:

- For each $1 \leq i \leq t$, $\sigma(y_i) = n_\perp$ if $\kappa'(y_i)$ is false, and $\sigma(y_i) = n_\top$ otherwise.
- If C_i is of the form $(y_j \vee y_\ell \vee x_k)$, for $1 \leq j < \ell \leq t$ and $1 \leq k \leq m$, then:
  1. $\sigma(z_i^1) = n_\perp^i$ if $\sigma(y_j) = n_\perp$, and $\sigma(z_i^1) = n_\top^i$ otherwise;
  2. $\sigma(z_i^2) = m_\perp^{i,1}$ if $\sigma(y_j) = \sigma(y_\ell) = n_\perp$, $\sigma(z_i^2) = m_\perp^{i,2}$ if $\sigma(y_j) = n_\top$ and $\sigma(y_\ell) = n_\perp$, $\sigma(z_i^2) = m_\top^{i,1}$ if $\sigma(y_j) = n_\perp$ and $\sigma(y_\ell) = n_\top$, and $\sigma(z_i^2) = m_\top^{i,2}$ otherwise;
  3. $\sigma(z_i^3) = t_i$.
- If C_i is of the form $(y_j \vee y_\ell \vee y_k)$, for $1 \leq j < \ell < k \leq t$, then:
  1. $\sigma(z_i^1) = n_\perp^i$ if $\sigma(y_j) = n_\perp$, and $\sigma(z_i^1) = n_\top^i$ otherwise;
  2. $\sigma(z_i^2) = m_\perp^{i,1}$ if $\sigma(y_j) = \sigma(y_\ell) = n_\perp$, $\sigma(z_i^2) = m_\perp^{i,2}$ if $\sigma(y_j) = n_\top$ and $\sigma(y_\ell) = n_\perp$, $\sigma(z_i^2) = m_\top^{i,1}$ if $\sigma(y_j) = n_\perp$ and $\sigma(y_\ell) = n_\top$, and $\sigma(z_i^2) = m_\top^{i,2}$ otherwise;
  3. $\sigma(z_i^3) = p_\perp^i$ if $\sigma(y_k) = n_\perp$, and $\sigma(z_i^3) = p_\top^i$ otherwise.

It is easy to see that assignment σ is well-defined. We prove next that $(G, \sigma) \models Q_\varphi$, and hence that Q_φ holds in G.

Recall that Q_φ consists of conjuncts $(y_i, (P \cup N), y_i)$, for each $1 \leq i \leq t$, and conjunctions θ_{C_i}, for each $1 \leq i \leq p$. Clearly, $(G, \sigma) \models (y_i, (P \cup N), y_i)$, for each $1 \leq i \leq t$. This is because $\sigma(y_i)$ is either $n_\perp$ and $n_\top$, which have self-loops labeled N and P, respectively. We prove next that $(G, \sigma) \models \theta_{C_i}$, for each $1 \leq i \leq p$.

Assume first that C_i is of the form $(y_j \vee y_\ell \vee x_k)$, for $1 \leq j < \ell \leq t$ and $1 \leq k \leq m$. Recall that in this case θ_{C_i} is defined as:

$$(z_i^1, S, y_j), (z_i^2, S, y_\ell), (z_i^1, \mathcal{C}_i \cdot V, z_i^2), (z_i^2, \mathcal{R}, z_i^3),$$

where the regular language $\mathcal{R}$ is defined as follows:

$$\mathcal{R} := (\bigcup_{a\in\Sigma\setminus\{0,1\}} a)\cdot C_i \cup \big((--)\cdot 1 \cup (-+)\cdot 0 \cup (+-)\cdot 0\big)\cdot C_i.$$

It is easy to see that, by definition, $(G,\sigma) \models (z_i^1, S, y_j), (z_i^2, S, y_\ell)$. This is because, in every possible case, $\sigma(z_i^1)$ (resp., $\sigma(z_i^2)$) corresponds to a node id that can access $\sigma(y_j)$ (resp., $\sigma(y_\ell)$) through an S-labeled edge. Further, clearly $(G,\sigma) \models (z_i^1, C_i \cdot V, z_i^2)$. This is because $\sigma(z_i^1)$ is either $n_\perp^i$ or $n_\top^i$, which have C_i-labeled self-loops, and in every possible case $\sigma(z_i^1)$ corresponds to a node id that can access $\sigma(z_i^2)$ through a V-labeled edge. We prove next by cases that $(G,\sigma) \models (z_i^2, \mathcal{R}, z_i^3)$:

1. Assume first that $h(X_i) \notin \{0,1\}$. Then clearly $(G,\sigma) \models (z_i^2, (\bigcup_{a\in\Sigma\setminus\{0,1\}} a)\cdot C_i, z_i^3)$. This is because $\sigma(z_i^3) = t_i$, t_i has a self-loop labeled C_i, and in every possible case there is an edge from $\sigma(z_i^2)$ into $\sigma(z_i^3) = t_i$ labeled with $h(X_i) \in \bigcup_{a\in\Sigma\setminus\{0,1\}} a$.

2. Assume second that $h(X_i) = 1$. Then $\kappa'(x_i)$ is true, and hence since κ' makes true exactly one propositional variable in C_i, it must be the case that $\kappa'(y_j) = \kappa'(y_\ell) =$ false, which implies that $\sigma(y_j) = \sigma(y_\ell) = n_\perp$. This implies that $\sigma(z_i^2) = m_\perp^{i,1}$, and hence that $(G,\sigma) \models (z_i^2, (--)\cdot 1 \cdot C_i, z_i^3)$. This is because $\sigma(z_i^2) = m_\perp^{i,1}$ has a self-loop labeled $(--)$, there is an edge from $\sigma(z_i^2)$ into $\sigma(z_i^3) = t_i$ labeled with $h(X_i) = 1$, and t_i has a self-loop labeled C_i.

3. The case when $h(X_i) = 0$ is similar and left to the reader.

In any case we conclude that $(G,\sigma) \models (z_i^2, \mathcal{R}, z_i^3)$.

Assume second that C_i is of the form $(y_j \vee y_\ell \vee y_k)$, for $1 \le j < \ell < k \le t$. Recall that in this case θ_{C_i} is defined as:

$$(z_i^1, S, y_j), (z_i^2, S, y_\ell), (z_i^3, S, y_k), (z_i^1, C_i \cdot V, z_i^2), (z_i^2, \mathcal{R}, z_i^3),$$

where the regular language $\mathcal{R}$ is defined as follows:

$$\mathcal{R} := \big((--)\cdot P \cup (-+)\cdot N \cup (+-)\cdot N\big)\cdot C_i.$$

As in the previous case, it is easy to see that, by definition, $(G,\sigma) \models (z_i^1, S, y_j), (z_i^2, S, y_\ell), (z_i^3, S, y_k)$. This is because, in every possible case, $\sigma(z_i^1)$ (resp., $\sigma(z_i^2)$ and $\sigma(z_i^3)$) corresponds to a node id that can access $\sigma(y_j)$ (resp., $\sigma(y_\ell)$ and $\sigma(y_k)$) through an S-labeled edge. Further, clearly $(G,\sigma) \models (z_i^1, C_i \cdot V, z_i^2)$. This is because $\sigma(z_i^1)$ is either $n_\perp^i$ or $n_\top^i$, which have C_i-labeled self-loops, and in every possible case $\sigma(z_i^1)$ corresponds to a node id that can access $\sigma(z_i^2)$ through a V-labeled edge. We prove next by cases that $(G,\sigma) \models (z_i^2, \mathcal{R}, z_i^3)$:

1. Assume first that $\kappa'(x_k) = \text{true}$. This implies that $\sigma(x_k) = p^i_\top$, and hence that $(G,\sigma) \models (z^2_i, (--)\cdot P \cdot \mathcal{C}_i, z^3_i)$. Indeed, since κ' makes true exactly one variable in C_i it must be the case that $\kappa'(y_j) = \kappa'(y_\ell) = \text{false}$, and hence $\sigma(y_j) = \sigma(y_\ell) = n_\bot$. This implies that $\sigma(z^2_i) = m^{i,1}_\bot$, and hence that $(G,\sigma) \models (z^2_i, (--)\cdot P \cdot \mathcal{C}_i, z^3_i)$. This is because $\sigma(z^2_i) = m^{i,1}_\bot$ has a self-loop labeled $(--)$, there is an edge labeled P from $\sigma(z^2_i)$ into $\sigma(z^3_i) = p^i_\top$, and $p^i_\top$ has a self-loop labeled $\mathcal{C}_i$.

2. The case when $\kappa'(x_k) = \text{false}$ is similar and left to the reader.

In any case we conclude that $(G,\sigma) \models (z^2_i, \mathcal{R}, z^3_i)$.

Assume now, on the other hand, that $\text{CERTAIN}(Q_\varphi, \pi_\varphi) = \texttt{true}$. Take an arbitrary valuation κ from the propositional variables $\{x_1,\ldots,x_k\}$ into Boolean values true and false. We prove next that there there is an assignment κ' for the propositional variables $\{x_1,\ldots,x_m,y_1,\ldots,y_t\}$ of φ into Boolean values true and false, such that κ' coincides with κ on $\{x_1,\ldots,x_m\}$ and κ' assigns value true to exactly one propositional variable in each clause C_i, $1 \le i \le p$. This is sufficient to prove that φ belongs to $\forall\exists$ POSITIVE 1-3 3SAT since κ is arbitrarily chosen.

Consider the graph database G over Σ that is obtained from π_φ by replacing, for each $1 \le i \le p$, the label variable X_i with 1, if $\kappa(X_i) = \text{true}$, and with 0 otherwise. Clearly, $G \in [\![\pi_\varphi]\!]$ and hence Q_φ holds in G. Assume then that σ is an assignment for the node variables $\{y_1,\ldots,y_t,z^1_1,z^2_1,z^3_1,\ldots,z^1_p,z^2_p,z^3_p\}$ of Q_φ such that $(G,\sigma) \models Q_\varphi$. Let us define κ' to be the following assignment for the propositional variables $\{x_1,\ldots,x_m,y_1,\ldots,y_t\}$ of φ: $\kappa'(x_i) = \kappa(x_i)$, for each $1 \le i \le m$, and for each $1 \le j \le t$ it is the case $\kappa'(y_j) = \text{true}$ if $\sigma(y_j) = n_\top$, and $\kappa'(y_j) = \text{false}$ otherwise. We prove next that κ' makes true exactly one propositional variable in each clause C_i, for $1 \le i \le p$.

Assume first that C_i is of the form $(y_j \vee y_\ell \vee x_k)$, for $1 \le j < \ell \le t$ and $1 \le k \le m$. Suppose initially that $\kappa(x_k) = \text{true}$. Then every edge that leads from $\sigma(z^2_i)$ into $\sigma(z^3_i)$ in G is labeled with 1 (since it was labeled with X_k in π_φ). Since $(G,\sigma) \models Q_\varphi$, it is the case that $(G,\sigma) \models (z^2_i, \mathcal{R}, z^3_i)$, where $\mathcal{R} = (\bigcup_{a\in\Sigma\setminus\{0,1\}} a)\cdot\mathcal{C}_i \cup \big((--)\cdot 1 \cup (-+)\cdot 0 \cup (+-)\cdot 0\big)\cdot\mathcal{C}_i$. But the only possibility for this to happen in this case is that $(G,\sigma) \models (z^2_i, (--)\cdot 1\cdot\mathcal{C}_i, z^3_i)$. Notice that this immediately implies that $\sigma(z^2_i) = m^{i,1}_\bot$, and hence that $\sigma(y_j) = \sigma(y_\ell) = n_\bot$. We conclude that $\kappa'(y_j) = \kappa'(y_\ell) = \text{false}$, and, therefore, κ' only makes true one propositional variable in C_i. The case when $\kappa(x_k) = \text{false}$ can be handled analogously.

The case when C_i is of the form $(y_j \vee y_\ell \vee x_k)$, for $1 \le j < \ell \le t$ and $1 \le k \le m$ is analogous and left to the reader.

This finishes the proof of the lemma. It is interesting to notice that although the proof just presented uses a non-fixed alphabet (in particular, dependent on the number of clauses in the propositional formula φ), one can quite easily come out with a refinement of this reduction that uses a fixed alphabet. We decided to show here the simpler reduction, with a non-fixed

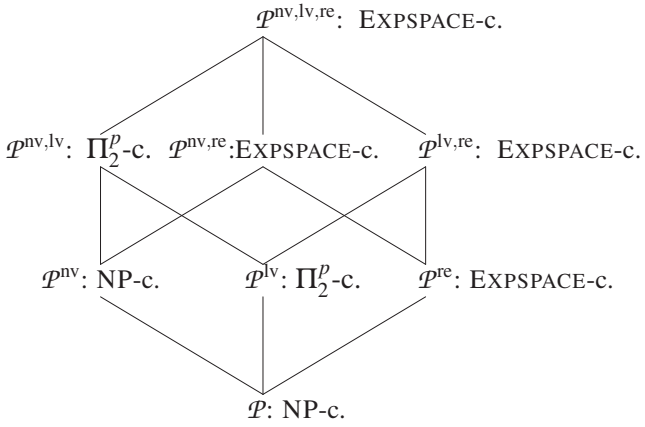


Figure 4.3: Combined complexity for CRPQs over graph patterns

alphabet, for the sake of presentation and readability. □

With the above lemma we finish the study of the combined complexity of the query answering problem. The following theorem summarizes our findings:

Theorem 4.4.2 *The combined complexity of answering CRPQs over classes of graph patterns is as shown in Figure 4.3 (The abbreviation '-c.' in the figure means, of course, complete for the class).*

Theorem 4.4.2 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, i.e., 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).

4.4.1.1 Restrictions to reduce the complexity

The next question is whether we can lower the EXPSPACE bound for patterns in $\mathcal{P}^{re}$. 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_2^p-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 4.4.3

- *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 $Q_{L'}() = (x, aL'a', y)$. 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'. □

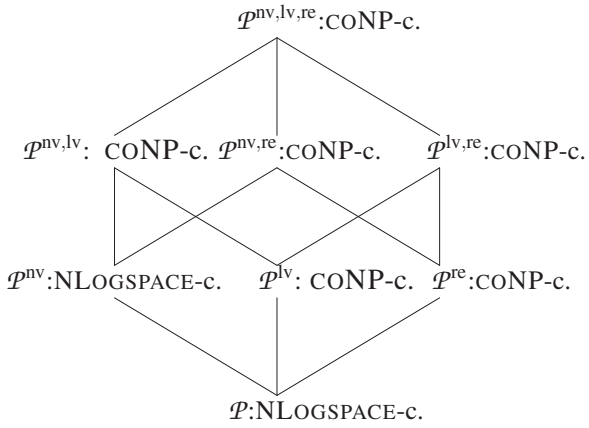


Figure 4.4: Data complexity for CRPQs over graph patterns

4.4.2 Data Complexity

We now turn to data complexity. As already mentioned, patterns in $\mathcal{P}$ are graphs and patterns in $\mathcal{P}^{nv}$ are naive tables, and thus naive evaluation works for them. This enabled us to show a PTIME upper bound for the case of arbitrary queries, and for CRPQs it gives us NLOGSPACE upper and lower bounds (the same bounds as the standard query evaluation problem for CRPQs [Cruz et al., 1987]). We have already proved a general coNP upper bound, showed that coNP-hardness exists for answering CRPQs over patterns in $\mathcal{P}^{nv,re}$, and promised to show the same bounds for $\mathcal{P}^{lv}$. Summing up, we have:

Theorem 4.4.4 *The data complexity of answering CRPQs over classes of graph patterns is as shown in Figure 4.4.*

Proof: All we need to show is coNP-hardness for patterns in $\mathcal{P}^{lv}$. We use again a reduction from 3-colorability, and one which is very similar to the reduction presented in the proof of Proposition 4.3.2. 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, i.e., (n_1,a,n_2) and (n_2,a,n_1). Now we turn it into a $\mathcal{P}^{lv}$ pattern π_G over alphabet $\Sigma = \{r,b,g\}$ by replacing the label of its edges, in the following fashion. For each node n, let X_n be a fresh label variable, and replace the label of all outgoing edges from n by X_n . It is now easy to see that the certain answers (with respect to Σ) to the Boolean RPQ $Q() = (x, rr|gg|bb, y)$ over π_G is `true` if and only if G is not 3-colorable. □

Just as we did for the case of combined complexity, a natural question at this point is how can the coNP lower bounds be lowered for patterns with regular expressions or label variables, perhaps by imposing additional restrictions on queries, or patterns, or perhaps both. This turns out to be an interesting, non-trivial question. The next chapter is completely dedicated towards answering it.

Chapter 5

Tractable Query Answering

While many results of Chapter 4 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.

The main goal of this chapter is to identify meaningful tractable fragments for the data complexity of query answering, which is coNP-hard as soon as patterns start using regular expressions or label variables. Recall that this problem asks, for a fixed graph query $Q = (\xi, \bar{x})$, and given a pattern π and a tuple $\bar{v}$ of nodes of size $|\bar{x}|$, whether $\bar{v}$ belongs to the certain answers of Q over π.

The first thing that we need to do in order to identify these restrictions is to study up to what extent the coNP-hardness persists when considering different restrictions of inputs to the certain answers problem. We do this in Section 5.1; the conclusion is that, in general, the lower bounds are very resilient, but that some advance can be made for some classes of patterns with particularly nice *underlying graphs*. In Sections 5.2 and 5.3 we present two disjoint sets of structural restrictions that guarantee the tractability of the Pattern Certain Answers(Q) problem. The first one is an adaptation of the notion of *bounded treewidth* for graph patterns, and the second one is roughly based on the notion of graphs with *bounded out-degree*.

Perhaps a different approach to reduce the coNP data complexity of computing certain answers is to rely on heuristics to solve this problem, at least in the majority of cases. The specific features of the certain answers problem 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. As a second contribution of this chapter, 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 of patterns.

5.1 How Regular Expressions and Label Variables are Problematic for Query Answering

Looking at Figure 4.4 in Section 4.4.2, we see that the features that cause coNP-hardness are 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.

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, i.e. $G_\pi = (N, \{(v_1,v_2) \mid (v_1,L,v_2) \in E)$.

5.1.1 The role of label variables

Our first result shows that the coNP-hardness result for query answering over patterns with label variables is very robust. Recall that $\mathcal{P}^{\sigma}_{\text{Codd}}$ stands for class of Codd patterns in $\mathcal{P}^{\sigma}$, i.e., patterns that use each variable once.

Theorem 5.1.1

- *There is a Boolean RPQ Q such that* Pattern Certain Answers(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 $\varphi() = (x,w,y)$, where w is a word in $\{0,1\}^*$, such that* Pattern Certain Answers(Q) *is* coNP*-hard even over $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ patterns whose underlying graph is a DAG.*

Proof: We begin with **Part 1**. 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 Pattern Certain Answers(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 [Schaefer, 1978], to the complement of Pattern Certain Answers(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, $1 \leq i \leq 10$, are defined as follows (we assume that Σ^* corresponds to the expression $(in^*0^*1^*\#^*out^*)^*$):

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

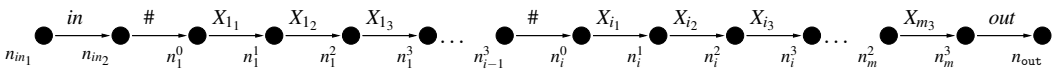
Clearly, L is a regular expression that uses concatenation and Kleene-star only.

The reduction is as follows. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a formula in 3CNF using variables $\{x_1, \ldots, x_k\}$, and assume that for each $1 \leq i \leq m$ 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 $\text{CERTAIN}(Q, \pi) =$ `false`.

($\Rightarrow$) Let $\gamma\colon \{x_1 \ldots, x_k\} \to \{0,1\}$ 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 $\text{CERTAIN}(Q, \pi) =$ `false`, we show the existence of a graph $G \in [\![\pi]\!]$ such that $Q(G) =$ `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 $Q(G) =$ `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 $(n^3_{i-1}, \#, n^0_i)$, for $1 < i \leq m$, and the edge $(n_{in_2}, \#, n^0_1)$.

Then, it must be the case that ρ' start in some node n^3_{i-1} ($1 < i \leq m$), or in node n_{in_2}, and therefore (by the construction of G), ρ' uses edges $(n^3_{i-1}, \#, n^0_i)$, $(n^0_i, \nu(X_{i_1}), n^1_i)$, $(n^1_i, \nu(X_{i_2}), n^2_i)$ and $(n^2_i, \nu(X_{i_3}), n^3_i)$ (or starting with $(n_{in_2}, \#, n^0_1)$ 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

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

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^3_m, 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.

We now continue with **Part 2**. We prove that there exists a Boolean RPQ Q of the form $Ans() \leftarrow (x,w,y)$, where w is a single word, such that PATTERN CERTAIN ANSWERS(Q) is coNP-hard even over input patterns in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ whose underlying graph is a DAG. We work with the word $w = 1011011101111$.

The proof is by a reduction from 3SAT to the complement of PATTERN CERTAIN ANSWERS(Q). For the sake of readability, we first construct in polynomial time from each propositional formula φ in 3CNF a pattern $\pi \in \mathcal{P}^{\text{lv,re}}_{\text{Codd}}$ over alphabet $\{0,1\}$, and show that φ is satisfiable if and only CERTAIN$(Q,\pi) = \texttt{false}$. Afterwards, we show how to construct in polynomial time from π a pattern $\pi' \in \mathcal{P}^{\text{lv}}_{\text{Codd}}$ over alphabet $\{0,1\}$, such that CERTAIN$(Q,\pi) = \texttt{false}$ iff CERTAIN$(Q,\pi') = \texttt{false}$.

Let $\varphi = \bigwedge_{1\leq j\leq m}(c^1_j \vee c^2_j \vee c^3_j)$ be a propositional formula in 3CNF that uses propositional variables from the set $\{x_1,\ldots,x_k\}$; that is, each c^i_j, for $1 \leq j \leq m$ and $1 \leq i \leq 3$, is either a variable x_ℓ, $1 \leq \ell \leq k$, or its negation. We associate with each propositional variable x_ℓ, $1 \leq \ell \leq k$, a fresh label variable X_ℓ. Also, with each clause $(c^1_j \vee c^2_j \vee c^3_j)$, $1 \leq j \leq m$ we associate three fresh label variables C^1_j, C^2_j and C^3_j.

The pattern π over alphabet $\{0,1\}$ contains the following edges:

- For each $1 \leq j \leq m$, π contains the edges $(n^1_j,1,n^2_j)$, (n^2_j,C^1_j,n^3_j), $(n^3_j,11,n^4_j)$, (n^4_j,C^2_j,n^5_j), $(n^5_j,111,n^6_j)$, (n^6_j,C^3_j,n^7_j), $(n^7_j,1111,n^8_j)$, where $n^1_j,\ldots,n^1_j$ are fresh node ids.
- For each $1 \leq \ell \leq k$, π contains the edges (p^1_ℓ,X_1,p^2_ℓ) and $(p^2_\ell,1111,p^3_\ell)$, where $p^1_\ell,p^2_\ell,p^3_\ell$ are fresh node ids.
- If $c^i_j = x_\ell$, for $1 \leq j \leq m$, $1 \leq i \leq 3$ and $1 \leq \ell \leq k$, the pattern π contains the edges $(n^{(2i+1)}_j,1101110,p^1_\ell)$ and $(n^{(2i+1)}_j,0110111,p^1_\ell)$.
- If $c^i_j = \neg x_\ell$, for $1 \leq j \leq m$, $1 \leq i \leq 3$ and $1 \leq \ell \leq k$, the pattern π contains the edges $(n^{(2i+1)}_j,110111,p^1_\ell)$ and $(n^{(2i+1)}_j,01101110,p^1_\ell)$.

Clearly, π belongs to $\mathcal{P}^{\text{lv,re}}_{\text{Codd}}$ and can be constructed in polynomial time from φ. Furthermore, it is easy to see that the underlying graph of π is a DAG. Figure 5.1 shows how this pattern looks for the case when $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$. We have skipped the names of node ids since they are all different and can be easily inferred from the context.

We prove next that φ is satisfiable if and only if CERTAIN$(Q,\pi) = \texttt{false}$.

($\Rightarrow$) Assume first that φ is satisfiable via assignment $\gamma\colon \{x_1\ldots,x_k\} \to \{0,1\}$. In order to prove that CERTAIN$(Q,\pi) = \texttt{false}$, we show the existence of a graph database G over alphabet $\{0,1\}$ such that $G \not\models Q$ but $G \models \pi$. To define G, we use again the notion of *canonical* graph database for π that we developed in the proof of Proposition 4.3.4.

Let σ be a canonical assignment for π, and define the following mapping ν from the label variables of π to $\{0,1\}$:

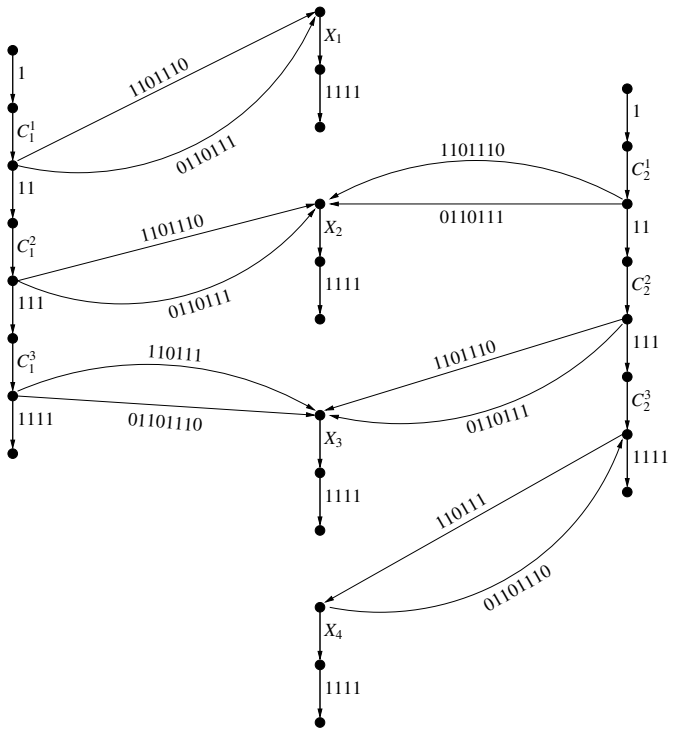


Figure 5.1: Pattern π for the case when $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$..

- For each $1 \leq \ell \leq k$, define $\nu(X_\ell) = 1$ if and only if $\gamma(x_\ell) = 1$.

- For each $1 \leq j \leq m$ and $1 \leq i \leq 3$ define $\nu(C_j^i) = 1$ if and only if $\gamma(x_\ell) = 1$, if the i-th literal of the j-th clause of φ corresponds to x_ℓ for some $1 \leq \ell \leq k$, or $\nu(C_j^i) = 1$ if and only if $\gamma(x_\ell) = 0$, if the i-th literal of the j-th clause of φ corresponds to $\neg x_\ell$ for some $1 \leq \ell \leq k$.

We then define G as the σ-canonical graph for π that is witnessed by ν (it is trivial to show that such graph always exists). Given that G is canonical for π, we obtain that $G \models \pi$.

Next we prove that $G \not\models Q$. We first claim that there is no path in G labeled w (recall that $w = 1011011101111$) that contains some vertex p_ℓ^1, for $1 \leq \ell \leq k$. Assume for the sake of contradiction that there exists a path ρ such that $\lambda(\rho) = 1011011101111$ and ρ contains the node id p_ℓ^1, for some $1 \leq \ell \leq k$. Notice that $|\rho| = 14$, since $|w| = 13$. It is not hard to see that G is a DAG, and, thus, the fourteen node ids in ρ need to be distinct. We consider several cases:

- The path does not use node ids in the set $\{n_j^1, \ldots, n_j^8 \mid 1 \leq j \leq m\}$. But from the properties of G, we know that this case is simply not possible. Indeed, the path in G from any node in the set $\{n_j^1, \ldots, n_j^8 \mid 1 \leq j \leq m\}$ to node p_ℓ^1 contains at most seven nodes from N (since each of these paths can only be labeled with a word of at most seven letters). Further, any path starting in p_ℓ^1 can have at most six nodes (going to p_ℓ^2 and then to p_ℓ^3). Thus, a

path in G that uses no node in the set $\{n_j^1,\ldots,n_j^8 \mid 1 \leq j \leq m\}$ but goes through p_ℓ^1 uses at most thirteen distinct node ids.

- The path does not contains a vertex $n_j^{(2i+1)}$, for some $1 \leq j \leq m$ and $1 \leq i \leq 3$, such that c_j^i is either x_ℓ or $\neg x_\ell$. This case is also not possible, given that the only edges that lead into node p_ℓ^1 are from nodes of the form $n_j^{(2i+1)}$, for some $1 \leq j \leq m$ and $1 \leq i \leq 3$, such that c_j^i is either x_ℓ or $\neg x_\ell$.

- The path contains a vertex $n_j^{(2i+1)}$, for some $1 \leq j \leq m$ and $1 \leq i \leq 3$, such that $c_j^i = x_\ell$. Then, from the definition of ν, we can check that $\nu(C_j^i) = \nu(X_\ell)$. From the properties of G, there are only two paths from $n_j^{(2i+1)}$ to p_ℓ^1: The first one is labeled with 1101110, and the second one is labeled with 0110111. Thus, either $\nu(C_j^i)\cdot 1101110\cdot\nu(X_\ell)$ is a subword of $w = 1011011101111$, or $\nu(C_j^i)\cdot 0110111\cdot\nu(X_\ell)$ is a subword of $w = 1011011101111$. But it is not hard to see that this cannot be the case since $\nu(C_j^i) = \nu(X_\ell)$.

- The path contains a vertex $n_j^{(2i+1)}$, for some $1 \leq j \leq m$ and $1 \leq i \leq 3$, such that $c_j^i = \neg x_\ell$. By using the same arguments as the previous case, one can show that in this case it must hold that $\nu(C_j^i)\cdot 110111\cdot\nu(X_\ell)$ is a subword of $w = 1011011101111$ or $\nu(C_j^i)\cdot 01101110\cdot$ $\nu(X_\ell)$ is a subword of $w = 1011011101111$. It is not hard to see that this cannot be the case, since ν is defined in such a way that $\nu(C_j^i) = 1$ if and only if $\nu(X_\ell) = 0$

We have proved that ρ does not contain any vertex of the form p_ℓ^1, for $1 \leq \ell \leq k$. We prove next that if we assume, for the sake of contradiction, that there is a path ρ in G such that $\lambda(\rho) = 1011011101111$, then it must be the case that ρ contains all nodes $n_j^1,\ldots,n_j^8$, for some $1 \leq j \leq m$. First, by a simple counting argument, it is easy to see that ρ cannot end in a node of the form n_j^s, for $1 \leq s \leq 7$ (because ρ uses at least fourteen distinct node ids). Thus, since ρ cannot contain any node of the form p_ℓ^1, $1 \leq \ell \leq k$, the only remaining option is that ρ ends in a node from G that is in a path that connects a node of the form $n_j^{(2i+1)}$, $1 \leq j \leq m$ and $1 \leq i \leq 3$, to a node of the form p_ℓ^1, for $1 \leq \ell \leq k$. But since w ends with 4 consecutive 1's, the only possibility is that ρ ends in the first two nodes of a path ρ' that connects a node of the form $n_j^{(2i+1)}$, $1 \leq j \leq m$ and $1 \leq i \leq 3$, to a node of the form p_ℓ^1, and such that ρ' is labeled either by 1101110 or 110111. But then this means that there is a path labeled 01 that leads in G to n_j^{2i}, which clearly cannot happen.

Thus, ρ is a path that goes from node id n_j^1 to node id n_j^8, for some $1 \leq j \leq m$. From the construction of G, this implies that $\nu(C_j^1) = \nu(C_j^2) = \nu(C_j^3) = 0$, which in turn implies that φ is not satisfied by valuation γ. This is the desired contradiction.

($\Leftarrow$) Assume, on the other hand, that $\textsc{certain}(Q,\pi) = \texttt{false}$. Then there is a graph G such that $G \models \pi$ but $G \not\models Q$. Since $G \models \pi$, there is a homomorphism $h = (h_1,h_2)$ from π to G, such that h_1 is the identity map on the node ids of π, and h_2 is a mapping from the

label variables used in π to the alphabet $\{0,1\}$. From h_2 we construct a valuation γ for the propositional variables of φ as follows: $\gamma(x_i) = 1$ if and only if $h_2(X_i) = 1$, for each $1 \leq i \leq k$. We show next that φ is satisfied by assignment γ. In order to prove this we use the following claim (for simplicity, we say that $\gamma(c_j^i)$ is the valuation of the i-th literal of the j-th clause of φ):

Claim 5.1.2 *For each $1 \leq j \leq m$ and $1 \leq i \leq 3$ it is the case that $\gamma(c_j^i) = 1$ if and only if $h_2(C_j^i) = 1$.*

Before proving the claim, we show how it follows from it that valuation γ satisfies the propositional formula φ. Indeed, assume on the contrary that γ does not satisfy φ, that is, for some $1 \leq j \leq m$ it is the case that $\gamma(c_j^1) = \gamma(c_j^2) = \gamma(c_j^3) = 0$. Then, from Claim 5.1.2 it follows that $h_2(C_j^1) = h_2(C_j^2) = h_2(C_j^3) = 0$, and, thus, it is easy to see that there is a path ρ from n_j^1 to n_j^8 in G that satisfies that $\lambda(\rho) = w = 1011011101111$. But this contradicts the fact that $G \not\models Q$.
Proof of Claim 5.1.2: Assume, for the sake of contradiction, that for some $1 \leq j \leq m$ and $1 \leq i \leq 3$ it is the case that $\gamma(c_j^i) \neq h_2(C_j^i)$. We consider four cases depending on h_2 and the choices of i and j:

- It holds that $h_2(C_j^i) = 0$ and $c_j^i = x_\ell$ for some $1 \leq \ell \leq k$. Then $\gamma(c_j^i) = 1$, $\gamma(x_\ell) = 1$ and, from the definition of γ, $h_2(X_\ell) = 1$. Further, since $G \models \pi$, we know that the following hold: (1) There is a path in G that goes from $n_j^{(2i-1)}$ to n_j^{2i} that is labeled with a word with i consecutive 1's. (2) There is a path in G that goes from n_j^{2i} to $n_j^{(2i+1)}$ that is labeled with $h_2(C_j^i) = 0$. (3) There is a path in G from $n_j^{(2i+1)}$ to p_ℓ^1 that is labeled with 1101110. (4) There is a path in G from p_ℓ^1 to p_ℓ^2 that is labeled $h_2(X_\ell) = 1$. (5) There is a path in G from p_ℓ^2 to p_ℓ^3 that is labeled with 1111. Combining all these paths, it is easy to see that there is a path in G that is labeled with $w = 1011011101111$. This is a contradiction, as we have assumed that $G \not\models Q$.

- It holds that $h_2(C_j^i) = 1$, and $c_j^i = x_\ell$ for some $1 \leq \ell \leq k$. This case is analogous to the previous one, but this time we use the fact that there is a path in G from $n_j^{(2i+1)}$ to p_ℓ^1 that is labeled with 0110111.

- We have that $h_2(C_j^i) = 0$ and $c_j^i = \neg x_\ell$ for some $1 \leq \ell \leq k$. Then $\gamma(c_j^i) = 1$, $\gamma(x_\ell) = 0$, and from the definition of γ, $h_2(X_\ell) = 0$. Further, since $G \models \pi$, we know that the following hold: (1) There is a path in G that goes from $n_j^{(2i-1)}$ to n_j^{2i} that is labeled with a word with i consecutive 1's. (2) There is a path in G that goes from n_j^{2i} to $n_j^{(2i+1)}$ that is labeled with $h_2(C_j^i) = 0$. (3) There is a path in G from $n_j^{(2i+1)}$ to p_ℓ^1 that is labeled with 110111. (4) There is a path in G from p_ℓ^1 to p_ℓ^2 that is labeled $h_2(X_\ell) = 0$. (5) There is a path in G from p_ℓ^2 to p_ℓ^3 that is labeled with 1111. Combining all these paths, it is easy to see that

there is a path in G that is labeled with $w = 1011011101111$. This is a contradiction, as we have assumed that $G \not\models Q$.

- It holds that $h_2(C^i_j) = 1$, and $c^i_j = \neg x_\ell$ for some $1 \leq \ell \leq k$. This case is analogous to the previous one, but this time we use the fact that there is a path in G from $n^{(2i+1)}_j$ to p^1_ℓ that is labeled with 01101110. □

This finishes the proof that φ is satisfiable if and only if $\text{CERTAIN}(Q,\pi) = \texttt{false}$.

To complete the proof, we need to show that there is a polynomial time procedure that, from π, constructs a pattern π' in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ such that $\text{CERTAIN}(Q,\pi) = \texttt{false}$ iff $\text{CERTAIN}(Q,\pi') = \texttt{false}$. But this is quite straightforward, since each regular language that is used to label an edge in π consists of a single word, and, thus, we can obtain an equivalent pattern π' by replacing each edge labeled with the word w with a path ρ of fresh node ids such that $\lambda(\rho) = w$. This finishes the proof of the theorem. □

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 the remainder of this chapter that there are indeed tractable classes that can be obtained along these lines.

5.1.2 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 5.1.3

- *There exists a Boolean RPQ Q of the form $\varphi() = (x,w,y)$, where w is a single word over $\Sigma = \{0,1\}$, such that* PATTERN CERTAIN ANSWERS(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* PATTERN CERTAIN ANSWERS(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 5.1.1. 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 $(0|1)$ (that is, $[\![\pi]\!] = [\![\pi']\!]$). 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, i.e., (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 the following sections.

5.2 Tractability Restrictions Based on Bounded Treewidth

Results of Section 5.1 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.

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 5.2.1 *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 follows from the proof of Proposition 3.3.4 because each pattern $\pi \in \mathcal{P}^{\text{nv,lv}}$ that does not repeat label variables is equivalent to a pattern $\pi' \in \mathcal{P}^{\text{nv,re}}$ with the same underlying graph as π.

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 5.2.2 *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 below) 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 $\mathcal{F} \times 2^S \times 2^{2^S}$ (that is, $t = |\mathcal{F}| \times 2^p \times 2^{2^p}$), 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 5.2.2, 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. (That is, 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). Next claim proves that the treewidth of $\mathcal{B}_{\pi,Q}$ is fixed.

Claim 5.2.3 *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:

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

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 below.

In the Y_j's we try to guess an assignment (that is, a graph database) that replaces each element of the form c_e in $\mathcal{B}_{\pi,Q}$ (that is, 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 (that is, 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 $\exists z_1 \exists z_3 Edges(z_1, z_2, z_3)$).

- 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 $1 \leq j \leq t$. 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, $1 \leq i \leq 4$, are as explained below.

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 $1 \leq i \leq p$ such that $s_i \in S'$. (2) For each nodes z, z' and w, if (a) z belongs to the interpretation X_j, $1 \leq j \leq p$, (b) $Edges(z, w, z')$ holds, (c) w belongs to Y_i, $1 \leq i \leq t$, then z' belongs to the interpretation of X_ℓ, for each $1 \leq \ell \leq p$ 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 above), 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 $1 \leq i \leq p$ such that $s_i \in S'$. Then recursively proceed as follows. If node p of $\mathcal{B}_{\pi,Q}$ is assigned to X_i (that is, 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 (that is, 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 $s_\ell \in f^j(s_i)$, i.e. 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:

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

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

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 (that is, that $\text{CERTAIN}(Q,\pi) = \texttt{false}$ if and only if $\mathcal{B}_{\pi,Q} \models \varphi_Q$).

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

Soundness and correctness: Assume first that $\mathcal{B}_{\pi,Q} \models \varphi_Q$. 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 $\mathbf{V}$ 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) $Edges(z,w,z')$ 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 $\mathcal{B}_{\pi,Q} \models Edges(p_0, c_{e_0}, q_0) \wedge Edges(p_{m+1}, c_{e_{m+1}}, q_{m+1})$. 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 4.3.3, 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 $\mathcal{B}_{\pi,Q} \models \varphi_Q$.

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 esentially 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 these types 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 proof actually shows the result for a larger class of patterns, with all the features, as long as label variables do not have multiple occurrences, and regular expressions do not mention them. The Codd interpretation of label variables is essential, since without it the problem is already coNP-hard for treewidth 1 (see Theorem 5.1.1). For $\mathcal{P}^{\text{re}}$ patterns, coNP-hardness results of Theorem 5.1.3 used classes of DAGs of unbounded treewidth.

Fixed parameter tractability. The formula φ_Q constructed in the proof only depends on Q, and the structure $\mathcal{B}_{\pi,Q}$ can be constructed in polynomial time from both π and Q. Thus, it immediately follows from Courcelle's theorem that the data complexity of finding certain answers to CRPQs over classes of patterns in $\mathcal{P}^{\text{nv,re}}$ and $\mathcal{P}^{\text{nv,lv}}_{\text{Codd}}$ is *fixed parameter tractable*, when the treewidth of the input pattern is used as the parameter of the problem. We refer the reader to [Downey and Fellows, 1999, Flum and Grohe, 2006] for a detailed discussion and definitions of fixed parameter tractability.

5.3 Tractability Restrictions for Patterns of Unbounded Treewidth

The proofs of Theorems 5.1.1 and 5.1.3 show that the data complexity of query answering rapidly becomes intractable if we allow patterns with underlying graphs of unbounded treewidth, even if these graphs are acyclic and the patterns does not use node variables.

However, by staring at these proofs, one observes that this intractability arises from some rather "unnatural" classes of patterns, in the sense that the structure of these patterns is extremely intricate, and the interaction of these patterns with the query given in these reductions is much more complex that what one would expect for many practical applications of query answering (we shall shortly explain why this is the case).

This gives hope to the idea of finding "natural" classes of patterns with tractable query evaluation. In this section we propose a set of structural restrictions on graph patterns, and on the interaction between patterns and queries, that gives rise to a class with good properties for computing certain answers.

We concentrate from now on a particular class of queries: CRPQs in which each regular expression occurring in the query does not use the Kleene-star (that is, the regular language defined by this expression is finite). We say that a CRPQ satisfying this restriction is *tame*. The class of tame CRPQs is relevant as it contains, among others, all conjunctive queries over the standard relational representation of graph databases. Moreover, notice that the intractability results in Theorem 5.1.1 and 5.1.3 for $\mathcal{P}^{\text{lv}}_{\text{Codd}}$ and $\mathcal{P}^{\text{re}}$ were actually proved for a tame RPQ (since the query in such theorem is a single word), and thus tame CRPQs are intractable even

over these classes of patterns.

As for the patterns, we know that finding tractable fragments for $\mathcal{P}^{\text{lv}}$ is almost hopeless. On the other hand, since $\mathcal{P}^{\text{re}}$ strictly contains the class of patterns in $\mathcal{P}^{\text{lv}}_{\text{Codd}}$, the class $\mathcal{P}^{\text{re}}$ seems to be a good alternative for our study. Furthermore, in order to maintain the presentation simple, we do not use node variables in our patterns, since they overly complicate the presentation (besides, Theorems 5.1.1 and 5.1.3 tells us that the problem is hard already for patterns without node variables).

We now define a restriction of the class $\mathcal{P}^{\text{re}}$ of patterns. This restriction is defined by two conditions that, when used together, yield tractability to the problem of computing certain answers of tame CRPQs over patterns in $\mathcal{P}^{\text{re}}$. As we later show, the obtained class of patterns is, in a precise sense, maximal for tractability.

5.3.1 Out-degree of patterns

Our first condition is a very simple restriction on the structure of patterns.

(C1) We require the *out-degree* of nodes in patterns to be bounded by a constant.

The out-degree of a node u in pattern π is defined as the number of edges of the form (u,R,u') in π. The out-degree of a pattern π, denoted by $\texttt{out}(\pi)$, is the maximum out-degree of a node in π. Let us denote by $\mathcal{OD}^{\text{re}}_{\leq d}$ the class of patterns π in $\mathcal{P}^{\text{re}}$ such that $\texttt{out}(\pi)$ is at most d.

Considering bounded out-degree is a common assumption in the study of complex networks [Manku et al., 2004]. This assumption reflects the idea that while incoming edges are generated distributively by all agents in a network, the outgoing edges are generated locally by a single node, and therefore its number can be assumed to be small as compared to the size of the whole network. For instance, in the Web it represents the fact that although a Web page can be linked by many other pages, each Web page has links to a very small number of Web pages (as compared to the size of the entire Web graph) [Donato et al., 2007].

By using condition (C1) we obtain a first simple tractability result for a limited class of queries.

Proposition 5.3.1 *Let $d \geq 0$ be a fixed value and Q a tame CRPQ without existentially quantified variables. Then* PATTERN CERTAIN ANSWERS(Q) *is in* PTIME*, when restricted to patterns in $\mathcal{OD}^{\text{re}}_{\leq d}$.*

We do not show the proof of this proposition, as it follows directly from Theorem 5.3.4 below.

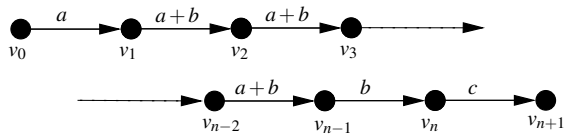
In the same way as a CRPQ can be represented as a graph query $Q = (\xi, \bar{x})$ with ξ a pattern in $\mathcal{P}^{\text{nv,re}}$, CRPQs without existentially quantified variables corresponds to queries of form $(\xi, \bar{x})$

with $\xi \in \mathcal{P}^{\text{re}}$. Thus, Proposition 5.3.1 can be restated in terms of graph queries in $\mathcal{P}^{\text{re}}$ that only use regular expressions that define finite languages.

5.3.2 Bounded Meaningful Functions

As we later show, tractability for tame CRPQs in general cannot be obtained by just bounding the out-degree of patterns (see Theorem 5.3.5 (2)). Thus, we provide another restriction that together with (C1) defines a tractable case for tame CRPQs. Let us illustrate the intuition of this condition by means of an example.

Example 5.3.2 *Let n be odd, and define* π_n *as:*



and consider tame CRPQs $Q = \exists x \exists y\ (x, aa|bb, y)$ *and* $Q' = \exists x \exists y\ (x, (a|b)bc, y)$.

Notice that the certain answers of Q and Q' *over* π_n *are* `true`. *In order to show this for the case of Q, one needs to check over all possible combinations resulting of assigning label either a or b to each of the edges of the form* $(v_i, a|b, v_{i+1})$. *On the other hand, in the case of* Q' *it is sufficient to inspect the labels of the last 3 edges of* π_n, *and thus, we essentially have to check only two combinations.*

The above example suggests that existential variables are problematic for query answering only when they can be witnessed by several nodes in the pattern, whereas queries in which the possible witnesses are limited should behave better. We formalize this intuition with our second condition, but first we need to introduce some terminology.

Let $\pi = (N, E)$ be a pattern over Σ and Q a tame CRPQ over the same schema. Assume that $\bar{x}$ is the tuple of free variables in Q and $\bar{y}$ the tuple of variables that are existentially quantified in Q, and consider a function $f : \bar{y} \to N \cup E$, that is, f maps each existentially quantified variable of Q either to a node (N) or to an edge (E) in π. Given a tuple $\bar{t}$, we say that f is *meaningful* for π, Q and $\bar{t}$, if there is a canonical graph database $G \in \text{Rep}_\Sigma(\pi)$ and a mapping σ, such that the following holds:

1. σ is a mapping from the variables of Q into the nodes of G, that witnesses $\bar{t} \in Q(G)$, in particular, $\sigma(\bar{x}) = \bar{t}$,
2. for each y in $\bar{y}$ we have $\sigma(y) = f(y)$ if $f(y) \in N$, or, if $f(y)$ is and edge $e \in E$, then $\sigma(y)$ is a node in the simple path $\rho_{f(y)}$ associated with to the edge e.

We denote by $\texttt{mf}(Q, \pi, \bar{t})$ the number of meaningful functions for π, Q and $\bar{t}$. We need a final definition before stating our condition. Given a pattern π, the *complete graph induced by* π,

denoted by $\mathtt{comp}(\pi)$, is the graph database obtained from π by removing each *incomplete edge* of π, that is, each edge labeled with a regular expression that is syntactically distinct from a symbol in Σ. We are now ready to state our second structural condition, which depends on π, Q, and $\bar{t}$.

(C2) If $\bar{t}$ does not belong to the evaluation of Q over $\mathtt{comp}(\pi)$, then the number of meaningful functions for π, Q and $\bar{t}$, is logarithmically bounded by the size of π.

From a practical point of view, we can understand this condition as follows. Assume that a query Q is not implied by the complete data of π. Then one can expect that the interaction between π and Q is rather sophisticated, and hence that Q should only have a small bound on the number of potential witnesses in a completion of the pattern. This is precisely what condition (C2) expresses, assuming such bound to be logarithmic, since meaningful functions essentially encode the potential witnesses of Q in a completion of the pattern.

Formally, given a tame CRPQ Q, a tuple $\bar{t}$, and a value $k \geq 0$, we denote by $\mathcal{MF}^{Q,\bar{t}}_{\leq k}$ the class of patterns π for which either $\bar{t} \in Q(\mathtt{comp}(\pi))$ or $\mathtt{mf}(Q,\pi,\bar{t}) \leq k \cdot \log(|\pi|)$, where $|\pi|$ is the size of π measured as its number of edges. Notice that for every tame CRPQ Q over Σ without existentially quantified variables (in particular for every tame RPQ), $\mathcal{MF}^{Q,\bar{t}}_{\leq k}$ contains all graph patterns over Σ independent of k and $\bar{t}$, since the notion of meaningful functions trivializes in this case.

The following lemma shows that membership in $\mathcal{MF}^{Q,\bar{t}}_{\leq k}$ can be efficiently checked, as long as the pattern is in $\mathcal{OD}^{\mathrm{re}}_{\leq d}$.

Lemma 5.3.3 *Given fixed tame CRPQ Q and values $d,k \geq 0$, checking if a pattern belongs to $\mathcal{OD}^{\mathrm{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$, for a tuple $\bar{t}$, can be done in polynomial time w.r.t. $|\pi|$.*

Proof: Assume that the tame CRPQ Q over Σ is of the form $\exists \bar{y} \bigwedge_{i=1}^{m}(x_i, L_i, y_i)$. By slightly abusing notation, we denote by $\bar{y}$ also the set of variables used in $\bar{y}$. Let π be a pattern over Σ. Clearly, checking whether π belongs to $\mathcal{OD}^{\mathrm{re}}_{\leq d}$ can be done in polynomial time. We prove next that if the latter is the case, that is, $\pi \in \mathcal{OD}^{\mathrm{re}}_{\leq d}$, then checking whether π belongs to $\mathcal{MF}^{Q,\bar{t}}_{\leq k}$ can also be done in polynomial time, which proves the lemma.

Clearly, the number of different functions from $\bar{y}$ into $N \cup D$ is polynomial, since the length of $\bar{y}$ is fixed, and hence in order to determine whether $\pi \in \mathcal{MF}^{Q,\bar{t}}_{\leq k}$ it is sufficient to show that the following problem is polynomial: Given a function f from $\bar{y}$ into $N \cup D$, determine whether f is meaningful for π, Q and $\bar{t}$. Indeed, if the latter is the case, we simply count in polynomial time the number s of meaningful functions for π, Q and $\bar{t}$, and then check whether $s \leq k \cdot \log(|\pi|)$, which clearly can be done in polynomial time in the size of π.

In order to check whether a function $f : \bar{y} \to N \cup D$ is meaningful for π, Q and $\bar{t}$, we do the following: We construct in polynomial time a propositional formula φ, of constant size, such that φ is satisfiable if and only if f is meaningful for π, Q and $\bar{t}$. Since φ is of constant size, it

contains at most a constant number of propositional variables, and hence its satisfiability can be decided in constant time. This proves the theorem.

Let $\pi = (N, D)$ be a pattern in $OD^{\text{re}}_{\leq d}$. An *edge-path* of π is a sequence $e_1 e_2 \dots e_n$, $n \geq 1$, of edges in D such that the end node of e_i, for $1 \leq i \leq n-1$, coincides with the initial node of e_{i+1}. If e is of the form (p, R, q), we denote its initial node p by $i(e)$ and its end node q by $s(e)$. Let $f : \bar{y} \to N \cup D$ be the function we want to check whether it is meaningful for π, Q and $\bar{t}$. For the sake of simplicity, we assume f also to be defined over the free variables of Q, by correspondingly assigning to those free variables the tuple $\bar{t}$.

In order to define φ it will be convenient to define first some relevant notation.

- For each word w that belongs to at least one of the (finite) languages L_i, $1 \leq i \leq m$, and each $p \leq |w|$, we define a set:

$$\mathcal{P}_w = \{(w_1, \dots, w_p) \mid \{w_1, \dots, w_p\} \in \Sigma^+, \text{ and } w_1 \cdots w_p = w\}.$$

  Intuitively, this set consists of all possible ways in which w can be divided into small pieces.

- Assume that $v \geq 0$ is the length of the longest string accepted by at least one language L_i, for $1 \leq i \leq m$. Then for each $1 \leq i \leq m$ and $p \leq v$ we define a set:

$$\mathcal{C}_{p,i} = \{(e_1 e_2 \cdots e_p) \text{ an edge-path in } \pi \mid (f(x_i) = e_1 \vee f(x_i) = i(e_1)) \text{ and } (f(y_i) = e_p \vee f(y_i) = s(e_p))\}.$$

  Intuitively, this set consists of all possible edge-paths in π of length p that start with edge e_1 and finish with edge e_p, assuming that the function f maps variable x_i into either e_1 or the initial node of e_1, $i(e_1)$, and f maps y_i into either e_p or the end node of e_p, $s(e_p)$.

- We also define a set $\mathcal{P}_Q$ consisting of all tuples $(\bar{w}_1, \dots, \bar{w}_m) \in \bigcup_{\{(w_1,\dots,w_m) \in L_1 \times \cdots \times L_m\}} \mathcal{P}_{w_1} \times \cdots \times \mathcal{P}_{w_m}$ that satisfy the following: (1) If $x_i = x_j$, for $1 \leq i < j \leq m$, and x_i and x_j belong to $\bar{y}$, then the first component of $\bar{w}_i$ coincides with the first component of $\bar{w}_j$. (2) If $y_i = y_j$, for $1 \leq i < j \leq m$, and y_i and y_j belong to $\bar{y}$, then the last component of $\bar{w}_i$ coincides with the last component of $\bar{w}_j$. (3) If $x_i = y_j$, for $1 \leq i \leq j \leq m$, and x_i and y_j belong to $\bar{y}$, then the first component of $\bar{w}_i$ coincides with the inverse of the last component of $\bar{w}_j$. This set represents the possible valuations for the L_i's in a graph database $G \in \text{Rep}(\pi)$ that are consistent with the information contained in the joins of Q.

The idea behind the propositional formula φ is trying to construct directly over π a graph database $G \in \text{Rep}(\pi)$ that witnesses the fact that f is meaningful for π, Q and $\bar{t}$. In order to do

that, π constructs G by "replacing" each edge $e = (p,R,q)$ of π with a simple path G_e [1] from p to q of fresh nodes, that is labelled with a word in R. Then it checks that $\bar{t} \in Q(G)$ via a mapping σ from the variables of Q into the nodes of G that satisfies the following: (i) $\sigma(\bar{x}) = \bar{t}$, and (ii) for each $y \in \bar{y}$ it is the case that $\sigma(y) = f(y)$, if $f(y) \in N$, and $\sigma(y)$ is a node in G_e, if $f(y)$ is the edge $e \in D$. That is, σ coincides with f on each existentially quantified variable y that f sends to the nodes of π, and if f sends y to an edge e of π then σ must map y into some node id that belongs to the path that replaces e in G.

However, recall that we are doing this process directly over π, and hence we have to do the following for each edge-path ρ linking $f(y_i)$ from $f(x_i)$ in π: Identify the possible replacements for the nested regular expressions that appear in ρ that force w to be satisfied by some path corresponding to ρ in G. Then take the disjunction over all those possible replacements, all paths ρ and all words in L_i.

The main technical problem at this stage is that $f(x_i)$ and $f(y_i)$ may be node ids in π, in the case when f sends these variables into N, or may have appeared as node ids in G at the moment of constructing G from π by replacing an edge labeled with a nested regular expression R with a path that is defined by R (in the case when f sends at least one of these variables into an edge in π). This complicates the construction, since the nodes $f(x_i)$ and/or $f(y_i)$ in this case have to be "guessed" by φ.

Notice that since the outdegree of π is constant, we only have to inspect a constant number of edge-paths (since each language in the query is fixed and finite, hence, we only need to inspect paths of length at most the length of the longest string in one of these finite languages).

Propositional formula φ is the conjunction of formulas φ_1 and φ_2, as defined below. The propositional variables of φ are of two classes: First, there are propositional variables of the form: $(L_e = w)$, $(L_e = w\Sigma^*)$, $(L_e = \Sigma^* w)$ and $(L_e = \Sigma^* w \Sigma^*)$, for (1) $e = (p,R,q)$ an edge of π labeled with regular expression R, and (2) w a nonempty subword of a word accepted by at least one of the L_i's. Intuitively, $(L_e = w)$ is true if and only if $e = (p,R,q)$ is replaced by path G_e, when constructing the graph database G that witnesses the fact that f is meaningful for π, Q and $\bar{t}$, such that w is precisely the label of the path G_e. Correspondingly, $(L_e = \Sigma^* w)$ is true if and only if e is replaced by path G_e such that the path from p to some node q' in G_e is labelled with w; $(L_e = w\Sigma^*)$ is true if and only if e is replaced by path G_e such that for some node p' in G_e the path from p' to q is labelled w; and $(L_e = \Sigma^* w \Sigma^*)$ is true if and only if e is replaced by graph database G_e such that for nodes p' and q' in G_e the path from p' to q' is labelled with w.

Second, we have all propositional variables of the form (L_e, P, S, W_1, W_2), where $e = (p,R,q)$ is an edge of π, and there is a path G_e from p to q that is labelled with a word in R and satisfies the following: (1) P is the set of all words w of size at most v such that there is a node q' in G_e such that the path from p to q' is labelled with w.

[1] for readability we use G_e instead of the standard ρ_e.

(2) S is the set of all words w of size at most v such that there is a node p' in G_e such that the path from p' to q is labelled with w.

(3) W_1 is the set of all words w of size at most v such that there are nodes p' and q' in G_e such that the path from p' to q' is labelled with w, and (4) W_2 is the set of all words w of size at most v such that the path from p to q is labelled w. Intuitively, (L_e, P, S, W_1, W_2) is true if and only if e is replaced in G by a path G_e such that the set of "prefixes" of G_e of lenth $\leq v$ is precisely P, the set of "suffixes" of G_e of length $\leq v$ is precisely S, the set of "subwords" of G_e of length $\leq v$ is precisely W_2, and the set of "words" of G_e that link p to q is precisely W_2.

Formula φ_1 is defined as:

$$\varphi_1 = \bigvee_{\{(\bar{w}_1,\ldots,\bar{w}_m)\in\mathcal{P}_Q\}} \bigwedge_{1\leq i\leq m} \bigvee_{\substack{\{\bar{e}=(e_1,\ldots,e_p)\in\mathcal{C}_{p,i}\},\\ \text{assuming } \bar{w}_i=(w_1,\ldots,w_p)}} \varphi_{\bar{e},i,\bar{w}_i},$$

where $\varphi_{\bar{e},i,\bar{w}_i}$ is defined by cases as follows:

$$\varphi_{\bar{e},i,\bar{w}_i} = \begin{cases} (L_e = \Sigma^* w \Sigma^*) & \text{if } \bar{e}=(e) \text{ and } f(x_i)=e=f(y_i) \\ \bigwedge_{l=1}^{k-1}(L_{e_l}=w_l)\wedge(L_{e_k}=w_k\Sigma^*) & \text{if } f(x_i)=i(e_1) \text{ and } f(y_i)=e_k \\ (L_{e_1}=\Sigma^* w_1)\wedge\bigwedge_{l=2}^{k}(L_{e_l}=w_l) & \text{if } f(x_i)=e_1 \text{ and } f(y_i)=s(e_k) \\ (L_{e_1}=\Sigma^* w_1)\wedge\bigwedge_{l=2}^{k-1}(L_{e_l}=w_l)\wedge(L_{e_k}=w_k\Sigma^*) & \text{if } f(x_i)=e_1 \text{ and } f(y_i)=e_k \\ \bigwedge_{l=1}^{k}(L_{e_l}=w_l) & \text{if } f(x_i)=i(e_1) \text{ and } f(y_i)=s(e_k) \end{cases}$$

This formula, intuitively, states that for each atom (x_i, L_i, y_i) in Q it is the case that $f(x_i)$ is linked to $f(y_i)$ by a path in G labeled with word in L_i, and hence that $Q(\bar{t})$ holds in G via some assignment specified by f.

Formula φ_2 states that edges in π are replaced by graph databases that satisfy the corresponding regular expression, and that these replacements are consistent with the variables that φ_1 assigns the value true. That is, φ_2 states that for each e mentioned in φ_1 exactly one variable of the form (L_e, P, S, W_1, W_2) is true. It also states, among others, that if $(L_e = w)$ and (L_e, P, S, W_1, W_2) are true, then $w \in W_2$; that if $(L_e = \Sigma^* w)$ and (L_e, P, S, W_1, W_2) are true, then $w \in S$; that if (L_e, P, S, W_1, W_2) is true, then $(L_e = w\Sigma^*)$ is true, for each $w \in P$; etc.

It is clearly the case that φ_1 can be constructed in polynomial time from π and $\bar{t}$. This is because, since the outdegree of π is constant, the number of different edge-paths in $\mathcal{C}_{p,i}$, for $p \leq v$, is also constant. Furthermore, we can determine in polynomial time the variables of the form (L_e, P, S, W_1, W_2) that are feasible, i.e. if $e = (p, R, q)$ then there is a path G_e from p to q that satisfies the aforementioned conditions.

Further, it is not hard to see that the size of φ is constant. In fact, φ_1 only mentions a constant number of propositional variables, and there exists a constant number of propositional variables of the form (L_e, P, S, W). We conclude that φ_2 is also of constant size. Hence, φ contains at most a constant number of propositional variables, and its satisfiability can clearly be checked in constant time.

We finally prove that f is meaningful for π, Q and $\bar{t}$ if and only if φ is satisfiable. Assume first that φ is satisfiable via assignment γ. Construct a canonical graph database $G \in \mathrm{Rep}(\pi)$ as follows. First, for each edge $e = (p,R,q)$ in π such that L_e is not mentioned in φ, replace edge e with an arbitrary path G_e (of fresh node ids) labelled with a word in R.

Second, for each edge $e = (p,R,q)$ in π such that L_e is mentioned in φ, do the following assuming that $\gamma(L_e,P,S,W) = 1$: Replace e with a graph path G_e that satisfies the conditions stated before, namely, that the label of G_e belongs to the language of R (1) P is the set of all words w of size at most v such that there is a node q' in G_e such that the path from p to q' is labelled with w.

(2) S is the set of all words w of size at most v such that there is a node p' in G_e such that the path from p' to q is labelled with w.

(3) W_1 is the set of all words w of size at most v such that there are nodes p' and q' in G_e such that the path from p' to q' is labelled with w, and (4) W_2 is the set of all words w of size at most v such that the path from p to q is labelled w.

Such graph database G exists because we ensured it with the construction of φ_2. We prove below that G witnesses the fact that f is meaningful for π, Q and $\bar{t}$.

Since γ satisfies φ_1, it is the case that γ satisfies

$$\bigvee_{\{(\bar{w}_1,\ldots,\bar{w}_m)\in\mathcal{P}_Q\}} \bigwedge_{1\leq i\leq m} \bigvee_{\substack{\{\bar{e}=(e_1,\ldots,e_p)\in\mathcal{C}_{p,i}\},\\ \text{assuming } \bar{w}_i=(w_1,\ldots,w_p)}} \varphi_{\bar{e},i,\bar{w}_i}.$$

Hence, for some $(\bar{w}_1,\ldots,\bar{w}_m) \in \mathcal{P}_Q$ it is the case that for every $1 \leq i \leq m$, if $\bar{w}_i = (w_1,\ldots,w_p)$ then there is an $\bar{e} = (e_1,\ldots,e_p) \in \mathcal{C}_{p,i}$ such that γ satisfies $\varphi_{\bar{e},i,\bar{w}_i}$. Then one should proceed by cases depending on where f takes values and the form of $\bar{e}$. All cases are rather analogous, so we only consider the case when $f(x_i) = e_1$, $f(y_i) = e_p$ and $p \geq 2$.

Consider an arbitrary i such that $1 \leq i \leq m$. Then there is an edge path $e_1 \cdots e_m$ in π such that γ satisfies $(L_{e_1} = \Sigma^* w_1)$, $(L_{e_m} = w_m\Sigma^*)$, and $(L_{e_j} = w_j)$, for each $2 \leq j \leq m-1$. Hence, since γ satisfies φ_2, it must be the case that $e_j = (p,R,q)$, for each $2 \leq j \leq m-1$, is replaced in G by path G_{e_j} labelled with w_j.

Similarly, $e_1 = (p,R,q)$ is replaced in G by a path G_{e_1} such that the path from p' to q is labelled w_1, for some node p' of G_{e_1}; and $e_m = (p,R,q)$ is replaced in G by a path G_{e_m} such that the path from p to q' is labelled w_m, for some node q' of G_{e_m}

Let us define an assignment σ for the variables of Q into G that sends x_i into node p', as defined above, and sends y_i to node q'. We can do this in a way that the assignment is consistent with the joins of Q (that is, with the repeated variables in different atoms of Q) since $(\bar{w}_1,\ldots,\bar{w}_p) \in \mathcal{P}_Q$. We conclude that $\bar{t} \in Q(G)$ via the mapping σ. Furthermore, clearly $\sigma(\bar{x}) = \bar{t}$, and for each $y \in \bar{y}$ it is the case that $\sigma(y) = f(y)$, if $f(y) \in N$, and $\sigma(y)$ is a node in G_e, if $f(y)$ is the edge $e \in D$.

Assume, on the other hand, that f is meaningful for π, Q and $\bar{t}$. Assume that G is a canonical graph database over Σ that witnesses this fact. That is, it is the case that $Q(G)$ via a mapping σ from the variables of Q into the nodes of G that satisfies the following: (i) $\sigma(\bar{x}) = \bar{t}$, and (ii) for each $y \in \bar{y}$ it is the case that $\sigma(y) = f(y)$, if $f(y) \in N$, and $\sigma(y)$ is a node in G_e, if $f(y)$ is the edge $e \in D$.

Let us define an assignment γ for φ as follows. For each edge e in π that is replaced by path G_e in G, do the following: First of all, for each e in π it is the case that γ satisfies the propositional variable (L_e, P, S, W_1, W_2), where P, S, W_1 and W_2 are defined as usual for G_e. Furthermore, γ satisfies each propositional variable mentioned in φ of the form $(L_e = w)$, $(L_e = \Sigma^* w')$, $(L_e = \Sigma^* w' \Sigma^*)$ and $(L_e = w' \in \Sigma^*)$ such that $w \in W_2$, $w \in S$, $w \in W_1$, and $w \in P$, respectively. It falsifies any other propositional variable in φ that mentions L_e. Clearly, γ satisfies φ_2. We prove below that it also satisfies φ_1.

Since $\bar{t} \in Q(G)$, there is a path $\rho(i)$ in G from $\sigma(x_i)$ into $\sigma(y_i)$ such that its label $\lambda(\rho(i))$ belongs to the language specified by L_i, for each $1 \leq i \leq m$. We can assume each one of these paths to be of the form $n_0 a_1 n_1 a_2 n_2 \cdots n_{p-1} a_p n_p$, where each n_i is a node id in G, each a_j is a symbol in Σ, $p \leq v$, $n_0 = f(x_i)$ and $n_p = f(y_i)$. But we can also assume without loss of generality that the path $\rho(i)$ is either of the form (a) ρ^i_1, where ρ^i_1 is the subword of a graph database of the form G_e such that G_e is the graph database associated with some edge e of π in G, or (b) $\rho^i_1 \rho^i_{e_1} \ldots \rho^i_{e_\ell} \rho^i_2$, $\ell \geq 0$, where each $\lambda(\rho^i_{e_j})$ is a word in the graph database G_{e_j} associated with some edge e_j of π in G, for $1 \leq j \leq \ell$, $\lambda(\rho^i_1)$ is a suffix of a graph database G_e that is associated with some edge e of π in G, and $\lambda(\rho^i_2)$ is the prefix of a graph database $G_{e'}$ that is associated with some edge e' of π in G. We only analyze the second more general case, the first one being analogous.

It is clear that $\lambda(\rho^i_1)\lambda(\rho^i_{e_1}) \cdots \lambda(\rho^i_{e_\ell})\lambda(\rho^i_2) = \lambda(\rho(i))$. Furthermore, we can assume without loss of generality that $\lambda(\rho^i_1)$, $\lambda(\rho^i_{e_2})$ and each $\lambda(\rho^i_{e_j})$, for $1 \leq j \leq \ell$, is a nonempty word. Thus, it is not hard to see that if $\bar{w}_i = (\lambda(\rho^i_1), \lambda(\rho^i_{e_1}), \ldots, \lambda(\rho^i_{e_\ell}), \lambda(\rho^i_2))$, then $(\bar{w}_1, \ldots, \bar{w}_m)$ belongs to $\mathcal{P}_Q$.

Consider an arbitrary $1 \leq i \leq m$. Then $\bar{e} = (e, e_1, \ldots, e_\ell, e')$ belongs to $\mathcal{C}_{\ell+2,i}$. Furthermore, γ makes true each propositional variable of the form $(L_{e_j} = \lambda(\rho^i_{e_j}))$, for $1 \leq j \leq \ell$, as well as propositional variable $(L_e = \Sigma^* \lambda(\rho^i_1))$ and $(L_{e'} = \lambda(\rho^i_2)\Sigma^*)$. In any possible case it must be the case that γ satisfies $\varphi_{\bar{e},i,\bar{w}_i}$. This is precisely what we wanted to prove, since it implies that for every $1 \leq i \leq m$ there is an $\bar{e} \in \mathcal{C}_{p,i}$, assuming $\bar{w}_1 = (w_1, \ldots, w_p)$, such that γ satisfies $\varphi_{\bar{e},i,\bar{w}_i}$. This finishes the proof of the Lemma. □

5.3.3 Query answering under restrictions (C1) and (C2)

The following result shows that checking whether $\bar{t} \in \text{CERTAIN}(Q, \pi)$ can be efficiently decided for patterns in $\mathcal{OD}^{\text{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$.

Theorem 5.3.4

Let $d,k \geq 0$ be fixed values and Q a tame CRPQ. Then PATTERN CERTAIN ANSWERS(Q) *is in* PTIME*, when restricted to patterns in $OD^{\text{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$.*

Proof: We start by checking whether $\bar{t} \in Q(\texttt{comp}(\pi))$, which clearly can be done in polynomial time. If this is the case, then we can conclude that $\bar{t} \in \text{CERTAIN}(Q,\pi)$. Otherwise, we go over the procedure described below.

We use essentially the same techniques of the proof of Lemma 5.3.3.

Assume again that the tame CRPQ Q over Σ is of the form $\exists \bar{y} \bigwedge_{i=1}^{m}(x_i, L_i, y_i)$. Once again, we may denote by $\bar{y}$ also the set of variables used in $\bar{y}$. We prove the theorem by constructing, in polynomial time, from each graph pattern $\pi \in OD^{\text{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$, without null values, and tuple $\bar{t}$ of node ids, a propositional formula φ of logarithmic length in the size of π such that φ is satisfiable if and only $\bar{t} \notin \Box Q(\pi)$. Since φ is of logarithmic size, it contains at most a logarithmic number of propositional variables, and hence its satisfiability can be decided in polynomial time.

Let $\pi = (N,D)$ be a pattern in $OD^{\text{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$ without null values. We denote by F the set of meaningful functions from $\bar{y}$ into $N \cup D$. By using the same techniques as in the proof of Lemma 5.3.3 one can show that the set F can be constructed in polynomial time from π and $\bar{t}$. Recall that since $\pi \in MF^{Q,\bar{t}}_{\leq k}$, the size of F is bounded by $k \cdot \log(|\pi|)$. For the sake of simplicity, once again we assume each $f \in F$ also to be defined over the free variables of Q, by correspondingly assigning to those free variables the tuple $\bar{t}$. It is also important to recall the sets $\mathcal{P}_w$, $\mathcal{C}_{p,i}$ and $\mathcal{P}_Q$ that we have defined in the proof of the previous Lemma.

The idea behind the propositional formula φ is trying to construct directly over π a graph database $G \in \text{Rep}(\pi)$ such that $\bar{t} \notin Q(G)$. In order to do that, π constructs a canonical database G, and checks that the tuple $\bar{t}$ of nodes in G does not satisfy the query Q. In order to do that, φ inspects each meaningful function $f \in F$, and show that for each one of them there is an atom (x_i, L_i, y_i) in Q that is not satisfied in the assignment given by f in G. In order to do that, it checks that no word w in L_i is witnessed by some path that links $f(y_i)$ from $f(x_i)$ in G. However, recall that we are doing this directly over π, and hence we have to do the following for each path ρ linking $f(y_i)$ from $f(x_i)$ in π: Identify the possible replacements for the nested regular expressions that appear in ρ that force w not to be satisfied by the path corresponding to ρ in G. Then take the conjunction over all those possible replacements, all paths ρ and all words in L_i.

The main technical problem at this stage is that $f(x_i)$ and $f(y_i)$ may be node ids in π, in the case when f sends these variables into N, or may have appeared as node ids in G at the moment of constructing G from π by replacing an edge labeled with a nested regular expression R with a graph database that satisfies R (in the case when f sends at least one of these variables into an edge in π). This complicates the construction a bit, since the nodes $f(x_i)$ and/or $f(y_i)$ in this

case have to be "guessed" by φ.

Since the outdegree of π is constant, we only have to inspect a constant number of paths for each meaningful function (since each language in the query is fixed and finite, hence, we only need to inspect paths of length at most the length of the longest string in one of these finite languages). Since there is a logarithmic number of meaningful functions, in the end we only have to inspect a logarithmic number of paths, which can be reduced to checking satisfiability of a logarithmic size formula. We formalize this idea below.

Propositional formula φ is the conjunction of formulas φ_1 and φ_2, as defined below. The propositional variables of φ are the same that we have used in the previous lemma: First, there are propositional variables of the form: $(L_e = w)$, $(L_e = w\Sigma^*)$, $(L_e = \Sigma^* w)$ and $(L_e = \Sigma^* w \Sigma^*)$, for (1) $e = (p,R,q)$ an edge of π labeled with regular expression R, and (2) w a nonempty subword of a word accepted by at least one of the L_i's.

And second, we have all propositional variables of the form (L_e, P, S, W_1, W_2), where $e = (p,R,q)$ is an edge of π, and there is a path G_e from p to q that is labelled with a word in R and satisfies the following: (1) P is the set of all words w of size at most v such that there is a node q' in G_e such that the path from p to q' is labelled with w.

(2) S is the set of all words w of size at most v such that there is a node p' in G_e such that the path from p' to q is labelled with w.

(3) W_1 is the set of all words w of size at most v such that there are nodes p' and q' in G_e such that the path from p' to q' is labelled with w, and (4) W_2 is the set of all words w of size at most v such that the path from p to q is labelled w.

Formula φ_1 is defined as:

$$\varphi_1 = \bigwedge_{f \in F} \bigwedge_{\{(\bar{w}_1,\ldots,\bar{w}_m) \in \mathcal{P}_Q\}} \bigvee_{1 \leq i \leq m} \bigwedge_{\substack{\{\bar{e}=(e_1,\ldots,e_p) \in \mathcal{C}_{p,i}\}, \\ \text{assuming } \bar{w}_i = (w_1,\ldots,w_p)}} \varphi_{\bar{e},f,i,\bar{w}_i},$$

where $\varphi_{\bar{e},f,i,\bar{w}_i}$ is defined by cases as follows:

$$\varphi_{\bar{e},f,i,\bar{w}_i} = \begin{cases} \neg(L_e = \Sigma^* w \Sigma^*), & \text{if } \bar{e} = (e) \text{ and } f(x_i) = e = f(y_i) \\ \bigvee_{l=1}^{k-1} \neg(L_{e_l} = w_l) \vee \neg(L_{e_k} = w_k \Sigma^*), & \\ & \text{if } f(x_i) = i(e_1) \text{ and } f(y_i) = e_k \\ \neg(L_{e_1} = \Sigma^* w_1) \vee \bigvee_{l=2}^{k} \neg(L_{e_l} = w_l), & \\ & \text{if } f(x_i) = e_1 \text{ and } f(y_i) = s(e_k) \\ \neg(L_{e_1} = \Sigma^* w_1) \vee \bigvee_{l=2}^{k-1} \neg(L_{e_l} = w_l) \vee \neg(L_{e_k} = w_k \Sigma^*), & \\ & \text{if } f(x_i) = e_1 \text{ and } f(y_i) = e_k \\ \bigvee_{l=1}^{k} \neg(L_{e_l} = w_l) & \text{if } f(x_i) = i(e_1) \text{ and } f(y_i) = s(e_k) \end{cases}$$

This formula, intuitively, states that for f in F there is an atom (x_i, L_i, y_i) in Q such that $f(x_i)$ is not linked to $f(y_i)$ by a path in G labeled with word in L_i, and hence that $\bar{t} \notin Q(G)$.

Formula φ_2 states that edges in π are replaced by graph databases that satisfy the corresponding regular expressions, and that these replacements are consistent with the variables that φ_1 assigns the value true. It is defined in a similar way that the proof of the previous Lemma.

Since F can be constructed in polynomial time from π and $\bar{t}$, and Q is fixed, it is clearly the case that φ_1 can be constructed in polynomial time from π and $\bar{t}$. This is because, since the outdegree of π is constant, the number of different edge-paths in $\mathcal{C}_{p,i}$, for $p \leq v$, is also constant. Furthermore, as we have discussed in the proof of the previous Lemma, we also have that φ_2 can be constructed in polynomial time.

Furthermore, it is not hard to see that the size of φ is logarithmically bounded. Indeed, each formula of the form:

$$\bigwedge_{\{(\bar{w}_1,\ldots,\bar{w}_m)\in\mathcal{P}_Q\}} \bigvee_{1\leq i\leq m} \bigwedge_{\substack{\{\bar{e}=(e_1,\ldots,e_p)\in\mathcal{C}_{p,i}\},\\ \text{assuming } \bar{w}_i=(w_1,\ldots,w_p)}} \varphi_{\bar{e},f,i,\bar{w}_i}$$

is of constant size, and hence, since F is of size logarithmic in π, the formula φ_1 is of size at most logarithmic in π. Furthermore, φ_2 is of constant size. Hence, φ contains at most a logarithmic number of propositional variables, and its satisfiability can clearly be checked in polynomial time.

We finally prove that $\bar{t} \notin \text{CERTAIN}(Q,\pi)$ if and only if φ is satisfiable. In order to do that, we use again the notion of canonical graph database introduced in Chapter 4.

Assume first that φ is satisfiable via assignment γ. Construct a canonical graph database $G \in \text{Rep}(\pi)$ as follows. First, for each edge $e = (p,R,q)$ in π such that L_e is not mentioned φ, replace edge $e \in \pi$ with an arbitrary path of fresh internal node ids, that is labelled with a word in R. Second, for each edge $e = (p,R,q)$ in π such that L_e is mentioned in φ, do the following assuming that $\gamma(L_e,P,S,W_1,W_2) = 1$: Replace e with a path G_e that satisfies the conditions stated before, namely, that the label of G_e belongs to the language of R (1) P is the set of all words w of size at most v such that there is a node q' in G_e such that the path from p to q' is labelled with w.

(2) S is the set of all words w of size at most v such that there is a node p' in G_e such that the path from p' to q is labelled with w.

(3) W_1 is the set of all words w of size at most v such that there are nodes p' and q' in G_e such that the path from p' to q' is labelled with w, and (4) W_2 is the set of all words w of size at most v such that the path from p to q is labelled w. We prove below that $\bar{t} \notin Q(G)$, which suffices for the proof.

Assume, for the sake of contradiction, that $\bar{t} \in Q(G)$, and that this is witnessed by a homomorphism σ (in particular, $\sigma(\bar{x}) = \bar{t}$, assuming the tuple of free variables of Q to be $\bar{x}$). Let f be the mapping from $\bar{y}$ into $N \cup D$ defined as follows: For each y mentioned in $\bar{y}$ we have that $f(y) = \sigma(y)$, if $\sigma(y)$ is a node in G that also belongs to π, and $f(y) = e$, if σ sends y to a fresh

internal node id in path G_e. It is clear then that f is a meaningful function for π, Q and $\bar{t}$, as witnessed by graph database G.

Since γ satisfies φ_1, it is the case that γ satisfies

$$\bigwedge_{\{(\bar{w}_1,\ldots,\bar{w}_m)\in\mathcal{P}_Q\}} \bigvee_{1\leq i\leq m} \bigwedge_{\substack{\{\bar{e}=(e_1,\ldots,e_p)\in\mathcal{C}_{f,i}\},\\ \text{if } \bar{w}_i=(w_1,\ldots,w_p)}} \varphi_{\bar{e},f,i,\bar{w}_i}.$$

Further, since $\bar{t}\in Q(G)$, there is a path $\rho(i)$ in G from $\sigma(x_i)$ into $\sigma(y_i)$ such that its label $\lambda(\rho(i))$ belongs to the language specified by L_i, for each $1\leq i\leq m$. We can assume each one of these paths to be of the form $n_0a_1n_1a_2n_2\cdots n_{p-1}a_pn_p$, where each n_i is a node id in G, each a_j is a symbol in Σ, $p\leq v$, $n_0=f(x_i)$ and $n_p=f(y_i)$. But we can also assume without loss of generality that the path $\rho(i)$ is either of the form (a) ρ^i_1, where ρ^i_1 is the subword of a graph database of the form G_e such that G_e is the path associated with some edge e of π in G, or (b) $\rho^i_1\rho^i_{e_1}\cdots\rho^i_{e_\ell}\rho^i_2$, $\ell\geq 0$, where each $\lambda(\rho^i_{e_j})$ is a word in the path G_{e_j} associated with some edge e_j of π in G, for $1\leq j\leq\ell$, $\lambda(\rho^i_1)$ is a suffix of a path G_e that is associated with some edge e of π in G, and $\lambda(\rho^i_2)$ is the prefix of a path $G_{e'}$ that is associated with some edge e' of π in G. We only analyze the second more general case, the first one being analogous.

It is clear that $\lambda(\rho^i_1)\lambda(\rho^i_{e_1})\cdots\lambda(\rho^i_{e_\ell})\lambda(\rho^i_2)=\lambda(\rho(i))$. Furthermore, we can assume without loss of generality that $\lambda(\rho^i_1)$, $\lambda(\rho^i_{e_2})$ and each $\lambda(\rho^i_{e_j})$, for $1\leq j\leq\ell$, is a nonempty word. Thus, it is not hard to see that if $\bar{w}_i=(\lambda(\rho^i_1),\lambda(\rho^i_{e_1}),\ldots,\lambda(\rho^i_{e_\ell}),\lambda(\rho^i_2))$, then $(\bar{w}_1,\ldots,\bar{w}_m)$ belongs to $\mathcal{P}_Q$.

Consider an arbitrary $1\leq i\leq m$. Then $\bar{e}=(e,e_1,\ldots,e_\ell,e')$ belongs to $\mathcal{C}_{\ell+2,i}$. Furthermore, γ makes true each propositional variable of the form $(L_{e_j}=\lambda(\rho^i_{e_j}))$, for $1\leq j\leq\ell$, as well as propositional variable $(L_e=\Sigma^*\lambda(\rho^i_1))$ and $(L_{e'}=\lambda(\rho^i_2)\Sigma^*)$. In any possible case it must be the case that γ either does not satisfy $\varphi_{\bar{e},f,i,\bar{w}}$ or it does not satisfy φ_2. This is our desired contradiction since it implies that either γ does not satisfy φ_2, or for every $1\leq i\leq m$ there is an $\bar{e}\in\mathcal{C}_{f,i}$ such that γ does not satisfy $\varphi_{\bar{e},f,i,\bar{w}_i}$.

Assume, on the other hand, that $\bar{t}\notin \textsc{certain}(Q,\pi)$. Then from Claim 4.3.3 there is a canonical graph database $G\in\text{Rep}(\pi)$ such that $\bar{t}\notin Q(G)$. Let us define an assignment γ for φ as follows. For each edge e in π that is replaced by path G_e in G, do the following: First of all, for each e in π it is the case that γ satisfies the propositional variable (L_e,P,S,W_1,W_2), where P, S, W_1 and W_2 are defined as usual for G_e. Furthermore, γ satisfies each propositional variable mentioned in φ of the form $(L_e=w)$, $(L_e=\Sigma^*w')$, $(L_e=\Sigma^*w'\Sigma^*)$ and $(L_e=w'\in\Sigma^*)$ such that $w\in W_2$, $w\in S$, $w\in W_1$, and $w\in P$, respectively. It falsifies any other propositional variable in φ that mentions L_e. Clearly, γ satisfies φ_2. We prove that it also satisfies φ_1.

Assume, for the sake of contradiction, that γ does not satisfy φ_1. This implies that there is

a function $f \in F$ such that for each $1 \leq i \leq m$ it is the case that γ satisfies:

$$\bigvee_{\{(\bar{w}_1,\ldots,\bar{w}_m)\in\mathcal{P}_Q\}} \bigwedge_{1\leq i\leq m} \bigvee_{\substack{\{\bar{e}=(e_1,\ldots,e_p)\in\mathcal{C}_{p,i}\},\\ \text{assuming } \bar{w}_i=(w_1,\ldots,w_p)}} \neg\varphi_{\bar{e},f,i,\bar{w}_i}$$

Hence, for some $(\bar{w}_1,\ldots,\bar{w}_m) \in \mathcal{P}_Q$ it is the case that for every $1 \leq i \leq m$, if $\bar{w}_i = (w_1,\ldots,w_p)$ then there is an $\bar{e} = (e_1,\ldots,e_p) \in \mathcal{C}_{p,i}$ such that γ satisfies $\neg\varphi_{\bar{e},f,i,\bar{w}}$. Then one should proceed by cases depending on where f takes values and the form of $\bar{e}$. All cases are rather analogous, so we only consider the case when $f(x_i) = e_1$, $f(y_i) = e_p$ and $p \geq 2$.

Consider an arbitrary i such that $1 \leq i \leq m$. Then there is an edge path $e_1 \cdots e_m$ in π such that γ satisfies $(L_{e_1} = \Sigma^* w_1)$, $(L_{e_m} = w_m\Sigma^*)$, and $(L_{e_j} = w_j)$, for each $2 \leq j \leq m-1$. Hence, since γ satisfies φ_2, it must be the case that $e_j = (p,R,q)$, for each $2 \leq j \leq m-1$, is replaced in G by a path G_{e_j} labelled with w_j. Similarly, $e_1 = (p,R,q)$ is replaced in G by a path G_{e_1} such that for a node p' of G_{e_1} the path from p' to q is labelled with w_1; and $e_m = (p,R,q)$ is replaced in G by a path G_{e_m} such that for some node q' in G_{e_2} the path from p to q' is labelled with w_m.

Let us define an assignment σ for the variables of Q into G that sends x_i into node p', as defined above, and sends y_i to node q'. We can do this in a way that the assignment is consistent with the joins of Q (that is, with the repeated variables in different atoms of Q) since $(\bar{w}_1,\ldots,\bar{w}_p) \in \mathcal{P}_Q$. We conclude that σ is a homomorphism from Q to G, and then $\bar{t} \in Q(G)$ via the mapping σ, which is our desired contradiction. □

It is possible to prove that the two described classes of patterns are maximal with respect to tractability, as lifting either condition (C1) or (C2), raises the complexity. It can also be proved that the class of tame CRPQs is maximal to obtain tractability for patterns satisfying (C1) and (C2). All this is summarized in our last result. It shows that data complexity remains coNP hard if we restrict only to tame queries and patterns in $\mathcal{P}^{\text{re}}$ with bounded meaningful functions (i.e. if we lift condition (C1)), or if we restrict only to tame queries and patterns with bounded out degree (that is, we lift condition (C2)), or if we restrict patterns according to C1 and C2, but do not restrict to tame queries, even if we only allow arbitrary RPQs instead of arbitrary CRPQs.

Theorem 5.3.5

1. *There is a value $k \geq 0$ and a tame CRPQ Q such that* Pattern Certain Answers(Q) *is* coNP*-hard, even when restricted to patterns in $\mathcal{P}^{\text{re}} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$.*
2. *There is a value $d \geq 0$ and a tame CRPQ Q such that* Pattern Certain Answers(Q) *is*coNP*-hard, even when restricted to patterns in $\mathcal{OD}^{\text{re}}_{\leq d}$.*
3. *There exist a non-tame RPQ Q and values $d,k \geq 0$ such that* Pattern Certain Answers(Q) *is* coNP*-hard, even when restricted to patterns in $\mathcal{OD}^{\text{re}}_{\leq d} \cap \mathcal{MF}^{Q,\bar{t}}_{\leq k}$.*

Proof:

For **part 1** we use essentially the same reduction presented in the first part of Theorem 5.1.3, but this time over the extended alphabet $\{a,0,1,b\}$. We use the tame RPQ Q' defined as (x,w',y), where w' is the word $w' = a1011011101111b$.

From φ in 3CNF, we first construct graph pattern π as in the proof of the first part of Theorem 5.1.3, and then extend it to a pattern π' by adding two fresh node ids m and m', such that there is an outgoing edge labeled a from m into every node of π, and an incoming edge labeled b from each node of π into m'. Clearly, $(m,m') \in \text{CERTAIN}(Q',\pi')$ if and only if $\text{CERTAIN}(Q,\pi) = \texttt{true}$, where Q is the query used in the previous reduction. This shows that the problem of checking whether $(m,m') \in \text{CERTAIN}(Q',\pi')$, for a graph pattern π' of the form described above, is coNP-hard. Clearly, the outdegree of the class of patterns of the form π' is not bounded, as the outdegree of m depends on the size of π'. On the other hand, the number of meaningful functions for π' and Q' is clearly logarithmically bounded since Q' has no existentially quantified variables.

Part 2 follows immediately from the proof of the first part of Theorem 5.1.3. Indeed, the second part of this theorem shows that there is a CRPQ Q of the form $\exists x \exists y(x,w,y)$, where $w \in \Sigma^*$, such that checking whether $\text{CERTAIN}(Q,\pi) = \texttt{true}$, for a graph pattern π over Σ, is coNP-complete. Clearly, Q is tame. Moreover, the lower bound is proved by a reduction from 3SAT that constructs graph patterns with bounded outdegree (in particular, with maximal outdegree 3).

For **Part 3** we reduce again from 3SAT. Given a propositional formula φ, we construct in polynomial time a pattern π over $\Sigma = \{0,1\}$ exactly as in the proof of the first part of Theorem 5.1.3. Then from π we construct, in polynomial time, a pattern π' over the extended alphabet $\Sigma \cup \{a,b,c,d\}$. The pattern π' extends π with two things: (1) A fresh node id m with a self-loop labeled d that has incoming edges labeled b from each node node in π. (2) A directed binary tree in which each internal node is a fresh node id, the leaves are exactly the nodes of π and the edges of the tree are labeled a. Further, the root of the tree, m', has a self-loop labeled c.

It is not hard to see, using arguments presented in the proof of the first part of Theorem 5.1.3, that if Q' is $(x,ca^*1011011101111bd,y)$, then $(m',m) \in \text{CERTAIN}(Q,\pi)$ if and only if $\text{CERTAIN}(Q',\pi')$, where Q is the query $\exists x \exists y(x,1011011101111,y)$ and π and π' are as described above. We conclude that the problem of checking whether $(m',m) \in \text{CERTAIN}(Q',\pi')$ is coNP-hard for the class of patterns π' as defined in the previous paragraph. Notice that this class of patterns has bounded outdegree (in particular, outdegree 3), and the number of meaningful functions for π' and Q' is clearly logarithmically bounded. Indeed, the only meaningful function is the one that sends variable x to the root m' of the binary tree and y to m.

□

5.4 Certain Answers Via Constraint Satisfaction

Since our notion of certain answers is closely related to the existence of homomorphisms between structures, it makes sense to look at a similar problem that is also related to homomorphisms: the *constraint satisfaction problem* (CSP). The field of constrain satisfaction has identified many tractable restrictions and, what is equally important, provided many practical heuristics that help solve these intractable problems [Dechter, 2003, Kolaitis and Vardi, 2007].

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]. More precisely, 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 patterns in $\mathcal{P}^{\text{nv,lv}}$. The consequences of the results of this section are twofold. On one hand this enables us to use the machinery that has been developed through many years in solving constrain satisfaction problems, but on the other hand, it tells us that it will be probably difficult to obtain a precise characterization of tractability for query answering, since characterizing the instances of CSP that are in PTIME is a longstanding open problem [Dechter, 2003, Kolaitis and Vardi, 2007].

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 $\text{CSP}(\mathcal{M}_1, \mathcal{M}_2)$. We are working with the non-uniform version of CSP, i.e., the structure $\mathcal{M}_2$ is fixed. This problem can be NP-hard: for instance, if $\mathcal{M}_2$ is a cycle on three nodes, $\text{CSP}(\mathcal{M}_1, \mathcal{M}_2)$ is the 3-colorability of $\mathcal{M}_1$.

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].

Consider a pattern $\pi = (N, E)$ in $\mathcal{P}^{\text{nv,lv}}$, i.e., $E \subseteq N \times (\Sigma \cup \mathcal{W}) \times N$ for a finite set $\mathcal{W}$ of label variables. Let Q be an RPQ of form $\varphi(x, y) = (x, L, y)$, 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:

$$\begin{array}{rclrcl} Nodes & := & N & Edge & := & E \\ Lab_a & := & \{a\} & Src & := & \{n\} \\ Expr & := & \mathcal{W} & Sink & := & \{n'\} \end{array}$$

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:

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

Theorem 5.4.1 *For patterns $\pi\in\mathcal{P}^{\mathrm{nv,lv}}$, under the above 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 $T\in\{Nodes,Expr,(Lab_a)_{a\in\Sigma}\}$. 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 $c\in Source\Rightarrow f'(c)\in Source$. 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 $c\in Sink\Rightarrow f'(c)\in Sink$.

It just rests 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 $(p,D,p')\in Edges\Rightarrow(f'(p),f'(D),f'(p'))\in Edges$. Assume that $(p,D,p')\in Edges$. 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 $(f'(p),f'(D),f'(p')) \in Edges$.

We conclude that there is a solution for 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 $1 \leq j \leq t$. 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 $0 \leq j < t$. 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.

As we have mentioned, the case of data complexity corresponds to the non-uniform version of CSP, with $\mathcal{M}_Q$ fixed. In that case one can immediately obtain some tractable restrictions for the query answering problem. For example using known results on CSP [Dechter, 2003, Kolaitis and Vardi, 2007] we can conclude that if we have a class of patterns $\pi \in \mathcal{P}^{\text{nv,lv}}$ which, when viewed as the ternary relation $Edge$, has bounded treewidth, then the data complexity of RPQs over such a class is in PTIME (note that this is incompatible with Theorem 5.2.1 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 5.4.1 for patterns in $\mathcal{P}^{\mathrm{nv,re}}$ was shown in [Calvanese et al., 2000c]. Combining both techniques we can extend the result to all patterns in $\mathcal{P}^{\mathrm{nv,lv,re}}$, but at the cost of much more complex definitions of the structures $\mathcal{M}_\pi(n,n')$ and $\mathcal{M}_Q$ compared to those we used here. We prefer to skip the formal proof here for the sake of simplicity, and because we feel that Theorem 5.4.1 already conveys all of our ideas.

Chapter 6

Schema Mappings and Data Exchange

Data exchange, data integration and schema mapping management have received little attention so far in the graph database context, and tools from relational or XML databases suffer from important drawbacks when applied on graph-structured data. It has been pointed out [Kolaitis, 2005] that the study of patterns, and incomplete information in general, has proved to be essential in the development of the machinery for interoperability issues amongst relational and XML databases, which suggests that graph patterns shall provide the correct framework for the study of these issues amongst graph databases. In this chapter we show how to apply our previous results regarding graph patterns into the study of interoperability issues for graph databases, including schema mappings, data exchange and certain answers computation.

6.1 A Motivating Example

Suppose one wants to create a database containing the information of all computer science researchers and their papers in conferences. We want to store this information as graph database over alphabet $\Sigma_T = \{\texttt{makes}, \texttt{inConf}\}$, where each node represents either a particular researcher, one of its papers, or a conference in computer science, and the intuition behind the edges is the following. Author A is connected via an edge labelled `makes` to paper P if A is an author of P, and paper P is connected via edge labelled `inConf` to a conference C is P was presented in C.

In order to create our graph database, we do not collect this information from scratch. Instead, we retrieve it from the RDF Linked Data representation of DBLP [DBLP, 2013], a fragment of which is shown in Figure 6.1. This fragment contains all the schema information that we need: authors are connected to their papers via the label `creator`, and in particular conference papers have an outgoing path labelled with the word `partOf`$\cdot$`series` from the author to the node representing the given conference (for simplicity, we omit prefixes `dc:`, `dct:`, and `sw:` in edge labels).

In order to create our database of authors of conference papers, we need to exchange in-

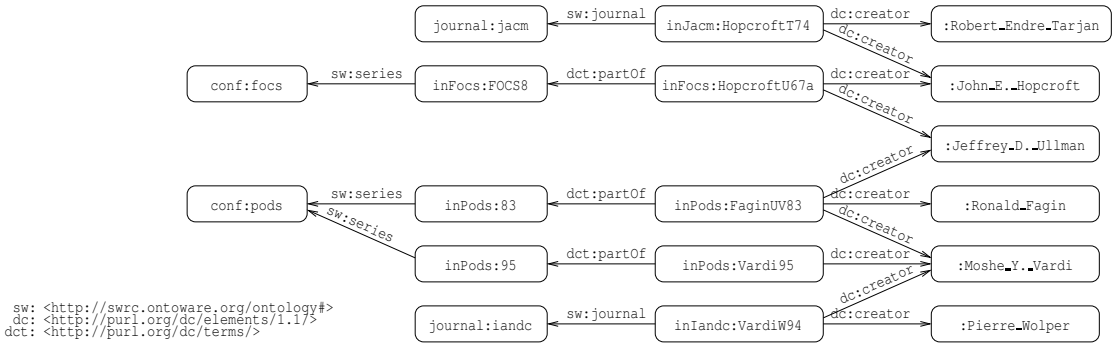
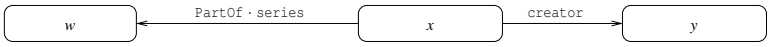


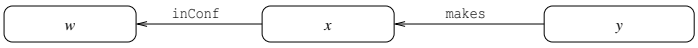
Figure 6.1: Graph G_1, a fragment of the RDF Linked Data representation of DBLP [DBLP, 2013] available at `http://dblp.l3s.de/d2r/`

formation from the graph in Figure 6.1 into our target graph database over $\Sigma_{\mathbf{T}}$. Usually, the instructions for this type of exchange are provided by means of a schema mapping. In a graph database context, schema mappings shall detail graph patterns in the *source* side (that is, the DBLP graph), and explain how these patterns are to be transformed into the *target* side (onto our graph of autorship).

The intuition behind our schema mapping is to find all authors of conference papers in the DBLP database, and then transfer them with the correct structure into our graph. This is done as follows. For every nodes A, P and C of G_1 such that the following pattern ξ_1 is realized in G_1 via a homomorphism that maps A to y, P to x and C to w:



Add to our *target* graph the information resulting of replacing y with A, x with P and w with C in the following pattern ξ_2:



We shall formally define our notion of schema mapping in Section 6.2. For now, let $Q_1(x,y,w) = (\xi_1, x, y, w)$ and $Q_2(x,y,w) = (\xi_2, x, y, w)$ be graph queries using patterns ξ_1 and ξ_2 presented above. Our schema mapping will then be formalized as the rule $Q_1(x,y,w) \rightarrow Q_2(x,y,w)$. With this information we have all the necessary ingredients to start populating our target graph; a possible result of exchanging the information in Figure 6.1 according to our mapping yields the graph database in Figure 6.2.

More complex mappings. As another example of an instance of data exchange, suppose we now want to extract from the graph G_1 of Figure 6.1 information about researchers and their workplaces, to populate, for instance, a network such as LinkedIn [LinkedIn, 2013]. The node of our graphs shall represent researchers and different research institutions, and the structure is given by the labels $\Sigma_{\mathbf{T}'} = \{\texttt{worksIn}, \texttt{workedIn}\}$. Each researcher is connected to one or more

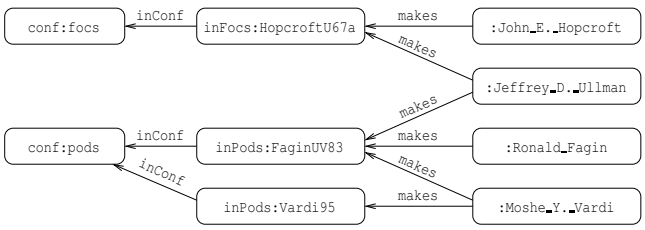
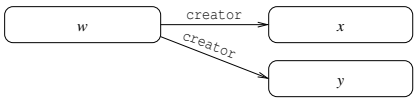


Figure 6.2: Graph G_2, a result of exchanging the information of the graph G_1 of Figure 6.1

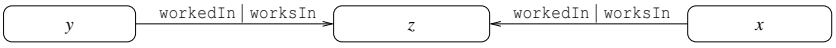
workplaces, label `worksIn` is for the current affiiliation of the researcher, and label `workedIn` connects a researcher with his or hers past affiliations.

Even if we do not know exactly the locations and/or affiliations of the researchers in the DBLP network, we can assume that, if they are co-authors, then they must have coincided at a given institute at a given point in the past. There are thus four possibilities: either both of them work on the same institute, or both of them worked in the same institute (but are in different places now), or one of them works in an institute where the other used to work as well. Under this circumstances it is safe to assume that, if two researchers are coauthors, then they should be related in our target graph via a combination of labels in the set $\{(\texttt{worksIn},\texttt{worksIn}),(\texttt{workedIn},\texttt{workedIn}),(\texttt{worksIn},\texttt{workedIn}),(\texttt{workedIn},\texttt{worksIn})\}$.

Our schema mapping for this exchange will now be given by the rule $Q'_1(x,y) \rightarrow Q_3(x,y)$, where $Q'_1 = (\xi'_1,x,y)$, with ξ'_1 the pattern



and $Q_3 = (\xi_3,x,y)$, with ξ_3 being the following pattern:



Note that, unlike the previous example, the graph query $Q_3(x,y)$ uses node variables and regular expressions in the edges. This time, if we start with the graph G_1 in Figure 6.1, it is not trivial to answer the question of what graph should be materialized. For example, we know that the pair of nodes `:John_E._Hopcroft` and `:Jeffrey_D._Ullman`: belong to $Q'_1(G_1)$. However, do not know the precise information about their workplaces. Since no other information about workplaces is given, for each pair of coauthors we have 4 possibilities, and the resulting number of choices is therefore exponential in the number of authors of our database.

This question is in fact is one of the fundamental problems in the area of data exchange. It is usually overcome by allowing the possibility of materializing incomplete information in the target side. And indeed, we shall overcome it by choosing to materialize graph patterns instead of standard graph databases. To be more precise, in the following sections we shall see that the

best alternative to materialize the exchange of graph G_1 with respect to this mapping is not a graph database, but instead a graph pattern.

There are several other fundamental questions in the areas of data exchange and schema mappings that deserve to be studied. Besides from the problem of choosing an instance to be materialized, we study in this chapter how to answer queries in a data exchange context. We also study these tasks from an algorithmic point of view, providing algorithms to perform them, and discuss about the optimality of our algorithms.

6.2 Schema Mappings

Schema mappings have been studied both in the relational [Fagin et al., 2005b] and the XML [Arenas and Libkin, 2008] scenario (see [Arenas et al., 2010], for a recent general presentation of the area). At a very high level, schema mappings are tuples of the form $\mathcal{M} = (\mathfrak{S}_1, \mathfrak{S}_2, \mathcal{T})$, where $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are appropriate schemas, and $\mathcal{T}$ is a finite set of rules of the form

$$\varphi_{\mathfrak{S}_1}(\bar{x}) \rightarrow \psi_{\mathfrak{S}_2}(\bar{x}), \tag{6.1}$$

with $\varphi_{\mathfrak{S}_1}(\bar{x})$ and $\psi_{\mathfrak{S}_2}(\bar{x})$ logical formulas over $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively, that specify the relationship between the two schemas.

Traditional rule specification has been carried out assuming that $\varphi_{\mathfrak{S}_1}$ and $\psi_{\mathfrak{S}_2}$ are suitable conjunctive queries (in other words, patterns) for the data model at hand. This class of rules is powerful enough to express how the schema $\mathfrak{S}_2$ is defined in terms of the existence of certain patterns over the schema $\mathfrak{S}_1$, but at the same time simple enough for practical purposes. For instance, in the relational case both $\varphi_{\mathfrak{S}_1}$ and $\psi_{\mathfrak{S}_2}$ correspond to usual conjunctive queries [Fagin et al., 2005b], while in the XML case they correspond to *tree pattern queries* [Arenas and Libkin, 2008], which are essentially acyclic conjunctive queries over XML trees including recursion at the atomic level.

In the same spirit, it seems completely natural to define mappings for graph databases by allowing rules of the form $\varphi_{\Sigma_1}(\bar{x}) \rightarrow \psi_{\Sigma_2}(\bar{x})$, with $\varphi_{\Sigma_1}(\bar{x})$ and $\psi_{\Sigma_2}(\bar{x})$ graph patterns over alphabets Σ_1 and Σ_2, respectively.

Definition 6.2.1 (graph mapping) *Let Σ_1 and Σ_2 be finite alphabets. A graph mapping $\mathcal{M}$ (or, simply, mapping, from now on) from Σ_1 to Σ_2 is a tuple $(\Sigma_1, \Sigma_2, \mathcal{T})$, where $\mathcal{T}$ is a finite set of rules of the form $\varphi_{\Sigma_1}(\bar{x}) \rightarrow \psi_{\Sigma_2}(\bar{x})$, with $\varphi_{\Sigma_1}(\bar{x})$ and $\psi_{\Sigma_2}(\bar{x})$ graph queries over Σ_1 and Σ_2, respectively.*

Example 6.2.2 *Recall alphabets $\Sigma_1 = \{\texttt{series}, \texttt{journal}, \texttt{partOf}, \texttt{creator}\}$, $\Sigma_2 = \{\texttt{makes}, \texttt{inConf}\}$ and $\Sigma_3 = \{\texttt{worksIn}, \texttt{workedIn}\}$, and rules $\mathcal{T}_1 = \{Q_1(x,y,w) \rightarrow Q_2(x,y,w)\}$ and $\mathcal{T}_1 = \{Q'_1(x,y) \rightarrow Q_3(x,y)\}$ as explained in the previous section. Then $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ are graph schema mappings.*

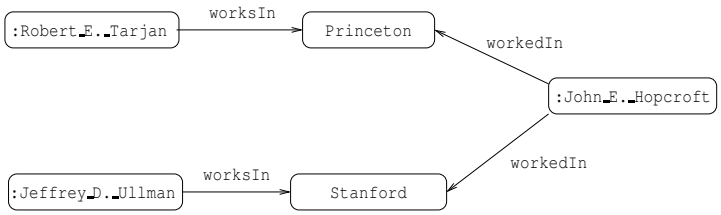


Figure 6.3: Graph G_3, (a fragment) of a solution for G_1 under $\mathcal{M}_{13}$

When the alphabet Σ_1 and Σ_2 are clear from context, we normally write the rules as $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$. In addition, when φ or ψ are graph queries in $\mathcal{P}^{\text{nv,re}}$, we often abuse the notation and denote the rules using the equivalent CRPQs. Thus, for instance, when referring to the mapping $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ above, we can also write the rule in $\mathcal{T}_{12}$ as

$$(x, \texttt{creator}, y) \wedge (x, \texttt{partOf} \cdot \texttt{series}, w) \ \rightarrow (y, \texttt{makes}, x) \wedge (x, \texttt{inConf}, w),$$

6.2.1 Solutions

If G_1 and G_2 are graph databases over Σ_1 and Σ_2, respectively, then the pair (G_1, G_2) satisfies the mapping $\mathcal{M}$, denoted $(G_1, G_2) \models \mathcal{M}$, if the following holds. For each rule in $\mathcal{T}$ of the form $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$ and each tuple $\bar{u}$ of node ids in G_1 such that $|\bar{u}| = |\bar{x}|$, we have that:

$$\bar{u} \in \varphi(G_1) \implies \bar{u} \in \psi(G_2). \tag{6.2}$$

Recall that $\varphi(\bar{x})$ and $\psi(\bar{x})$ are graph queries, and hence they are of the form $(\xi_\varphi, \bar{x})$ and $(\xi_\psi, \bar{x})$, where ξ_φ and ξ_ψ are graph patterns over Σ_1 and Σ_2, respectively. Moreover, assume that the nodes of ξ_ψ are $\bar{y} \cup \bar{x}$, and the label variables of ξ_ψ are $\bar{X}$. Therefore, statement (6.2) means that whenever $G_1 \models \xi_\varphi[\bar{u}/\bar{x}]$, for some tuple $\bar{u}$ of nodes in G_1 such that $|\bar{u}| = |\bar{x}|$, it is also the case that $G_2 \models \xi_\psi[\bar{u}/\bar{x}, \bar{v}/\bar{y}, \bar{A}/\bar{X}]$ for some tuples $\bar{v}$ of nodes and $\bar{A}$ of labels from Σ_2.

Following the usual data exchange terminology, we say that G_2 is a *solution* for G_1 under $\mathcal{M}$ (or simply a solution, if $\mathcal{M}$ is clear from the context) whenever $(G_1, G_2) \models \mathcal{M}$. The set of solutions for G_1 under $\mathcal{M}$, denoted $\text{Sol}_{\mathcal{M}}(G_1)$, is $\{G_2 \mid (G_1, G_2) \models \mathcal{M}\}$. Finally, the semantics $[\![\mathcal{M}]\!]$ of mapping $\mathcal{M}$ is the set $\{(G_1, G_2) \mid (G_1, G_2) \models \mathcal{M}\}$. Two mappings $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent* if $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$.

Example 6.2.3 *The graph G_2 in Figure 6.2 is a solution for the graph G_1 in Figure 6.1 under the mapping $\mathcal{M}_{12}$ in example 6.2.2. The graph G_3 in Figure 6.3 shows part of a solution for the graph G_1 under the mapping $\mathcal{M}_{13}$ in example 6.2.2*

6.3 Properties of Graph Mappings

6.3.1 Classification

Our mappings are based on graph patterns, so it is natural to classify them in the same way than we did with patterns. Let $\mathcal{P}^{\sigma}$ and $\mathcal{P}^{\sigma'}$ be classes of patterns. Then a graph mapping $(\Sigma_1, \Sigma_2, \mathcal{T})$ is a $\mathcal{P}^{\sigma}$-TO-$\mathcal{P}^{\sigma'}$ mapping if every rule in $\mathcal{T}$ is of form $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$, where $\varphi(\bar{x}) = (\chi_{\varphi}, \bar{x})$ is a graph query in $\mathcal{P}^{\sigma}$ and $\psi(\bar{x})$ is a graph query in $\mathcal{P}^{\sigma'}$.

Thus, for instance, the class of $\mathcal{P}^{\mathrm{nv,lv,re}}$-TO-$\mathcal{P}^{\mathrm{nv,lv,re}}$ mappings is the class of all graph mappings, and the class of $\mathcal{P}^{\mathrm{nv}}$-TO-$\mathcal{P}^{\mathrm{nv}}$ is the class of graph mappings that can be represented with relational mappings. The mapping $\mathcal{M}_{12}$ from Example 6.2.2 is a $\mathcal{P}^{\mathrm{nv,re}}$-TO-$\mathcal{P}^{\mathrm{nv}}$ mapping, and the mapping $\mathcal{M}_{13}$ from the same example is a $\mathcal{P}^{\mathrm{nv,re}}$-TO-$\mathcal{P}^{\mathrm{nv,re}}$ mapping.

6.3.2 Navigational exchange properties of schema mappings

The mappings we study in the present dissertation are designed for exchanging *tuples* of node ids from source to target. But what about more sophisticated *navigational* exchange tasks that are relevant in the graph database context, such as exporting entire paths of data satisfying certain conditions on the source side? We show that graph mappings can express some interesting exchange properties of this type. We do it by means of an example, but the techniques that we use can be generalized to show that graph mappings are capable of expressing a broad class of exchange properties based on the idea of exporting entire *paths* in the source graph that satisfy certain regular conditions.

Assume that we have source and target alphabets $\Sigma_{\mathbf{S}} = \{a, b, c, d\}$ and $\Sigma_{\mathbf{T}} = \{a', b', c', d'\}$, respectively. We wish to exchange data according to the following intuitive rule: Copy each path from the source to the target that starts and ends with an edge labeled c, and has at least two consecutive edges labeled a or at least one edge labeled b. Clearly, the regular expression $r = c \cdot \Sigma^* \cdot (aa + b) \cdot \Sigma^* \cdot c$ extracts from the source the pairs of nodes that are linked by a path satisfying the regular condition mentioned above. But how can we express our desired copying rule as a graph mapping?

We start by posing the following question: Under which circumstances do we have to copy an edge labeled a from the source as an edge labeled a' in the target, while navigating the source data? This is the case if one of the following holds:

1. Two consecutive edges labeled a, or one edge labeled b, is yet to appear. That is, we have read $R_1^1 = c \cdot \Sigma^*$ and have yet to read $R_2^1 = \Sigma^* \cdot (aa + b) \cdot \Sigma^* \cdot c$.

2. We are reading the first of the consecutive edges labeled a. That is, we have read $R_1^2 = c \cdot \Sigma^*$ and have to read $R_2^2 = a \cdot \Sigma^* \cdot c$.

3. We are reading the second of the two consecutive edges labeled a. That is, we have read $R_1^3 = c \cdot \Sigma^* \cdot a$ and need to read $R_2^3 = \Sigma^* \cdot c$.

4. We already read two consecutive edges labeled a or an edge labeled b, but are waiting for the final edge labeled c. In this case we have read $R_1^4 = c \cdot \Sigma^* \cdot (aa + b) \cdot \Sigma^*$ and are waiting to read $R_2^4 = \Sigma^* \cdot c$.

The pairs (R_1^i, R_2^i), for $1 \leq i \leq 4$, have the following property: A word of the form $w_1 \cdot a \cdot w_2$, for $w_1, w_2 \in (\Sigma_{\mathbf{S}})^*$, belongs to the language defined by r if and only if it belongs to $L(R_1^i) \cdot a \cdot L(R_2^i)$, for some $1 \leq i \leq 4$. We say then that the set $\{(R_1^i, R_2^i) \mid 1 \leq i \leq 4\}$ is a *remnant* of r with respect to a. This remnant allows us to create the rules that will copy the edges labeled a as edges labeled a', precisely when it is needed. In the same way we can define remnants of r with respect to b, c and d, respectively.

Then the mapping that defines our desired copying rule consists of the rules:

$$\exists z \exists w \big((z, R_1^i, x) \wedge (x, a, y) \wedge (y, R_2^i, w) \big) \rightarrow (x, a', y),$$

for $1 \leq i \leq 4$, together with similar rules for the remnants of b, c and d.

6.4 Graph Data Exchange

Data exchange is one of the main applications of schema mappings [Fagin et al., 2005b, Kolaitis, 2005, Barceló, 2009, Arenas et al., 2010]. In this section we study *graph data exchange* under the mappings we defined in the previous section, that is, we use graph mappings for specifying how to translate graph data from a source into a target schema. Our study focuses in two of the main problem in data exchange: materializing a target solution, and query answering. To define these in more detail, assume that we have a mapping $\mathcal{M}$ from a source alphabet $\Sigma_{\mathbf{S}}$ to a target alphabet $\Sigma_{\mathbf{T}}$, and a source graph database $G_{\mathbf{S}}$.

The first problem we study is the data exchange problem. It consists in choosing which solution $G_{\mathbf{T}}$ for $G_{\mathbf{S}}$ under $\mathcal{M}$ to materialize, and how to materialize it. We have mentioned that the study of patterns and incomplete information has been crucial for the development of relational and XML data exchange theories. In the following section we show that this is also the case for graph patterns. In fact, we show that the best alternative for materializing solutions comes in forms of graph patterns. Afterwards, in section 6.4.2, we study how to answer queries that are posed over the target alphabet. Once again, we want to compute the certain answers, or the answers that are independent of which solution one materializes. Here we also take advantage of our previous results in querying graph patterns, and show how to reduce the problem of query answering in data exchange to the problem of querying graph patterns.

6.4.1 Universal representatives

Given the semantics of mappings, we know that there are infinitely many solutions for a given graph database $G_{\mathbf{S}}$. This phenomenon also occurs in relational and XML data exchange [Fagin et al., 2005b, Arenas and Libkin, 2008]. Thus, in data exchange one usually wants to compute a "universal representative" [Fagin et al., 2005b, Arenas et al., 2010, Arenas et al., 2011b], which is (in very broad terms) a finite representation of the set of all solutions. These representatives are normally tree patterns (in case of XML) or naive tables (in the relational case), so it is natural to start the study of graph data exchange with a definition based on graph patterns.

Definition 6.4.1 (Universal representative) *Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a mapping and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. A graph pattern $\pi_{\mathbf{T}}$ is a* universal representative *of $G_{\mathbf{S}}$ under $\mathcal{M}$, if* $\mathrm{Sol}_{\mathcal{M}}(G_{\mathbf{S}}) = [\![\pi_{\mathbf{T}}]\!]_{\Sigma_{\mathbf{T}}}$.

Example 6.4.2 *Recall the mapping $\mathcal{M}_{12} = \Sigma_1, \Sigma_2, \mathcal{T}_{12}$ given by the rule*

$$(x, \texttt{creator}, y) \wedge (x, \texttt{partOf} \cdot \texttt{series}, w) \rightarrow (y, \texttt{makes}, x) \wedge (x, \texttt{inConf}, w),$$

The graph database G_2 shown in Figure 6.2 is a a universal representative of the graph database G_1 (shown in Figure 6.1) under $\mathcal{M}_{12}$.

Consider now mapping $\mathcal{M}_{13} = \Sigma_1, \Sigma_3, \mathcal{T}_{13}$ given by the rule

$$\exists z \exists w (z, \texttt{creator}, x) \wedge (z, \texttt{creator}, y) \wedge (z, \texttt{partOf}, w) \rightarrow \exists z (x,\ \texttt{worksIn} \mid \texttt{workedIn}, z) \wedge (y,\ \texttt{worksIn} \mid \texttt{workedIn}, z)$$

It is not difficult to show that every universal representative (in form of a graph pattern) for G_1 under $\mathcal{M}_{13}$ must use regular expressions in the edges. In other words, there is no complete graph that is a universal representative for G_1 under $\mathcal{M}_{13}$. For example, a possible universal representative for G_1 under $\mathcal{M}_{13}$ is a pattern containing a fresh node n and edges

$$(A,\ \texttt{worksIn} \mid \texttt{workedIn}, n) \text{ and } (B,\ \texttt{worksIn} \mid \texttt{workedIn}, n),$$

for each pair of nodes A and B in G_1 that satisfy the left hand side of $\mathcal{M}_{13}$, i.e. the query $\exists z \exists w (z, \texttt{creator}, x) \wedge (z, \texttt{creator}, y) \wedge (z, \texttt{partOf}, w)$.

In what follows we show how a universal representative can be computed for each source graph database and mapping. This universal representative will turn out to be crucial for answering queries in graph data exchange.

Universal representative computation. The standard techniques for constructing universal representatives in relational data exchange are based on the *chase* [Fagin et al., 2005b]. Those

techniques can be adapted in a very simple way to design a procedure that constructs universal representatives also in the graph database context. Such procedure works in polynomial space (and, thus, in single exponential time) when the input consists of a mapping and a source graph database (that is, in combined complexity). It works in nondeterministic logarithmic space (and, thus, in polynomial time) when the mapping is fixed and the input consists of the source graph database only (thus, in data complexity).

Proposition 6.4.3 *There is a procedure that, given a mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ and a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, computes a graph pattern $\pi_{\mathbf{T}}$ that is a universal representative of $G_{\mathbf{S}}$ under $\mathcal{M}$ in* PSPACE. *The procedure works in* PTIME *if the mapping $\mathcal{M}$ is assumed to be fixed.*

Proof: We use a standard *chase* procedure. Initialize the universal representative π to be the empty pattern. For each rule $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$ in $\mathcal{T}$, such that $\bar{x} = (x_1, \ldots, x_n)$, $\varphi = (\xi_\varphi, \bar{x})$ is a graph query over $\Sigma_{\mathbf{S}}$, $\psi = (\xi_\psi, \bar{x} \cup \bar{y})$ is a graph query over $\Sigma_{\mathbf{T}}$, the node variables of ξ_ψ are $\bar{x} \cup \bar{y}$, where $\bar{y} = (y_1, \ldots, y_m)$; and the label variables of ξ_ψ are $X_1, \ldots, X_\ell$, do the following. For every tuple $\bar{u} = (u_1, \ldots, u_n)$ of node ids in $G_{\mathbf{S}}$ such that $G_{\mathbf{S}} \models \xi_\varphi[\bar{u}/\bar{x}]$, choose tuples $\bar{w} = (w_1, \ldots, w_m)$ of fresh null values, and $\bar{W} = W_1, \ldots, W_\ell$ of fresh label variables, and define $\pi = \pi \cup \xi_\psi[\bar{u}/\bar{x}, \bar{w}/\bar{y}, \bar{W}/\bar{X}]$. Clearly this procedure works in polynomial space. We now show that it outputs as result a graph pattern π over $\Sigma_{\mathbf{T}}$ that is a universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$.

To see that $[\![\pi]\!] \subseteq \mathrm{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$, take an arbitrary graph $G \in [\![\pi]\!]$. We need to show that G belongs to $\mathrm{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$. To that extent, consider an arbitrary rule $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$ in $\mathcal{T}$, such that $\bar{x} = (x_1, \ldots, x_n)$, $\varphi = (\xi_\varphi, \bar{x})$ is a graph query over $\Sigma_{\mathbf{S}}$, $\psi = (\xi_\psi, \bar{x} \cup \bar{y})$ is a graph query over $\Sigma_{\mathbf{T}}$, the node variables of ξ_ψ are $\bar{x} \cup \bar{y}$, where $\bar{y} = (y_1, \ldots, y_m)$; and the label variables of ξ_ψ are $X_1, \ldots, X_\ell$, and assume that there is a tuple $\bar{u} = (u_1, \ldots, u_n)$ of node ids in $G_{\mathbf{S}}$ such that $G_{\mathbf{S}} \models \xi_\varphi[\bar{u}/\bar{x}]$. Then $\pi = \pi' \cup \xi_\psi[\bar{u}/\bar{x}, \bar{w}/\bar{y}, \bar{W}/\bar{X}]$, where $\bar{w}$ is a tuple of fresh null values, $\bar{W}$ is a tuple of fresh label variables, and π' is a graph pattern. Given the homomorphism from π to G we can now construct a homomorphism from $\xi_\psi[\bar{u}/\bar{x}, \bar{w}/\bar{y}, \bar{W}/\bar{X}]$ to G, which proves that G is a solution for $G_{\mathbf{S}}$ under $\mathcal{M}$ since the graph G, the rule and the tuple $\bar{u}$ were arbitrarily chosen. For the other direction, take an arbitrary graph $G \in \mathrm{Sol}_{\mathcal{M}}(G_{\mathbf{S}})$. We need to construct a homomorphism from π to G. But since every variable introduced in the chase procedure is a fresh variable, we can just take the union of all the homomorphisms from the right hand side of a rule in $\mathcal{T}$ to G, that are known to exist every time there is a tuple $\bar{u}$ of nodes witnessing the left hand side of this same rule.

Regarding data complexity, notice that if $\mathcal{M}$ is fixed then each graph query can be evaluated in PTIME. Furthermore, since the size of $\bar{x}$ and $\bar{y}$ is constant, we can easily build a PTIME machine that checks all possible combinations of nodes from $G_{\mathbf{S}}$ of size k. We then proceed exactly as before. □

The following follows directly from the proof above. We use this property extensively in this Chapter.

Corollary 6.4.4 *Let σ_1, σ_2 be subsets of $\{\text{nv}, \text{lv}, \text{re}\}$. If $\mathcal{M}$ is a $\mathcal{P}^{\sigma_1}$-to-$\mathcal{P}^{\sigma_2}$ mapping from $\Sigma_\mathbf{S}$ to $\Sigma_\mathbf{T}$, then for every graph $G_\mathbf{S}$ over $\Sigma_\mathbf{S}$ there exists a pattern in $\mathcal{P}^{\sigma_2}$ that is a universal representative for $G_\mathbf{S}$ under $\mathcal{M}$.*

Traditional data exchange analysis has been carried out in terms of data complexity (save for a few exceptions [Kolaitis et al., 2006, Arenas et al., 2011a]). But as we mentioned in the Introduction, this analysis is no longer appropriate for graph data exchange, due to the vast volumes of data stored by graph data applications. For instance, it is not difficult to construct a family of mappings $\mathcal{M}_n$ and source graph databases G_n of size $O(n)$, such that any universal representative of G_n under $\mathcal{M}_n$ is of size comparable to $|G_n|^{|\mathcal{M}_n|}$. Computing this representative is prohibitively expensive for big source databases, even for small mappings. Furthermore, the problem remains computationally hard in combined complexity even when this exponential blowup can be avoided. Recall that $\text{FP}^{\text{NP[log]}}$ is the class of functions that can be computed in polynomial time using a logarithmic number of calls to an NP oracle [Krentel, 1988].

Proposition 6.4.5 *The problem of computing a universal representative for a graph database $G_\mathbf{S}$ under a mapping $\mathcal{M} = (\Sigma_\mathbf{S}, \Sigma_\mathbf{T}, \mathcal{T})$ is $\text{FP}^{\text{NP[log]}}$-hard, even if restricted to inputs $G_\mathbf{S}$ and $\mathcal{M}$ such that there is a universal representative of $G_\mathbf{S}$ under $\mathcal{M}$ of size $p(|G_\mathbf{S}|)$, for a fixed polynomial p.*

Proof: We use a reduction from CHROMATIC NUMBER, a problem defined as follows: given a (standard, non database) connected graph G, compute its chromatic number. This problem was shown to be $\text{FP}^{\text{NP[log]}}$-hard in [Krentel, 1988].

Let then $G = (N, E)$ be the input to CHROMATIC NUMBER, and assume $N = \{n_1, \ldots, n_k\}$. We associate to each node n_j in N a corresponding variable z_j, and define the CRPQ $Q(x)$ over alphabet $\{a, b\}$ as $Q(x) = \big(\bigwedge_{(n_i,n_j)\in E}(z_i, a, z_j)\big) \wedge (x, b, z_1)$. The idea is that the underlying graph of the query Q resembles the structure of G, but it has one additional edge, namely (x, b, z_1).

Construct the mapping $\mathcal{M} = (\Sigma_\mathbf{S}, \Sigma_\mathbf{T}, \mathcal{T})$ as follows: $\Sigma_\mathbf{S} = \{a, b\}$, $\Sigma_\mathbf{T} = \{c\}$ and $\mathcal{T} = \{Q(x) \to (x, c, x)\}$. Finally, $G_\mathbf{S}$ contains, for each $1 \leq i \leq k$, a clique of a-labelled edges of size i, each of these cliques constructed using fresh node values. Graph $G_\mathbf{S}$ also contains, for each $1 \leq i \leq k$, a fresh node v_i and an edge of form (v_i, b, u_i), that connects the element v_i with an arbitrary node u_i of the clique of size i in $G_\mathbf{S}$.

By the construction of $\mathcal{M}$, there is always a universal representative for $G_\mathbf{S}$ under $\mathcal{M}$ that is of size $|G_\mathbf{S}|$. Furthermore, it is not difficult to see that any universal representative $G_\mathbf{T}$ for $G_\mathbf{S}$ under $\mathcal{M}$ contains a self loop with the element v_i if and only if $v_i \in Q(x)$. Since the latter holds if and only if there is a homomorphism from G to K_i, we obtain that $G_\mathbf{T}$ contains the edge

(v_i, c, v_i) if and only if G is i-colorable. One can then easily compute the chromatic number of G from $G_{\mathbf{T}}$, by checking the lowest v_i for which $G_{\mathbf{T}}$ contains the self loop (v_i, c, v_i). This shows that by computing $G_{\mathbf{T}}$ one can effectively compute the chromatic number of any connected graph, which finishes the reduction. □

It is thus crucial for the development of graph data exchange tools to identify relevant classes of mappings that allow for efficient universal representative computation in combined complexity. We deal with this important issue in Sections 6.5 and 6.6. Those results shall also be used to achieve efficient query answering in graph data exchange.

6.4.2 Query answering

As for other applications that need to query incomplete information, in data exchange one is typically interested in computing the *certain* answers of queries [Fagin et al., 2005b, Arenas and Libkin, 2008, Arenas et al., 2010]. Recall that, for graph patterns, we defined certain answers as the answers that hold in all graphs represented by that pattern. Consequently, in data exchange we define them as the answers that hold regardless of the solution one chooses to materialize. In formal terms, given a mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, and a query Q over $\Sigma_{\mathbf{T}}$, we define the certain answers of Q with respect to $G_{\mathbf{S}}$ under $\mathcal{M}$, denoted by $\text{CERTAIN-DE}_{\mathcal{M}}(Q, G_{\mathbf{S}})$[1], as the set $\bigcap\{Q(G_{\mathbf{T}}) \mid G_{\mathbf{T}} \in \text{Sol}_{\mathcal{M}}(G_{\mathbf{S}})\}$. We are thus interested in the following problem:

Problem:	DATA EXCHANGE CERTAIN ANSWERS
Input:	Mapping $\mathcal{M}$ from $\Sigma_{\mathbf{S}}$ to $\Sigma_{\mathbf{T}}$, graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, k-ary graph query Q over $\Sigma_{\mathbf{T}}$, and k-ary tuple $\bar{v} \in \mathbf{V}^k$.
Question:	Is $\bar{v} \in \text{CERTAIN-DE}_{\mathcal{M}}(Q, G_{\mathbf{S}})$?

Notice that the source graph database, the mapping and the query are part of the input, and thus, we are considering the combined complexity of the problem.

Usually in data exchange one relies on the universal representative to compute query answering. Our case is not different. Consider a mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, and arbitrary graph database and queries $G_{\mathbf{S}}$ and Q, respectively. Our algorithm to compute the certain answers of Q with respect to $G_{\mathbf{S}}$ under $\mathcal{M}$ is given by the following two-step approach:

1. Compute a universal representative $\pi_{\mathbf{T}}$ for $G_{\mathbf{S}}$ under $\mathcal{M}$. By Proposition 6.4.3, $\pi_{\mathbf{T}}$ is a graph pattern, and it can be computed in PSPACE.

2. Compute the certain answers of Q over $\pi_{\mathbf{T}}$, as explained in Chapter 4. by Proposition 4.3.1, this can be performed in EXPSPACE.

[1] In [Barceló et al., 2013a] and most data exchange papers the term $\text{CERTAIN}_{\mathcal{M}}(Q, G_{\mathbf{S}})$ is used instead. We use $\text{CERTAIN-DE}_{\mathcal{M}}(Q, G_{\mathbf{S}})$ to avoid confusion with certain answers over graph patterns.

These two steps, applied naively, yield an algorithm that runs in double exponential space. However, by carefully inspecting the proof of Theorem 4.3.1, one realizes that the complexity of computing certain answers for data exchange remains the same as the complexity of computing certain answers for graph patterns. Hardness follows again from an easy adaptation of the EXPSPACE-hardness proof for querying graph patterns.

Theorem 6.4.6 *The complexity of* DATA EXCHANGE CERTAIN ANSWERS *is* EXPSPACE-*complete.*

Proof: Let $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ be a mapping, Q a graph query and $G_{\mathbf{S}}$ a graph database over $\Sigma_{\mathbf{S}}$. By Proposition 6.4.3 there is a universal representative $\pi_{\mathbf{T}}$ for $G_{\mathbf{S}}$ under $\mathcal{M}$ of size at most exponential in the size of $G_{\mathbf{S}}$, Q and $\mathcal{M}$. Therefore, the automaton that represent all possible codifications for the graph databases that belong to $[\![\pi_{\mathbf{T}}]\!]$, as explained in the proof of Proposition 4.3.1, is double exponential. But this carries no extra computational cost, since the other automata constructed in that proof are already of double exponential space, and we check the nonemptiness of the resulting intersection via a standard on-the-fly procedure that runs in EXPSPACE even if all the automata that we are intersecting are of double exponential space. Hardness can be proved by constructing a data exchange setting $\mathcal{M}$ and a source graph $G_{\mathbf{S}}$ such that the representative for $G_{\mathbf{S}}$ under $\mathcal{M}$ is precisely the pattern used to show EXPSPACE hardness for containment of CRPQs in [Calvanese et al., 2000b]. □

We are left facing a complexity that is prohibitively high for practical purposes. Results from Chapter 4 tells us that, in order to obtain tractability from a combined complexity point of view, we cannot use graph queries as our language of choice, since answering queries is NP-hard even for the most simplest patterns and queries. The following shows that this is also the case for graph data exchange settings.

Proposition 6.4.7 *The complexity of* DATA EXCHANGE CERTAIN ANSWERS *is* NP-*hard, even if restricted to* $\mathcal{P}$-TO-$\mathcal{P}$ *mappings and RPQs.*

Proof: It is well known that the following problem is NP-hard: The input is a graph pattern in $\mathcal{P}$ and a nonempty graph G, and the question is whether such pattern can be realized in the graph (i.e., if there is a homomorphism prom the pattern to G). We can then reduce from this problem as follows. Given pattern π and graph G over Σ, let $Q = (\pi, \cdot)$ be the boolean graph query given by π, and construct the following mapping from Σ to $\{a, b\}$: it contains rules $Q \to (x, a, y)$ and a rule of form $(x, d, y) \to (x, b, y)$ for each symbol $d \in \Sigma$. Furthermore, consider boolean RPQ $Q() = (x, a, y)$. It follows that the certain answers for Q over G under $\mathcal{M}$ are true if and only if the evaluation of π over G is true, i.e., if π is realized in G. □

There are other possibilities of restrictions that reduce the combined complexity of query answering. Typical examples are queries based on acyclicity constraints [Yannakakis, 1981],

or queries with fixed number of variables [Vardi, 1995]. Amongst the most natural classes of queries that fulfill these restrictions are *binary* queries, such as RPQs. In the following section we show how to obtain efficient query answering for a class of mappings based on binary queries. RPQs, as we shall see, are too limited in expressive power, but there are other binary query languages which are much more expressive than RPQs, and that will aid us in obtaining the right tradeoff between complexity and expressive power in our context.

6.5 A Practical Class of Mappings

We have seen in the previous sections that the definition of graph mappings based on patterns ties us almost inevitably to a high computational complexity when solving two of the most important tasks of data exchange, namely materialization and query answering.

The standard way of constructing a universal representative in data exchange, and the approach we adopted for the proof of Proposition 6.4.3 is to evaluate the left-hand side of each rule against the source database, and then populate the target as defined by the right-hand side of the respective rules. Hence the most natural way of obtaining restricted classes of graph mappings, for which universal representatives can be computed in polynomial time, is by restricting the left-hand side of rules to queries that allow for efficient computation of its answer set (instead of arbitrary patterns, or even patterns in $\mathcal{P}$, that are much more expensive from a computational point of view).

Formally, let $\mathcal{C}$ be a class of graph queries such that the problem of computing $Q(G)$, for a given query $Q(\bar{x}) \in \mathcal{C}$ and a graph database G, can be solved in polynomial time. Then there is a polynomial time procedure that, given a mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, in which each left-hand side of a rule in $\mathcal{T}$ is a query in $\mathcal{C}$, and a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, computes a universal representative of $G_{\mathbf{S}}$ under $\mathcal{M}$.

There are several relevant classes $\mathcal{C}$ of queries that satisfy the condition mentioned above (i.e. the set $Q(G)$ can be computed in polynomial time, for each graph database G and query $Q \in \mathcal{C}$). These include, for instance, syntactic restrictions of patterns, based on *acyclicity* [Yannakakis, 1981], for queries of fixed arity.[2]

In this section we study mappings based on binary graph queries, instead of arbitrary ones. More precisely, we investigate mappings that are defined using *nested regular expressions*, a powerful yet computationally simple language. In the rest of this chapter we show how, by using these expressions to construct our mappings, we achieve a very interesting equilibrium in the ever-present complexity - expressivity tradeoff of schema mappings. Not only we show that the computational cost of many data exchange tasks is much less for this type of mappings, but,

[2] Being precise, those restrictions have been defined for relational conjunctive queries, but the same complexity bounds apply for graph queries, if the structural restrictions are defined on the underlying undirected graphs of the patterns defining these queries.

as we demonstrate in Section 6.5.3, these mappings still allow us to express many interesting properties.

6.5.1 Nested regular expressions

Let Σ be a finite alphabet. *Nested regular expressions* (NREs) over Σ extend regular expressions with an *existential nesting test* operator $[(\cdot)]$ (or just nesting operator, for short), and an *inverse* operator a^-, over each $a \in \Sigma$ [Pérez et al., 2010].

The syntax of NREs is given by the following grammar.

$$R \;:=\; \varepsilon \;\mid\; a \;(a \in \Sigma) \;\mid\; a^- \;(a \in \Sigma) \;\mid\; R \cdot R \;\mid\; R^* \;\mid\; R + R \;\mid\; [R] \quad (6.3)$$

We formalize the semantics of an NRE R over a graph G as a binary relation $[\![R]\!]_G$ defined as expected: (a is a symbol in Σ, and n, n_1 and n_2 are arbitrary NREs):

$$\begin{aligned}
[\![\varepsilon]\!]_G &= \{(u,u) \mid u \text{ is a node id in } G\} \\
[\![a]\!]_G &= \{(u,v) \mid (u,a,v) \in G\} \\
[\![a^-]\!]_G &= \{(u,v) \mid (v,a,u) \in G\} \\
[\![n_1 \cdot n_2]\!]_G &= [\![n_1]\!]_G \circ [\![n_2]\!]_G \\
[\![n_1 + n_2]\!]_G &= [\![n_1]\!]_G \cup [\![n_2]\!]_G \\
[\![n^*]\!]_G &= [\![\varepsilon]\!]_G \cup [\![n]\!]_G \cup [\![n \cdot n]\!]_G \cup [\![n \cdot n \cdot n]\!]_G \cup \cdots \\
[\![\,[n]\,]\!]_G &= \{(u,u) \mid \text{there exists } v \text{ s.t. } (u,v) \in [\![n]\!]_G\},
\end{aligned}$$

where the symbol $\circ$ denotes the usual composition of binary relations, that is, $[\![n_1]\!]_G \circ [\![n_2]\!]_G = \{(u,v) \mid \text{there exists } w \text{ s.t. } (u,w) \in [\![n_1]\!]_G \text{ and } (w,v) \in [\![n_2]\!]_G\}$.

A nested path query, or NPQ for short, is a query of form $Q(x,y) = (x,R,y)$, where R is a nested regular expression. The semantics is evident: Given graph database G and an NPQ $Q(x,y) = (x,R,y)$, the set $Q(G)$ is just $[\![R]\!]_G$.

Note that NPQ strictly contains the language of 2 way RPQs, or 2RPQs, the extension to RPQs with the inverse operator proposed in [Calvanese et al., 2000b].

Example 6.5.1 *The following NPQ matches, in the graph database G shown in Figure 6.1, all pairs (u,v) such that u and v are connected by a coauthorship sequence that only considers conference papers:*

$$Q(x,y) = (x,\ (\texttt{creator}^- \cdot [\texttt{partOf} \cdot \texttt{series}] \cdot \texttt{creator})^+ ,v)$$

Let us give the intuition of the evaluation of this expression. Assume that we start at node u. The (inverse) edge $\texttt{creator}^-$ *forces us to navigate from u to a paper v created by*

u. Then the existential test $[\texttt{partOf} \cdot \texttt{series}]$ *is used to check that from v we can navigate to a conference (and thus, v is a conference paper). Finally, we follow edge* `creator` *from v to an author w of v. The* $(\cdot)^+$ *over the expression allows us to repeat this sequence several times. For instance,* $(\texttt{:John\_E.\_Hopcroft}, \texttt{:Moshe\_Y.\_Vardi})$ *is in* $Q(G)$*, but* $(\texttt{:John\_E.\_Hopcroft}, \texttt{:Pierre\_Wolper})$ *is not in* $Q(G)$*. It can be proved that the use of nesting is essential for expressing this query.*

With respect to complexity of query evaluation, NPQs are not only *polynomial* in combined complexity (i.e. when both the database and the query are given as input), but they can be evaluated linearly in both the size of the database and the expression. Given a graph database G and an NPQ Q, we use $|G|$ to denote the size of G (in terms of the number of egdes $(u,a,v) \in G$), and $|Q|$ to denote the size of Q.

Proposition 6.5.2 *[Pérez et al., 2010] Checking, given a graph database G, a pair of nodes* (u,v)*, and an NPQ Q, whether* $(u,v) \in Q(G)$*, can be done in time* $O(|G| \cdot |Q|)$*.*

Inverse of a nested regular expression. Along this Chapter we make use of the notion of *inverse* of an NRE R, which is an NRE $(R)^{-1}$ such that for each graph database G and pair of node ids (u,v) in G it is the case that $(u,v) \in [\![R]\!]_G$ if and only if $(v,u) \in [\![(R)^{-1}]\!]_G$. It is easy to prove that the class of NREs is closed under inverse, and hence that $(R)^{-1}$ is well-defined.

The following inductive construction shows how to compute the inverse of an NRE.

$$
\begin{aligned}
\varepsilon^{-1} &= \varepsilon \\
a^{-1} &= a^-, \text{ for each } a \in \Sigma \\
(a^-)^{-1} &= a, \text{ for each } a \in \Sigma \\
(R_1 \cdot R_2)^{-1} &= R_2^{-1} \cdot R_1^{-1} \\
(R^*)^{-1} &= (R^{-1})^* \\
(R_1 + R_2)^{-1} &= R_1^{-1} + R_2^{-1} \\
[R]^{-1} &= [R]
\end{aligned}
$$

It is straightforward to prove the correctness of the construction. Moreover, a polynomial construction can be implemented by considering any binary parse tree of the expression, and then performing the operations in a bottom-up fashion.

6.5.2 NPQ-restricted mappings

We shall now focus on a particular class of mappings that satisfies our tractability conditions and allows for a simple definition: The class of mappings such that the left-hand side of each rule is an NPQ. We pinpoint the precise complexity of the problem of computing universal

representatives for this class, and discuss about the expressiveness of the class for graph data exchange purposes. It is worth remarking that the other natural choice, restricting left-hand sides of rules to acyclic CRPQs, yields a much less expressive class of mappings, and thus all of the results below hold for acyclic CRPQs as well.

Definition 6.5.3 (NPQ-restricted mapping) *Let Σ_1 and Σ_2 be finite alphabets. An NPQ-restricted mapping $\mathcal{M}$ from Σ_1 to Σ_2 is a tuple $(\Sigma_1, \Sigma_2, \mathcal{T})$, where $\mathcal{T}$ is a finite set of rules of the form $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$, with $\varphi(\bar{x})$ an NPQ over Σ_1 and $\psi(\bar{x})$ a graph query over Σ_2.*

Note that one could have defined these mappings in a much more symmetric way, by allowing the right side of the rules of our mappings to be based on graph queries built using patterns with nested regular expressions in the edges. However, this would over complicate the presentation, without adding up much expressive power from a practical point of view. For this reasons we have chosen to introduce NPQ-restricted mappings in their present form.

6.5.3 Expressivity of NPQ-restricted mappings

One evidently loses expressive power when focusing on mappings given by binary queries, instead of arbitrary graph patterns. Nevertheless, in this section we show that this loss is much smaller that what would appear at a first glance, as NPQs, and therefore NPQ-restricted mappings, can express many interesting properties, some of which can not be expressed in any of the mapping languages based on graph patterns.

Let us come back to the example presented in Section 6.3.2. We had source and target alphabets $\Sigma_{\mathbf{S}} = \{a, b, c, d\}$ and $\Sigma_{\mathbf{T}} = \{a', b', c', d'\}$, respectively, and wish to exchange data according to the following intuitive rule: Copy each path from the source to the target that starts and ends with an edge labeled c, and has at least two consecutive edges labeled a or at least one edge labeled b. The regular expression $r = c \cdot \Sigma^* \cdot (aa + b) \cdot \Sigma^* \cdot c$ extracts from the source the pairs of nodes that are linked by a path satisfying the regular condition mentioned above. We showed in Section 6.3.2 how to construct a graph mapping to express this exchange rule. Interestingly, we now show how to express it with an NPQ restricted mapping.

Recall that in Section 6.3.2 we pointed out how to compute the remnants of r with respect to a, which are pairs of nested regular expressions (R_1^i, R_2^i), for $1 \leq i \leq 4$, that have the following property: A word of the form $w_1 \cdot a \cdot w_2$, for $w_1, w_2 \in (\Sigma_{\mathbf{S}})^*$, belongs to the language defined by r if and only if it belongs to $L(R_1^i) \cdot a \cdot L(R_2^i)$, for some $1 \leq i \leq 4$.

We can now express our mapping with rules:

$$(x, [(R_1^i)^{-1}] \cdot a \cdot [R_2^i], y) \rightarrow (x, a', y), \quad \text{for } 1 \leq i \leq 4,$$

together with similar rules for the remnants of b, c and d. Note that the expression $[(R_1^i)^{-1}] \cdot a \cdot [R_2^i]$ is an NRE, and therefore we are constructing an NPQ-restricted mapping. Furthermore,

the use of nesting in the queries is crucial for expressing these types of rules without using more expressive patterns, or queries with conjunction.

The example above is a consequence of the following stronger result. Let $\mathcal{C}$ be a class of mappings. As usual, we say that a mapping $\mathcal{M}$ (not necessarily in $\mathcal{C}$) can be *expressed* as a $\mathcal{C}$-mapping if one can find an equivalent mapping $\mathcal{M}'$ in $\mathcal{C}$ that is equivalent to $\mathcal{M}$.

Proposition 6.5.4

1. *Every* $\mathcal{P}^{\text{nv,re}}$-TO-$\mathcal{P}^{\text{nv,lv,re}}$*-mapping* $\mathcal{M}$ *such that each of its rules is of form* $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$*, with* φ *a binary graph query whose underlying graph is acyclic and connected can be expressed as an NPQ-restricted-mapping.*

2. *There exists an NPQ-restricted-mapping that cannot be expressed as a* $\mathcal{P}^{\text{nv,lv,re}}$-TO-$\mathcal{P}^{\text{nv,lv,re}}$*-mapping.*

The proof of this proposition follows almost immediately from the following result on expressibility of NPQs. The proof is in the appendix.

Lemma 6.5.5

1. *Every binary graph query* (ξ, x_1, x_2) *where the underlying graph of* ξ *is acyclic and connected is equivalent to an NPQ.*

2. *There is an NPQ that is not equivalent to any graph query.*

Using nesting to specify complex mappings. The first part of Proposition 6.5.4 tells us that by restricting to NPQs we are also dealing with acyclic CRPQs. The second part shows that, in addition, the use of NPQs in mappings allows us to express some properties that couldn't be expressed even if we used the full power of $\mathcal{P}^{\text{nv,lv,re}}$-TO-$\mathcal{P}^{\text{nv,lv,re}}$-mappings. Let us give an example of such a mapping

Recall the graph G_1 of Figure 6.1 in Section 6.1 over alphabet Σ_1 containing the information about authors, and their publications. We have seen that the query $Q_1'(x,y) = \exists z \exists w (z, \texttt{creator}, x) \wedge (z, \texttt{creator}, y)$ extracts from G_1 the information about coauthors of a paper. However, if one wants to extract only those researchers that are coauthors of a conference paper, then one needs to add more information to the queries that form the mapping. One possibility is to use query $Q_1''(x,y) = \exists z \exists w \exists p (z, \texttt{creator}, x) \wedge (z, \texttt{creator}, y) \wedge (z, \texttt{partOf}, w) \wedge (w, \texttt{series}, p)$.

But suppose now that one wishes to extract the *transitive closure* of the coauthor relation, restricted to conference papers only. We want to store this information into a target graph over alphabet $\Sigma_4 = \{\texttt{confConnected}\}$, such that two nodes A and B are connected via an edge

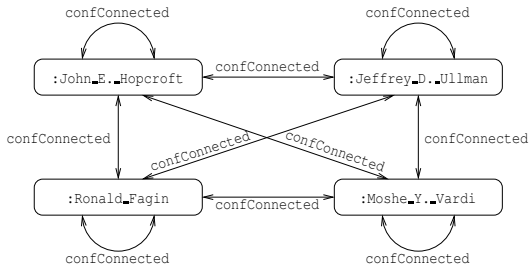


Figure 6.4: Graph G_4, a universal representative for the graph G_1 of Figure 6.1 under $\mathcal{M}_{14}$.

confConnected in this graph if A and B belong to the transitive closure of the coauthor relation. The following rule satisfies precisely this desiderata:

$$\mathcal{T}_{14} = \big(x, (\texttt{creator}^- \cdot [\texttt{partOf} \cdot \texttt{series}] \cdot \texttt{creator})^+, y\big) \rightarrow (x, \texttt{confConnected}, y).$$

Let $\mathcal{M}_{14} = (\Sigma_1, \Sigma_4, \mathcal{T}_{14})$. Then $\mathcal{M}_{14}$ is an NPQ-restricted mapping. A universal representative for G_1 under $\mathcal{M}_{14}$ is shown in Figure 6.4. Using the same tools involved in the proof of Proposition 6.5.4 one can show that this mapping is not equivalent to any $\mathcal{P}^{\text{nv,lv,re}}$-TO-$\mathcal{P}^{\text{nv,lv,re}}$-mapping.

6.6 Feasible Data Exchange Using NPQ-restricted Mappings

In the previous section we introduced NPQ-restricted mappings, and showed that they could express many interesting navigational properties in the context of graph data exchange. We now show their good properties with respect to the data exchange tasks we study in this dissertation.

6.6.1 Computing representatives in polynomial time

Since the combined complexity of evaluating NPQs over graphs is already in PTIME, we can use the same algorithm that we used in Section 6.4.1 to compute universal representatives in the general case, and immediately obtain tractability. The following theorem states the precise complexity of the problem:

Theorem 6.6.1 *There is a procedure that, given an NPQ-restricted mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ and a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$, computes a universal representative of $G_{\mathbf{S}}$ under $\mathcal{M}$ in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{M}|)$.*

Proof: It was shown in [Pérez et al., 2010] that given a NRE R and a graph G computing the set $[\![R]\!]_G$ can be done in time $O(|G|^2 \cdot |R|)$ (see Theorem 3.3 in [Pérez et al., 2010][3]).

[3]Actually, in [Pérez et al., 2010], the following stronger property was proved from which the bound $|G|^2 \cdot |R|$ follows directly. Given a node u, computing the set of all elements v such that $(u,v) \in [\![R]\!]_G$ can be done in time $O(|G| \cdot |R|)$ [Pérez et al., 2010].

From this fact it immediately follows that the procedure in Proposition 6.4.3 to compute a representative for a graph $G_{\mathbf{S}}$ under an NPQ-restricted mapping $\mathcal{M} = (G_{\mathbf{S}}, G_{\mathbf{T}}, \mathcal{T})$ runs in time $O(|G|^2 \cdot |\mathcal{M}|)$, by computing first $[\![R]\!]_{G_{\mathbf{S}}}$ for each expression in the left hand side of the rules of $\mathcal{T}$, and then proceed accordingly. □

Thus, by focusing on NPQ-restricted mappings, we reduce the complexity of universal representative computation from $|G_{\mathbf{S}}|^{O(|\mathcal{M}|)}$, implicit in Proposition 6.4.3, to quadratic in the size of the source graph database. One can also prove that the bound in Theorem 6.6.1 is tight.

Proposition 6.6.2 *There are families of NPQ-restricted mappings $\{\mathcal{M}_n = (\Sigma^n_{\mathbf{S}}, \Sigma^n_{\mathbf{T}}, \mathcal{T}_n)\}_{n\geq 1}$ and graph databases $\{G^m_{\mathbf{S}}\}_{m\geq 1}$, such that $|\mathcal{M}_n|$ is $O(n)$, $|G^m_{\mathbf{S}}|$ is $O(m)$, and every universal representative of $G^m_{\mathbf{S}}$ under $\mathcal{M}_n$ is of size $\Omega(m^2 \cdot n)$.*

Proof: For every $n \geq 1$ consider the alphabets $\Sigma^n_{\mathbf{S}} = \{a\}$, $\Sigma^n_{\mathbf{T}} = \{b_1, \ldots, b_n\}$, the mapping $\mathcal{M}_n = (\Sigma^n_{\mathbf{S}}, \Sigma^n_{\mathbf{T}}, \mathcal{T}^n)$, with $\mathcal{T}_n$ given by the rules

$$\begin{array}{rcl} (x, a^*, y) & \rightarrow & (x, b_1, y) \\ (x, a^*, y) & \rightarrow & (x, b_2, y) \\ & \vdots & \\ (x, a^*, y) & \rightarrow & (x, b_n, y) \end{array}$$

Notice that $|\mathcal{M}_n|$ is $O(n)$. Now, for every $m \geq 1$ consider the graph database $G^m_{\mathbf{S}}$ over $\Sigma^n_{\mathbf{S}}$ containing the edges

$$(1, a, 2), (2, a, 3), \ldots (m-1, a, m), (m, a, 1),$$

that is, $G^m_{\mathbf{S}}$ is a cycle of length m. Clearly $|G^m_{\mathbf{S}}|$ is $O(m)$. Now notice that for every $1 \leq k \leq n$ and for every pair (i, j) such that $1 \leq i, j \leq m$, we have that i and j are connected by a path of a-labelled edges, and thus they satisfy the query on the left hand side of each of the rules in $\mathcal{M}_n$. This implies that for every graph database $G_{\mathbf{T}}$ such that $G_{\mathbf{T}} \in \mathrm{Sol}_{\mathcal{M}_n}(G^m_{\mathbf{S}})$ each of the pairs (i, j) must satisfy all of the right hand side queries of $\mathcal{M}_n$. In other words, for every $1 \leq k \leq n$ and for every pair (i, j) such that $1 \leq i, j \leq m$ we have that $(i, b_k, j) \in G_{\mathbf{T}}$. Therefore, we have that $|G_{\mathbf{T}}| \geq m^2 \cdot n$. Finally, since all the values (i, j) such that $1 \leq i, j \leq m$ are actually values in the source graph $G^m_{\mathbf{S}}$, we have that every pattern that is a universal representative for $G^m_{\mathbf{S}}$ has at least $m^2 \cdot n$ edges. This completes the proof. □

6.6.2 Query answering

So far, we have achieved to identify an expressive class of mappings that has good properties in terms of computation of universal representatives. However, this universal representative might still be any arbitrary graph pattern. Since we know that querying arbitrary graph patterns

is computationally hard, we need a further restriction on this class of mappings in order to obtain a good class for query answering purposes.

In the same way, we cannot expect to have efficient query evaluation algorithms if we allow as query language the whole class of graph patterns, or even CRPQs, as a querying mechanism, since they are computationally hard in combined complexity. For this reason we focus solely on queries defined by NPQs.

NPQ-TO-$\mathcal{P}^{\text{nv}}$ **mappings**. The class of mappings we consider restrict the right hand side of dependencies to be graph patterns in $\mathcal{P}^{\text{nv}}$ i.e., conjunctive queries. In other words, it prohibits the use of regular expressions and label variables, which are the major contributors to complexity of query answering. More formally, a mapping $\mathcal{M} = (\Sigma_1, \Sigma_2, \mathcal{T})$ is an NPQ-TO-$\mathcal{P}^{\text{nv}}$ mapping if $\mathcal{T}$ is a finite set of rules of the form $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$, with $\varphi(\bar{x})$ an NPQ and $\psi(\bar{x})$ a graph query in $\mathcal{P}^{\text{nv}}$.

Our approach to compute certain answers for this class of mappings is as follows. Let $\mathcal{M}$ be a NPQ-TO-$\mathcal{P}^{\text{nv}}$-mapping, Q an NPQ and $G_{\mathbf{S}}$ a source graph for $\mathcal{M}$. Note that the universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$, by Corollary 6.4.4, is as well a pattern in $\mathcal{P}^{\text{nv}}$, and thus naive evaluation works for this pattern. One can then answer queries by first computing this universal representative, and then evaluate Q over the representative, in linear time, as was pointed out in [Pérez et al., 2010]. Summing up, we have:

Theorem 6.6.3 *Given an NPQ-*TO-$\mathcal{P}^{\text{nv}}$ *mapping $\mathcal{M}$ from $\Sigma_{\mathbf{S}}$ to $\Sigma_{\mathbf{T}}$, a source graph database $G_{\mathbf{S}}$ and an NPQ Q over $\Sigma_{\mathbf{T}}$,* DATA EXCHANGE CERTAIN ANSWERS *can be solved in time $O(|G_{\mathbf{S}}|^2 \cdot |\mathcal{M}| \cdot |Q|)$.*

GAV mappings. By further restricting the class of mappings, we can obtain even linear combined complexity, in the size of the database, for the problem of computing certain answers for NPQs.

This class of mappings corresponds to *NPQ*-TO-$\mathcal{P}^{\text{nv}}$ mappings of form $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$ in which all of its rules are of form $\varphi(x,y) \rightarrow (x,a,y)$, where a is just a letter of the target alphabet $\Sigma_{\mathbf{T}}$. This class of mappings defines target symbols in terms of NPQs views over the source, which resemble *global-as-view* (GAV) mappings as studied in relational databases [Lenzerini, 2002]. For these reason we call this mappings *NPQ-GAV mappings*. However limited in expressive power, these can still define many simple yet useful mappings. For instance, the first of the mappings presented in Section 6.1 is an NPQ-GAV-mapping.

Theorem 6.6.4 *Given an NPQ-GAV mapping $\mathcal{M}$ from $\Sigma_{\mathbf{S}}$ to $\Sigma_{\mathbf{T}}$, a graph database $G_{\mathbf{S}}$ over $\Sigma_{\mathbf{S}}$ and an NPQ Q over $\Sigma_{\mathbf{T}}$, the problem* DATA EXCHANGE CERTAIN ANSWERS *can be solved in time $O(|G_{\mathbf{S}}| \cdot |Q| \cdot |\mathcal{M}|)$.*

The lower bound proved in Proposition 6.6.2 also holds for NPQ-GAV mappings, so one cannot use universal representative computation in order to obtain Theorem 6.6.4. Instead, the proof of Theorem 6.6.4 is based on query *rewriting* techniques, as explained in the following technical result.

Lemma 6.6.5 *There is a procedure that, given an NPQ-GAV mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, and an NPQ Q over $\Sigma_{\mathbf{T}}$, computes an NPQ $rew_{\mathcal{M}}(Q)$ over $\Sigma_{\mathbf{S}}$, in time $O(|Q| \cdot |\mathcal{M}|)$, such that* CERTAIN-DE$_{\mathcal{M}}(Q, G_{\mathbf{S}}) = rew_{\mathcal{M}}(Q)(G_{\mathbf{S}})$ *for every source graph database $G_{\mathbf{S}}$.*

Proof: We describe an algorithm that receives an NPQ-GAV mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, with $\mathcal{T}$ of form

$$\mathcal{T} \quad = \quad \{(x, R_1, y) \to (x, a_1, y), \ldots, (x, R_n, y) \to (x, a_n, y)\},$$

where each R_i is an NRE and an NPQ Q over $\Sigma_{\mathbf{T}}$, and computes an NPQ $\text{rew}_{\mathcal{M}}(Q)$ over $\Sigma_{\mathbf{S}}$ that satisfies the conditions of the Lemma. We let $\text{rew}_{\mathcal{M}}(Q) = (x, \text{rew}_{\mathcal{M}}(R), y)$, where $\text{rew}_{\mathcal{M}}(R)$ is defined in an inductive fashion. For simplicity, and since $\mathcal{M}$ is clear from the context, we omit it from in the description of the algorithm, and speak of $\text{rew}(R)$ instead of $\text{rew}_{\mathcal{M}}(R)$.

Notice that if the language defined by R does not contain ε, then R can be written as an ε-free expression, and if it does contain ε, then R can be written in the form $\varepsilon + R'$ were R' is ε-free. Given this, it is safe to assume that R is either ε or it is ε-free. We assume this in the proof.

- If $R = \varepsilon$, then the rewriting $\text{rew}(R)$ corresponds to $[R_1 \mid \cdots \mid R_n] \mid [R_1^{-1} \mid \cdots + R_n^{-1}]$.
- If $R = a$ for $a \in \Sigma$, then assume that $R_{k_1}, \ldots, R_{k_m}$ are the expressions in the left hand side of the rules in $\mathcal{T}$ such that the right hand side of this rule corresponds to (x, a, y). Then $\text{rew}(R) = R_{k_1} \mid \cdots \mid R_{k_m}$.
- If $R = e_1 \mid e_2$, for e_1 and e_2 NRE's over $\Sigma_{\mathbf{T}}$, then $\text{rew}(R) = \text{rew}(e_1) \mid \text{rew}(e_2)$.
- If $R = e_1 \cdot e_2$, then $\text{rew}(R) = \text{rew}(e_1) \cdot \text{rew}(e_2)$.
- If $R = e_1^*$, then $\text{rew}(R) = \text{rew}(\varepsilon) \mid \text{rew}(e_1) \cdot (\text{rew}(e_1))^*$.
- If $R = e_1^-$, then $\text{rew}(R) = (\text{rew}(e_1))^{-1}$.
- If $R = [e_1]$, then $\text{rew}(R) = [\text{rew}(e_1)]$.

It is clear that the resulting expression is of size $|R| \times |\mathcal{M}|$. For the correctness of the rewriting, let $G_{\mathbf{T}}$ be the universal representative for $G_{\mathbf{S}}$ under $\mathcal{M}$. Note that this time $G_{\mathbf{T}}$ is indeed a graph database (that is, a pattern belonging to $\mathcal{P}$), and thus the notions of naive evaluation and usual evaluation coincide. Since we know from Section 4.2 that certain answers

for patterns in $\mathcal{P}^{\text{nv}}$ (and therefore in $\mathcal{P}$) can be computed by naive evaluation, it is enough to show the following claim. The proof is in the Appendix.

Claim 6.6.6 *Given an NPQ-GAV mapping $\mathcal{M} = (\Sigma_{\mathbf{S}}, \Sigma_{\mathbf{T}}, \mathcal{T})$, an NPQ Q over $\Sigma_{\mathbf{T}}$ and the universal representative $G_{\mathbf{T}}$ for $G_{\mathbf{S}}$ under $\mathcal{M}$, we have that $Q(G_{\mathbf{T}}) = rew_{\mathcal{M}}(Q)(G_{\mathbf{S}})$.*

This finishes the proof of the Lemma. □

Hence in order to evaluate the certain answers of an NPQ Q over $G_{\mathbf{S}}$ under the NPQ-GAV mapping $\mathcal{M}$, we can perform the following algorithm: Compute from $\mathcal{M}$ and Q the NPQ Q'. Then evaluate Q' over $G_{\mathbf{S}}$ in time $O(|G_{\mathbf{S}}| \cdot |Q'|)$ (as stated in Proposition 6.5.2), and thus in time $O(|G_{\mathbf{S}}| \cdot |Q| \cdot |\mathcal{M}|)$.

6.7 Composition of Mappings

We have seen that the class of NPQ-GAV mappings has remarkably good properties for query answering. In this section we take advantage of those properties, in particular the existence of rewritings over the source alphabets, and make a case for the usefulness of this language in a rather different scenario: when composing schema mappings. Composition has been identified as a fundamental process for several interoperability tasks [Melnik, 2004, Bernstein, 2003], and, as such, it has received considerable attention in relational and XML data exchange [Madhavan and Halevy, 2003, Melnik, 2004, Fagin et al., 2005c, Nash et al., 2005, Arenas et al., 2009, Amano et al., 2009]. On the other hand, the composition of schema mappings for graph databases has not yet been considered in the literature.

Given mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ is a new mapping that, intuitively, has the same effect as the application of $\mathcal{M}_1$ and $\mathcal{M}_2$ one after the other. Formally, given mappings $\mathcal{M}_1$ from Σ_1 to Σ_2, and $\mathcal{M}_2$ from Σ_2 to Σ_3, the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the mapping from Σ_1 to Σ_3 defined by $[\![\mathcal{M}_1]\!] \circ [\![\mathcal{M}_2]\!] = \{(G_1, G_3) \mid \text{there exists } G_2 \text{ over } \Sigma_2 \text{ such that } (G_1, G_2) \in [\![\mathcal{M}_1]\!] \text{ and } (G_2, G_3) \in [\![\mathcal{M}_2]\!]\}$ [Melnik, 2004, Fagin et al., 2005c].

Example 6.7.1 *Recall mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{14} = (\Sigma_1, \Sigma_4, \mathcal{T}_{14})$ from sections 6.1 and 6.5.3, respectively, given by rules*

$$\begin{aligned} \mathcal{T}_{12} &= (x, \texttt{creator}, y) \wedge (x, \texttt{partOf} \cdot \texttt{series}, w) \rightarrow (y, \texttt{makes}, x) \wedge (x, \texttt{inConf}, w) \\ \mathcal{T}_{14} &= \big(x, (\texttt{creator}^- \cdot [\texttt{partOf} \cdot \texttt{series}] \cdot \texttt{creator})^+, y\big) \rightarrow (x, \texttt{confConnected}, y) \end{aligned}$$

Let us show how to obtain the mapping $\mathcal{M}_{14}$ by means of $\mathcal{M}_{12}$. Indeed, consider mapping $\mathcal{M}_{24} = (\Sigma_2, \Sigma_4, \mathcal{T}_{24})$, with

$$\mathcal{T}_{24} = \big(x, (\texttt{makes} \cdot \texttt{makes}^-)^+, y\big) \rightarrow (x, \texttt{confConnected}, y).$$

Then, it is possible to show that $\mathcal{M}_{14}$ is equivalent to the composition $\mathcal{M}_{12} \circ \mathcal{M}_{24}$. As a proof of concept, consider again the graph database G_1 in Figure 6.1. The graph G_2 in Figure 6.2 is a universal representative for G_1 under $\mathcal{M}_{12}$. If one uses again G_2 to exchange information using mapping $\mathcal{M}_{24}$, one has that the graph G_4 in Figure 6.4 is a universal representative for G_2 under $\mathcal{M}_{24}$. But this graph is also a universal representative for G_1 under $\mathcal{M}_{14}$. Thus, successive applications of $\mathcal{M}_{12}$ and $\mathcal{M}_{24}$ yield the same result as a single application of $\mathcal{M}_{14}$.

One fundamental question in this context is definability of composition: given $\mathcal{M}_1$ and $\mathcal{M}_2$ defined in some mapping language, what is the language needed to specify the composition of both mappings? Of particular interest is the search for a mapping language $\mathcal{L}$ that is *closed under composition*. This means that for any two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ specified in $\mathcal{L}$, the composition $[\![\mathcal{M}_1]\!] \circ [\![\mathcal{M}_2]\!]$ can also be specified in $\mathcal{L}$ (i.e. there is a mapping $\mathcal{M}$ in $\mathcal{L}$ such that $[\![\mathcal{M}]\!] = [\![\mathcal{M}_1]\!] \circ [\![\mathcal{M}_2]\!]$). It has been shown that in the relational scenario the language of GAV mappings is closed under composition [Fagin et al., 2005c]. The next result shows that for NPQ-GAV mappings we obtain a similar good behavior. The proof is based on the rewriting properties of NPQ-GAV mappings that we stated in the previous section.

Theorem 6.7.2 *The language of NPQ-GAV mappings is closed under composition.*

Proof: Let $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$ be NPQ GAV-mappings. Next we define the set of rules $\mathcal{T}_{13}$ such that the mapping $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ specifies the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. To construct $\mathcal{T}_{13}$, replace each dependency in $\mathcal{T}_{23}$ of form $(x, R, y) \rightarrow (x, a, y)$ with $(x, \mathrm{rew}_{\mathcal{M}_{12}}(R), y) \rightarrow (x, a, y)$, where $\mathrm{rew}_{\mathcal{M}_{12}}(R)$ is the rewriting of R with respect to $\mathcal{M}_{12}$, as shown in Lemma 6.6.5. Notice that $\mathcal{M}_{13}$ is indeed an NPQ-GAV mapping. The following Lemma shows that the above mapping correctly defines the composition.

Lemma 6.7.3 *Given NPQ-GAV mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$, the mapping $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$, where $\mathcal{T}_{13}$ is as defined above, is such that $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$.*

Proof: We need to prove that $(G_1, G_3) \in [\![\mathcal{M}_{13}]\!]$ if and only if there exist a graph G_2 over Σ_2 such that $(G_1, G_2) \in [\![\mathcal{M}_{12}]\!]$ and $(G_2, G_3) \in [\![\mathcal{M}_{23}]\!]$.

($\Longrightarrow$): Assume that $(G_1, G_3) \in [\![\mathcal{M}_{13}]\!]$. Moreover, let G_2 be the universal representative for G_1 under $\mathcal{M}_{12}$. Notice that G_2 is a proper graph database over Σ_2, since $\mathcal{M}_{12}$ is an NPQ-GAV mapping. It is clear that $(G_1, G_2) \in [\![\mathcal{M}_{12}]\!]$, since G_2 is a universal representative. Next we show that $(G_2, G_3) \in [\![\mathcal{M}_{23}]\!]$. Let $(x, R, y) \rightarrow (x, a, y)$ be a rule in $\mathcal{T}_{23}$, and $(u_1, u_2) \in [\![R]\!]_{G_2}$, for a pair of nodes u_1, u_2 in G_2. We need to show that there is an edge in G_3 of form (u_1, a, u_2). But this follows from the construction of $\mathcal{T}_{13}$ and Lemma 6.6.5: In this case we know that $(u_1, u_2) \in [\![\mathrm{rew}_{\mathcal{M}_{12}}(R)]\!]_{G_1}$, and then since $(G_1, G_3) \in [\![\mathcal{M}_{13}]\!]$ and there is a rule of form $(x, \mathrm{rew}_{\mathcal{M}_{13}}(R), y) \rightarrow (x, a, y)$ in $\mathcal{T}_{13}$, it must be that (u_1, a, u_2) is an edge in G_3.

($\Longleftarrow$): Assume that there exist a graph G_2 over Σ_2 such that $(G_1, G_2) \in [\![\mathcal{M}_{12}]\!]$ and $(G_2, G_3) \in [\![\mathcal{M}_{23}]\!]$. We have to show that $(G_1, G_3) \in [\![\mathcal{M}_{13}]\!]$. Let $(x, \mathrm{rew}_{\mathcal{M}_{12}}(R), y) \to (x, a, y)$ be a rule in $\mathcal{T}_{13}$, and suppose that a pair (u_1, u_2) of nodes from G_1 belong to $[\![\mathrm{rew}_{\mathcal{M}_{12}}(R)]\!]_{G_1}$. We show that there is an edge (u_1, a, u_2) in G_3. By Lemma 6.6.5, we have that (u_1, u_2) belongs to the certain answers of R over G_1 under $\mathcal{M}_{12}$, and therefore $(u_1, u_2) \in [\![R]\!]_{G_2}$, since G_2 is a solution for G_1 under $\mathcal{M}$. But then since G_3 is a solution for G_2 under $\mathcal{M}_{23}$, and (by construction) $\mathcal{T}_{23}$ contains a rule of form $(x, R, y) \to (x, a, y)$, it must be the case that the edge (u_1, a, u_2) belongs to G_3. □ □

We finish this section with a remark about the possibility of obtaining similar closure results for other classes of mappings studied in this chapter. We show that this is not the case, when restricted to the GAV case, which is an advantage of NPQ-restricted mappings over other choices of binary relations. In what follows, we use the notion of $\mathcal{L}$-GAV mappings, where $\mathcal{L}$ is a class of queries, for the class of mappings specified by rules of the form $\varphi(x, y) \to (x, a, y)$, where $\varphi(x, y)$ is a binary query in $\mathcal{L}$ over the source and a is a symbol in the target.

We start with languages without conjunction, that is, RPQs. In order to show that the nesting operator of NPQs is the crucial feature needed for composition, we also consider the language of 2RPQs, the extension of RPQs with the inverse operator. The next proposition shows, in particular, that the nesting feature of NPQs is necessary to obtain the closure result in Theorem 6.7.2.

Proposition 6.7.4 *There exist RPQ-GAV mappings $\mathcal{M}_{12}$ from Σ_1 to Σ_2 and $\mathcal{M}_{23}$ from Σ_2 to Σ_3, such that the mapping $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is not equivalent to a 2RPQ-GAV mapping.*

Proof: We need the following definition. Let $G = (N, E)$ be a graph over Σ. A semipath in G is a sequence $u_1, a_1, u_2, a_2, \ldots, u_m, a_m, u_{m+1},$, where each u_i belongs to N, each a_i belongs to $\Sigma \cup \{a^- \mid a \in \Sigma\}$, and for each u_i, a_i, u_{i+1}, we have that (u_i, a_i, u_{i+1}) belongs to E, if a_i is not an inverse symbol, and (u_{i+1}, a_i, u_i) belongs to E if a_i is an inverse symbol, i.e., of form a^- for some $a \in \Sigma$.

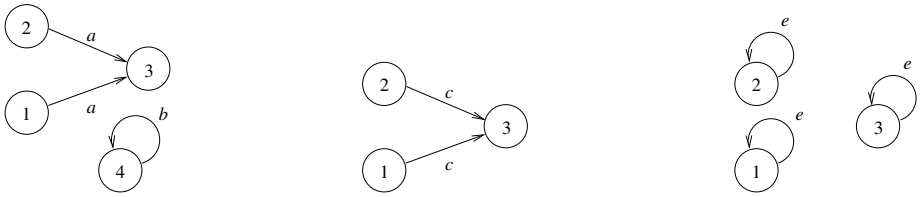
Let $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{c, d\}$, $\Sigma_3 = \{e\}$, and consider RPQ-mappings $\mathcal{M}_{12} = (\Sigma_1, \Sigma_2, \mathcal{T}_{12})$ and $\mathcal{M}_{23} = (\Sigma_2, \Sigma_3, \mathcal{T}_{23})$, with

$$\begin{aligned} \mathcal{T}_{12} &= \{(x, a, y) \to (x, c, y)\}, \\ \mathcal{T}_{23} &= \{(x, d^*, y) \to (x, e, y)\}. \end{aligned}$$

We next show that the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ cannot be specified by 2RPQ-GAV mapping. On the contrary, assume that $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ is a mapping such that $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!] = [\![\mathcal{M}_{13}]\!]$. Since $\mathcal{M}_{13}$ is 2RPQ-GAV, we have that $\mathcal{T}_{13}$ is of the following form:

$$\mathcal{T}_{13} = \{(x, r_1, y) \to (x, e, y), \ldots, (x, r_k, y) \to (x, e, y)\},$$

with r_i a 2RPQ over Σ_1 for every $i \in \{1,\ldots,k\}$. Consider now the graphs G_1, G_2 and G_3 over Σ_1, Σ_2 and Σ_3, respectively, given by:



G_1 G_2 G_3

It is not difficult to see that $(G_1,G_2) \in [\![\mathcal{M}_{12}]\!]$, and that $(G_2,G_3) \in [\![\mathcal{M}_{23}]\!]$. Thus, we also have that $(G_1,G_3) \in [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$. Moreover it is not difficult to show that for every graph $G \in \mathrm{Sol}_{\mathcal{M}_{12}}(G_1)$ it holds that $G_2 \subseteq G$, and that for every $G \in \mathrm{Sol}_{\mathcal{M}_{23}}(G_2)$ it holds that $G_3 \subseteq G$. From this and since we are assuming that $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!] = [\![\mathcal{M}_{13}]\!]$, we obtain that $G_3 \in \mathrm{Sol}_{\mathcal{M}_{13}}(G_1)$ and that for every $G \in \mathrm{Sol}_{\mathcal{M}_{13}}(G_1)$ it holds that $G_3 \subseteq G$.

Now consider the set $\mathcal{T}_{13}$. We next show that there exists a dependency $(x,r,y) \rightarrow (x,e,y)$ in $\mathcal{T}_{13}$ such that $(1,1) \in [\![r]\!]_{G_1}$. To obtain a contradiction, assume that it is not the case. Then we have that $(G_1, G_3 \setminus \{(1,e,1)\}) \models \mathcal{T}_{13}$ given that $G_1 \not\models (1,r,1)$ for every r that appears in the left-hand side of a dependency in $\mathcal{T}_{13}$. This is a contradiction since we know that for every $G \in \mathrm{Sol}_{\mathcal{M}_{13}}(G_1)$ it holds that $G_3 \subseteq G$, and $G_3 \setminus \{(1,e,1)\} \not\subseteq G_3$. Then consider the dependency $(x,r,y) \rightarrow (x,e,y)$ in $\mathcal{T}_{13}$ such that $(1,1) \in [\![r]\!]_{G_1}$. We consider two cases:

- First, if the language defined by r contains the empty path (ε) then we have that $G_1 \models (4,r,4)$ and thus $G_3 \models (4,e,4)$ which is a contradiction.
- Second, assume that every semi path defined by r has at least one edge. Then every path from 1 to 1 that conforms to r should visit node 3. Moreover, notice that every sequence of edges that defines a (semi) path in G_1 from 3 to 1 also defines a (semi) path from 3 to 2 (this is only because of the symmetry of G_1). Thus, given that $(1,1) \in [\![r]\!]_{G_1}$ and every semi path defined by r has at least one edge, we obtain that $(1,2) \in [\![r]\!]_{G_1}$, but this is a contradiction since $G_3 \not\models (1,e,2)$.

In both cases we obtain a contradiction, such 2RPQ mapping $\mathcal{M}_{13}$ cannot exist. □

One may also ask whether the use of conjunctions over 2RPQs or RPQs can lead to a closed mapping language. The following result shows that this is not the case.

Proposition 6.7.5 *There are CRPQ-GAV mappings $\mathcal{M}_{12}$ from Σ_1 to Σ_2 and $\mathcal{M}_{23}$ from Σ_2 to Σ_3, such that the mapping $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is not equivalent to any $\mathcal{P}^{\mathrm{nv,lv,re}}$-TO-$\mathcal{P}^{\mathrm{nv,lv,re}}$-mapping.*

Proof: Let $\Sigma_1 = \{a,b,c,d\}$, $\Sigma_2 = \{e,f\}$ and $\Sigma_3 = \{g\}$, and consider the mappings $\mathcal{M}_{12} = (\Sigma_1,\Sigma_2,\mathcal{T}_{12})$, and $\mathcal{M}_{23} = (\Sigma_2,\Sigma_3,\mathcal{T}_{23})$, where

$$\begin{aligned} \mathcal{T}_{12} &= \{\exists z\big((x,a,y) \wedge (y,b,z) \wedge (z,c,x)\big) \rightarrow (x,e,y),\ (x,d,x) \rightarrow (x,f,x)\} \\ \mathcal{T}_{23} &= \{(x,f,x) \wedge (x,e^+,y) \wedge (y,f,y) \rightarrow (x,g,y)\} \end{aligned}$$

We next show that the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ cannot be specified by a graph mapping. Assume that there is a mapping $\mathcal{M}_{13}$ such that $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$. We now show that there is a graph G_1 over Σ_1 such that $(G_1, \emptyset) \notin [\![\mathcal{M}_{13}]\!]$ but $(G_1, \emptyset) \in [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$.

Consider then an arbitrary mapping $\mathcal{M}_{13} = (\Sigma_1, \Sigma_3, \mathcal{T}_{13})$ such that $[\![\mathcal{M}_{13}]\!] \subseteq [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$. We assume that $\mathcal{T}_{13}$ is of the form:

$$\mathcal{T}_{13} \quad = \quad \{\varphi_1(\bar{x}_1) \to \psi_1(\bar{x}_1), \ldots, \varphi_k(\bar{x}_k) \to \psi_k(\bar{x}_k)\},$$

with each φ_i and ψ_i are arbitrary graph queries over Σ_1 and Σ_3, respectively, for every $i \in \{1, \ldots, k\}$.

Let $p = |\mathcal{M}| + 2$. We then define graph G_1 as follows: it consists of elements $\{u_1, v_1, u_2, v_2, \ldots, u_{p-1}, v_{p-1}, u_p\}$, and contains edges $(u_i, a, u_{i+1}), (u_{i+1}, b, v_i), (v_i, c, u_i)$ for each $1 \leq i \leq p$, plus edges (u_1, d, u_1) and (u_p, d, u_p). In other words, it is a sequence of triangles, all of them concatenated together, with a self loop in the beginning and in the end. Now consider graph G_3, given by the single edge (u_1, g, u_p). It is clear that (G_1, G_3) belong to $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$, and any graph G such that $(G_1, G) \in [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$ is a superset of G_3. Since $[\![\mathcal{M}_{13}]\!] = [\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$, we have in particular that $(G_1, G_3) \models \mathcal{M}_{13}$. All of this means that G_3 is a universal representative for G_1 under $\mathcal{M}_{13}$, and therefore any representative must contain the edge (u_1, g, u_p). Then, there has to be a rule in $\mathcal{T}_{13}$, say $\varphi_j(\bar{x}_j) \to \psi_j(\bar{x}_j)$, that produces the edge (u_1, g, u_p) in G_3, when computing the universal representative according to the algorithm in the proof of Proposition 6.4.3. Then there are nodes $u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}}$ that witness the satisfaction of $\varphi_j(\bar{x}_j)$ over G_1, and with h the corresponding homomorphism from $\bar{x}_j$ to $u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}}$.

Note that $|\bar{x}_j|$ is strictly smaller than p. Assume that $\varphi_j(\bar{x}_j) = (\xi_{\varphi_j}, \bar{x}_j)$, where ξ_{φ_j} is a graph pattern, and let G_1' be a graph database over Σ_1 that is canonical for G_1. By construction $G_1' \models \varphi_j(u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}})$. But notice however that G_1 can only contain a limited number of nodes with more than 1 incoming edge, namely nodes $u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}}$. The rest of the nodes of G_1' are added as a result of the canonical construction, and are therefore only connected to the previous and next node of a path.

There are now two possibilities:

- There is no path from u_1 and u_p in G_1' consisting only of concatenated triangles. This shows that $(G_1', \emptyset)$ belongs to $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$. This is a contradiction, since $(u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}})$ belongs to $[\![\varphi_j(\bar{x}_j)]\!]_{G_1'}$, the chase for G_1' over $\mathcal{M}_{13}$ as explained in the proof of Proposition 6.4.3 is not empty, and thus by definition $(G_1', \emptyset)$ cannot belong to $[\![\mathcal{M}_{12}]\!] \circ [\![\mathcal{M}_{23}]\!]$.

- There is a path from u_1 and u_p in G_1' that are only concatenated triangles. Then all of the nodes of this path belong to $u_{m_1}, \ldots, u_{m_{|\bar{x}_j|}}$. From this fact one can conclude that the universal representative for G_1 under $\mathcal{M}_{13}$ is strictly a superset of G_3, which is also a contradiction.

□

As a final remark we note that the restriction to GAV mappings in general is crucial for obtaining the closure result in Theorem 6.7.2. In fact, it is not difficult to adapt results by Fagin et al. [Fagin et al., 2005c] to show that our graph mappings cannot express even the composition of two NPQ-TO-$\mathcal{P}^{\mathrm{nv}}$-mappings.

Chapter 7

Applications in Formal Language Theory

Just as graph databases can be viewed as finite automata, graph patterns in turn can be viewed as *incomplete automata*, in which the precise information about the transition relation is lost. In this chapter we formally define this model of automata, and study some of their basic properties. We define two notions of acceptance for them, and show how they give rise to a much intricate model, with applications not only in graph databases, but also in other fields such as program analysis and constraint satisfaction.

7.1 Incomplete Automata

An incomplete automaton is 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. Recall that $\text{REG}(\Sigma)$ denotes the language of all regular expressions that can be constructed from the symbols of an alphabet Σ. We then have the following definition:

Definition 7.1.1 (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 $\delta \subseteq Q \times \text{REG}(\Sigma \cup \mathcal{W}) \times Q$.*

Example 7.1.2 *Figure 7.1 depicts two incomplete automata. The automaton $\mathcal{A}_1$ on the left uses alphabet $\{0,1\}$, variables X and Y in its transitions and the regular expression XY. Automaton $\mathcal{A}_2$ over alphabet $\{a,b\}$ uses only regular expressions in some of its transitions (the expression a^+), but no label variables. As usual, initial states are pointed by an arrow, and final states are depicted with double lines.*

Figure 7.1: Incomplete automata $\mathcal{A}_1$ and $\mathcal{A}_2$

7.1.1 Semantics

There are two natural notions of acceptance that can be defined for incomplete automata. To state them formally, 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 $\theta : (Q \times \mathrm{REG}(\Sigma \cup \mathcal{W}) \times Q) \to (Q \times \Sigma^* \times Q)$ 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 introduce our two semantics for incomplete automata. The first one is inspired in the notion of certain answers, and we call it *certainty semantics*. Formally, let $\mathcal{A}$ be an incomplete automaton. We define the language of $\mathcal{A}$ under certainty semantics, that we denote by $\mathcal{L}_\Box(\mathcal{A})$, as

$$\mathcal{L}_\Box(\mathcal{A}) := \bigcap \{ \mathcal{L}(\nu(\mathcal{A})) \mid \nu \text{ is a valuation for } \mathcal{A} \}.$$

As expected, the study of incomplete automata under certainty semantics has direct applications in graph databases, specifically when querying graph patterns; and indeed we have already proved some results for these automata when studying the complexity of query answering in Chapter 4. We explain these applications in Sections 7.2.1 and 7.2.2.

Our second semantics is based on the unions of the languages of the valuations of the automata, instead of the intersection, and for this reason we call it *possibility semantics*. Formally, the language of $\mathcal{A}$ under possibility semantics, that we denote by $\mathcal{L}_\Diamond(\mathcal{A})$, is the set

$$\mathcal{L}_\Box(\mathcal{A}) := \bigcup \{ \mathcal{L}(\nu(\mathcal{A})) \mid \nu \text{ is a valuation for } \mathcal{A} \}.$$

Example 7.1.3 *Consider again incomplete automaton $\mathcal{A}_1$ from figure 7.1. The language $\mathcal{L}_\Box(\mathcal{A}_1)$ generated by the certainty semantics on $\mathcal{A}_1$ is simply the language $\{1\}$ (containing the single word 1), as it corresponds to the intersection of the languages given by expressions $(00)^*1(00)^*$, $(00)^*1(01)^*$, $(01)^*1(10)^*$ and $(01)^*1(11)^*$. The language $\mathcal{L}_\Diamond(\mathcal{A}_1)$ generated by the possibility semantics on $\mathcal{A}_1$ corresponds to the union of the languages of the aforementioned expressions. In other words, it is given by the expression $(00)^*1(00)^* \mid (00)^*1(01)^* \mid (01)^*1(10)^* \mid (01)^*1(11)^*$.*

The possibility semantics has been already studied in some cases (see e.g. [Grumberg et al., 2010, Kaminski and Zeitlin, 2010, Freydenberger, 2011]), and has several applications, particularly in program analysis. We shall also detail these in the next section, but before let us introduce a more restricted model of automata that will take a very important role in this Chapter.

7.1.2 Parameterized automata and parameterized regular expressions

If we only allow incomplete automata to have transitions labelled by symbols in $\Sigma \cup \mathcal{W}$ (instead of arbitrary regular expressions), then we arrive at a restricted notion of incomplete automata that has been already studied in formal language theory, albeit mostly in the context of words of infinite length. We denote these as *parameterized automata*, since the variables in the transitions can be seen as *parameters* of the automata. Formally, a *parameterized automaton* 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 now $\delta \subseteq Q \times (\Sigma \cup \mathcal{W}) \times Q$. In other words, these automata are just patterns in $\mathcal{P}^{\text{lv}}$, with distinguished nodes for the initial and final states.

For these automata we have a corresponding notion of regular expressions, that we call *parameterized regular expressions*, which are simply defined as regular expressions built from alphabet $\Sigma \cup \mathcal{W}$. For example, $(Xab)^*$ and $(0X)^*1(XY)^*$ are parameterized regular expressions.

Valuations for parameterized automata (or expressions) are just mappings $\nu : \mathcal{W} \to \Sigma$, so we can define certainty semantics and possibility semantics in the same way as for incomplete automata. More precisely, let e be a parameterized regular expression over $\Sigma \cup \mathcal{W}$. Then $\mathcal{L}_\Box(e) := \bigcap\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$ and $\mathcal{L}_\Diamond(e) := \bigcup\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$

Equivalence between parameterized automata and expressions. Since the transitions for parameterized automata are now subsets of $Q \times (\Sigma \cup \mathcal{W}) \times Q$, we can view them as NFAs over alphabet $\Sigma \cup \mathcal{W}$, and therefore they also define regular languages over this alphabet. In the same way, parameterized regular expressions also define regular languages over $\Sigma \cup \mathcal{W}$. Thus, given a parameterized expression e, we can perform the usual expression to automata translation (See e.g. [Hagenah and Muscholl, 1998]), and obtain an incomplete automaton $\mathcal{A}_e$ that accepts the same language (over $\Sigma \cup \mathcal{W}$) as the expression e). It is immediate from the definition that $\mathcal{A}_e$ and e also define the same languages under both possibility and certainty semantics, now of course under alphabet Σ. The converse is also true, i.e., one can go from parameterized automata to parameterized regular expressions. We thus obtain:

Lemma 7.1.4 *For every parameterized regular expression e one can construct, in* PTIME*, a parameterized automata $\mathcal{A}_e$ such that $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(\mathcal{A}_e)$ and $\mathcal{L}_\Diamond(e) = \mathcal{L}_\Diamond(\mathcal{A}_e)$. For every parameterized automata $\mathcal{A}$ one can construct in* EXPTIME *a parameterized regular expression $e_\mathcal{A}$ such that $\mathcal{L}_\Box(\mathcal{A}) = \mathcal{L}_\Box(e_\mathcal{A})$ and $\mathcal{L}_\Diamond(\mathcal{A}) = \mathcal{L}_\Diamond(e_\mathcal{A})$*

This lemma states that, as usual, we can easily go from expressions to automata, while the converse translation may incur in an exponential blowup. For this reason when we study decision problems associated with these models we usually state the lower bounds for regular expressions, while the upper bounds are stated in terms of automata. We shall see that, despite the asymmetry mentioned above, for most of the problems studied in this dissertation the assumption of regular expressions or automata does not make any substantial difference in the computational complexity associated with these tasks.

Example 7.1.5 *Consider again automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from figure 7.1. We have that $\mathcal{A}_1$ is a parameterized automaton, and a parameterized regular expression equivalent to $\mathcal{A}_1$ is $e_1 = (0X)^*1(XY)^*$, i.e. we have that $\mathcal{L}_\Diamond(\mathcal{A}_1) = \mathcal{L}_\Diamond(e_1)$ and $\mathcal{L}_\Box(\mathcal{A}_1) = \mathcal{L}_\Box(e_1)$. On the other hand, $\mathcal{A}_2$ is not a parameterized automaton. In fact, we shall later see that the language generated by $\mathcal{A}_2$ under the possibility semantics is not even a regular language.*

7.2 Applications of Incomplete Automata

Incomplete automata and parameterized regular expressions arise in a variety of applications, in particular in the fields of querying graph-structured data, and static analysis of programs. We now explain these connections.

7.2.1 Certain answers over patterns

While not completely equivalent to the certainty or possibility semantics for incomplete automata, the task of querying graph patterns is closely related with computational tasks for incomplete automata. Given a graph pattern $\pi = (N,E) \in \mathcal{P}^{\mathrm{nv,lv,re}}$ over Σ that uses label variables $\mathcal{W}$, and two nodes n_1, n_2 from $\mathbf{V} \cap N$ (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.

Proposition 7.2.1 *Let $\varphi(x,y) = (x,L,y)$ be an RPQ, $\pi = (N,E)$ a graph pattern, and n_1,n_2 two of its nodes from $\mathbf{V}$. Then $(n_1,n_2) \in \mathrm{CERTAIN}(Q,\pi)$ if and only if $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$ for every valuation ν.*

Proof: Let $Q(x,y) = (x,L,y)$ be an RPQ, $\pi = (N,E)$ be a graph pattern, with $\mathcal{W}$ the set of label variables mentioned in π; and n_1,n_2 two of its nodes from $\mathbf{V}$. We prove that $(n_1,n_2) \in \mathrm{CERTAIN}(Q,\pi)$ if and only if $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$, for every valuation ν.

($\Rightarrow$): Assume that $(n_1,n_2) \in \mathrm{CERTAIN}(Q,\pi)$. 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 $\theta : (N \times \mathrm{REG}(\Sigma \cup \mathcal{W}) \times N) \to (N \times \Sigma^* \times N)$ 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 $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$, which suffices for the proof since ν is arbitrarily chosen.

Let σ be the assignment from the nodes of π into $\mathbf{V}$ 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 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 $(n_1,n_2) \in \text{CERTAIN}(Q,\pi)$, it is the case that $(n_1,n_2) \in Q(G)$. Thus, there is a path ρ in G from n_1 to n_2 such that $\lambda(\rho) \in L$. It is now easy to show that there is a run of $\nu(\mathcal{A}_\pi(n_1,n_2))$ that accepts a word in L (namely, the word $\lambda(\rho)$). 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,w',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 w'.

($\Leftarrow$): Assume that $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$ for every valuation ν. We prove that $(n_1,n_2) \in \text{CERTAIN}(Q,\pi)$. From Claim 4.3.3 we only need to show that for every canonical graph database G for π, it is the case that $(n_1,n_2) \in Q(G)$. This is what we do next.

Let G be an arbitrary graph database $G \in [\![\pi]\!]$, and assume that G is σ-canonical for π via assignment $\eta : \mathcal{W} \to \Sigma$. (Recall that σ is an assignment from the nodes of π into $\mathbf{V}$ that is the identity on node ids and maps each node variable x into a different node id n_x). Clearly, both node ids n_1 and n_2 belong to G. For each edge e in π, let us denote by ρ_e the path of G that is associated with e.

Let us define now a mapping $\theta : (N \times \text{REG}(\Sigma \cup \mathcal{W}) \times N) \to (N \times \Sigma^* \times N)$ that assigns to each edge $e = (p,r,q) \in E$ a transition (p,w,q), where w is the word $\lambda(\rho_e)$. This is clearly well-defined, since, by definition, $\lambda(\rho_e)$ satisfies $\eta(r)$.

Since $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$ for every valuation, it is the case that $L \cap L(\nu(\mathcal{A}_\pi(n_1,n_2))) \neq \emptyset$. Thus, there is a word $w \in L$ that is accepted by $\nu(\mathcal{A}_\pi(n_1,n_2))$. It is not hard to prove then that there is a 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 precisely the paths of G that are associated with the edges of π; that is, if (p,w',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 w'.

This implies that $(n_1,n_2) \in Q(G)$. Since G is an arbitrary graph database that is canonical for π, we conclude that $(n_1,n_2) \in \text{CERTAIN}(Q,\pi)$ □

Thus, for instance, if L defines a single word $w \in \Sigma^*$, then $(n_1,n_2) \in \text{CERTAIN}(Q,\pi)$ if and only if w belongs to $\mathcal{L}_\square(\mathcal{A}_\pi(n_1,n_2))$. This already motivates the need for studying the membership problem for incomplete automaton over the certainty semantics, as it is tightly related to the task of querying RPQs defined by words over graph patterns. Note that the

results in chapter 5 tells us that query answering is non-trivial even for this class of queries.

There are other much more intricate connections between incomplete automata and querying graph patterns. We now present a different, novel scenario where the notion of certainty semantics is of crucial importance: certain answers for queries that can output paths.

7.2.2 Certain answers for queries returning 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

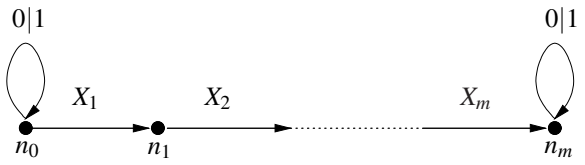
$$Ans(\bar{z}, \rho) \leftarrow (x, \rho : R, y)$$

where, on top of the usual RPQ $Q(\bar{z}) = (x, R, y)$, one is allowed to name the path ρ witnessing the query, and to output its label. Of course the number of R-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 R-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 above, and a graph pattern π. Let n_1, n_2 be two nodes from $\mathbf{V}$ 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 a path between n_1 and n_2 with label ρ, and ρ belongs to the language of R. 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 7.2.2 *For $m > 0$, consider the pattern π_m over $\Sigma = \{0, 1\}$ shown in the figure below.*



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 non-circular 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).

Note that, if we see the above pattern π as an incomplete automata $\mathcal{A}_\pi(n_0, n_m)$, with n_0 and n_m the initial and final nodes, when computing the set of words in word in $\text{CERTAIN}_\Sigma^{\text{path}}(Q;\pi,n_0,n_m)$ we are really asking for the language of $\mathcal{A}_\pi(n_0,n_m)$ under certainty semantics, or $\mathcal{L}_\square(\mathcal{A}_\pi(n_0,n_m))$. This is formalized in the following Proposition:

Proposition 7.2.3 *Let $Ans(x,y,\rho) \leftarrow (x,\rho : L, y)$ be an RPQ with path output, $\pi = (N,E)$ a graph pattern, and n_1, n_2 two of its nodes from $\mathbf{V}$. Then $w \in \text{CERTAIN}^{\text{path}}(Q;\pi,n_1,n_2)$ if and only if $w \in L$ and w belongs to $\mathcal{L}_\square(\mathcal{A}_\pi(n_1,n_2))$.*

Proof: ($\Rightarrow$): 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 belongs to $\mathcal{L}_\square(\mathcal{A}_\pi(n_1,n_2))$. 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 $\mathbf{V}$ 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 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 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,w',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 w'.

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

($\Leftarrow$): Assume that $w \in L$ and that w belongs to $\mathcal{L}_\square(\mathcal{A}_\pi(n_1,n_2))$. 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 belongs to $\mathcal{L}_\square(\mathcal{A}_\pi(n_1,n_2))$, 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, w', 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 w'. 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. □

In other words, the above proposition states that the set $\text{CERTAIN}^{\text{path}}(Q; \pi, n_1, n_2)$ of words corresponds to $\mathcal{L}_\Box(\mathcal{A}_\pi(n_1, n_2)) \cap L$. This simple connection further motivates the need to study certainty semantics for incomplete automata, as it raises several questions about it. For example, is the set $\mathcal{L}_\Box(\mathcal{A}_\pi(n_1, n_2))$ regular? if so, can we return the set of words $\mathcal{L}_\Box(\mathcal{A}_\pi(n_1, n_2)) \cap L$ as an NFA? And, further, is it possible to decide if there are any certain paths at all in a pattern π, i.e. is the set $\mathcal{L}_\Box(\mathcal{A}_\pi(n_1, n_2))$ nonempty? What is the complexity of checking if a word belongs to $\mathcal{L}_\Box(\mathcal{A}_\pi(n_1, n_2))$? We shall answer this questions shortly, but let us first motivate the study of possibility semantics for incomplete automata.

7.2.3 Applications in program analysis

Finally, we analyze the need for the study of incomplete automata under maybe semantics. Here we focus not on incomplete automata, but rather on parameterized regular expressions. That regular expressions with variables appear naturally in program analysis tasks was noticed, for instance, in [Liu et al., 2004, Liu and Stoller, 2006, de Moor et al., 2003]. One uses the alphabet that consists of symbols related to operations on variables, pointers, or files, e.g., `def` for defining a variable, `use` for using it, `open` for opening a file, or `malloc` for allocating a pointer. A variable then follows: $\texttt{def}(x)$ means defining variable x. While variables and alphabet symbols do not mix freely any more, it is easy to enforce correct syntax with an automaton. An example of a regular condition with parameters is searching for uninitialized variables: $(\neg\texttt{def}(x))^*\texttt{use}(x)$.

Expressions like this are evaluated on a graph that serves as an abstraction of a program. One considers two evaluation problems: whether under some evaluation of variables, either some path, or every path between two nodes satisfies it. This amounts to computing $\mathcal{L}_\Diamond(e)$ and checking whether all paths, or some path between nodes is in that language. In case of uninitialized variables one would be using 'some path' semantics; the need for the 'all paths' semantics arises when one analyzes locking disciplines or constant folding optimizations [Liu et al., 2004, de Moor et al., 2003]. So in this case the language of interest for us is $\mathcal{L}_\Diamond(e)$, as one wants to check whether there is an evaluation of variables for which some property of a program is true.

Parameterized regular expressions appear in other applications as well, e.g., in phase-sequence prediction for dynamic memory allocation [Shen et al., 2007], or as a compact way to express a family of legal behaviors in hardware verification [Bhadra et al., 2005], or as a tool to state regular constraints in constraint satisfaction problems [Pesant, 2004]. In all of these

applications we deal with problems similar to the ones stated for certainty semantics, such as regularity, nonemptiness, membership, to name a few. To answer these problems we devote the remainder of this chapter, and the next one.

7.3 Properties of Languages Generated by Incomplete Automata

Our first task in the study of these languages is to compare them in terms of their regularity. While not immediately obvious from the definition, we can show that language generated by certainty semantics for incomplete automata are regular. The intuition behind this fact is based on the idea that one can dismiss all transitions in incomplete automaton that are labeled by expressions that define infinite languages.

On the other hand, or possibility semantics the languages need not be regular. However, we show regularity for parameterized automata, the fragment of incomplete automata that disallows the use of regular expressions in the transitions.

Whenever the languages are regular, we show how to compute NFA's that define these languages. These can be of size doubly exponential for certainty semantics, or single exponential for parameterized automata under possibility semantics. We give precise algorithms to construct these NFAs, and finish this section by proving the optimality of these algorithms.

Along the section we make use of the following technical but self evident claim.

Claim 7.3.1 *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 $\eta : \mathcal{W} \to \Sigma$.*

7.3.1 Certainty semantics

We start by proving that all languages generated by certainty semantics are regular. The proof is based in the idea that all transitions not defining a finite language can be dismissed when dealing with certainty semantics. More precisely, let $\mathcal{A} = (Q, \Sigma, \mathcal{W}, q_0, F, \delta)$ be an incomplete automaton, and 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 $(Q, \Sigma, \mathcal{W}, q_0, F, \delta^{fin})$. The following lemma formalizes the idea presented above:

Lemma 7.3.2

$\mathcal{L}_{\Box}(\mathcal{A}) = \mathcal{L}_{\Box}(\mathcal{A}^{fin})$

Proof: The fact that $\mathcal{L}_{\Box}(\mathcal{A}^{fin}) \subseteq \mathcal{L}_{\Box}(\mathcal{A})$ is straightforward. Thus, we only need to show that $\mathcal{L}_{\Box}(\mathcal{A}) \subseteq \mathcal{L}_{\Box}(\mathcal{A}^{fin})$. Assume then that a word w belongs to $\mathcal{L}_{\Box}(\mathcal{A})$. To prove that w belongs to $\mathcal{L}_{\Box}(\mathcal{A}^{fin})$ 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 7.3.1, $\eta(L)$ is also an infinite language).

Since $w \in \mathcal{L}_{\Box}(\mathcal{A})$ 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 $w \in \mathcal{L}_{\Box}(\mathcal{A}^{fin})$, which finishes the proof of the Lemma. □

We now have all the ingredients to show regularity for certainty semantics.

Proposition 7.3.3 *For an incomplete automaton $\mathcal{A}$, the language $\mathcal{L}_{\Box}(\mathcal{A})$ is regular. An NFA accepting $\mathcal{L}_{\Box}(\mathcal{A})$ can be constructed in doubly exponential time.*

Proof: First, we prove that $\mathcal{L}_{\Box}(\mathcal{A})$ is regular. By Lemma 7.3.2 we know that $\mathcal{L}_{\Box}(\mathcal{A})$ 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 7.3.1, 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 $L(\mathcal{B}) = \mathcal{L}_{\Box}(\mathcal{A})$. We have argued in the previous paragraph that $\mathcal{L}_{\Box}(\mathcal{A})$ 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 7.3.4 *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 $|r|$ (that is, they have at most $|r|$ symbols). Furthermore, $L(r)$ contains at most $O(|\Sigma|^{|r|})$ words.*

From Claims 7.3.1 and 7.3.4 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 7.3.1 we have that $\eta(L)$ always defines a finite language, and thus by Claim 7.3.4 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. This shows that $\mathcal{B}$ can be constructed in double exponential time. □

We later show that the algorithm for computing NFA's presented in this section is optimal. First, let us examine the possibility semantics.

7.3.2 Possibility semantics

In contrast with certainty semantics, we easily loose regularity when we deal with possibility semantics.

Example 7.3.5 *Consider again the incomplete automaton $\mathcal{A}_2$ depicted in Figure 7.1 in Section 7.1. It contains states p_0 and p_1, and transitions (p_0,a^+,p_1) and (p_1,b,p_0). Since it does not use variables in the transitions (only regular expressions) the valuations for this automaton are just assignments that map the transition (p_0,a^+,p_1) to transitions of form (p_0,w,p_1), with w a word in a^+. From this observation we conclude that the language $\mathcal{L}_\diamond(\mathcal{A}_2)$ generated by possibility semantics corresponds to $\{(a^ib)^* \mid i \geq 1\}$, which is clearly not a regular language, and in fact not even context-free.*

Example 7.3.5 suggest that the regularity is lost not because of the possibility of using variables in incomplete automata, but because of the regular expressions in the transitions. Indeed, next we show that that the language defined by parameterized automata over the possibility semantics is regular.

Proposition 7.3.6 *For a parameterized automaton $\mathcal{A}$, the language $\mathcal{L}_\diamond(\mathcal{A})$ is regular. An NFA accepting $\mathcal{L}_\diamond(\mathcal{A})$ can be constructed in exponential time.*

Proof: Let $\mathcal{A} = (Q,\Sigma,\mathcal{W},q_0,F,\delta)$ be a parameterized automaton. In order to construct an NFA accepting $\mathcal{L}_\diamond(\mathcal{A})$ one just needs to construct each of the $\nu(\mathcal{A})$, for each of the $|\Sigma|^{|\mathcal{W}|}$ valuations $\nu : \mathcal{W} \to \Sigma$, and then simply combine all of them with a nondeterministic choice. □

In view of these results, for the remainder of the chapter we sometimes focus solely on parameterized automata when dealing with possibility semantics, and leave the complete study of incomplete automata under possibility semantics as future work. For certainty semantics we study full incomplete automata. This does not contrast the use case and applications that were

presented in the previous section, since all the ones using possibility semantics required only parameterized automata.

We finish this section by showing that the bounds for computing NFAs in Propositions 7.3.3 and 7.3.6 are tight.

7.3.3 Lower bounds

In this section we show the optimality of the algorithms for building NFAs over Σ capturing $\mathcal{L}_\Diamond(\mathcal{A})$ and $\mathcal{L}_\Box(\mathcal{A})$, by providing matching lower bounds on the sizes of such NFAs. For certainty semantics we actually prove a stronger result, since both of our lower bounds shall be given with parameterized regular expressions.

We say that the *sizes of minimal NFAs for $\mathcal{L}_*$ are necessarily exponential (respectively, double-exponential)* if there exists a family $\{e_n\}_{n\in\mathbb{N}}$ of parameterized regular expressions such that:

- the size of each e_n is $O(n)$, and
- every NFA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e_n)$ has at least 2^n (resp., 2^{2^n}) states.

Theorem 7.3.7 *The sizes of minimal NFAs are necessarily double-exponential for $\mathcal{L}_\Box$, and necessarily exponential for $\mathcal{L}_\Diamond$.*

Proof: We begin with the double exponential bound for $\mathcal{L}_\Box$. For each $n \in \mathbb{N}$, let e_n be the following parameterized regular expression over alphabet $\Sigma = \{0,1\}$ and variables $x_1, \ldots, x_{n+1}$:

$$e_n = ((0 \mid 1)^{n+1})^* \cdot x_1 \cdots x_n \cdot x_{n+1} \cdot ((0 \mid 1)^{n+1})^*.$$

Notice that each e_n uses $n+1$ variables, and is of linear size in n. We first show a technical lemma:

Lemma 7.3.8 *Let $u \in \{0,1\}^{n+1}$ be a word of size $n+1$. Then u is a subword of every word $w \in \mathcal{L}_\Box(e_n)$. Moreover, there is a match for u in w that starts in a position j of w ($1 \leq j \leq |w|$) such that $j = 1 \mod n+1$.*

Proof: Consider an arbitrary word $u = u_1, \ldots, u_{n+1} \in \{0,1\}^{n+1}$, and let ν be the valuation for e_n such that $\nu(x_i) = u_i$, for $1 \leq i \leq (n+1)$. Then $\nu(e_n) = ((0 \mid 1)^{n+1})^* \cdot u \cdot ((0 \mid 1)^{n+1})^*$, and thus all words w in $\mathcal{L}(\nu(e_n))$ contain u as a subword, matching in a position $j = 1 \mod n+1$ of w. The lemma follows since by definition $\mathcal{L}_\Box(e_n) \subseteq \mathcal{L}(\nu(e_n))$. □

We now show that every NFA deciding $\mathcal{L}_\Box(e_n)$ has 2^{2^n} states. Our main tool comes from [Glaister and Shallit, 1996], and can be shown using a standard communication complexity argument:

Theorem 7.3.9 *[Glaister and Shallit, 1996] If $L \subset \Sigma^*$ is a regular language, and there exists a set of pairs $P = \{(u_i, v_i) \mid 1 \leq i \leq m\} \subseteq \Sigma^* \times \Sigma^*$ such that:*

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

then every NFA accepting L has at least m 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 $\{0,1\}^{n+1}$ that are not in S.

Then define a set of pairs $P_n = \{(w_S, w_{\bar{S},n}) \mid S \subset \{0,1\}^{n+1} \text{ and } |S| = 2^n\}$. Since there are 2^{n+1} binary words of length $n+1$, there are $\binom{2^{n+1}}{2^n}$ different subsets of $\{0,1\}^{n+1}$ of size 2^n, and thus P_n contains $\binom{2^{n+1}}{2^n} \geq 2^{2^n}$ pairs. Next, we show that $\mathcal{L}_\Box(e_n)$ and P_n satisfy properties (1) and (2) in Theorem 7.3.9, which proves the double exponential lower bound.

1. We need to show that for every set $S \subset \{0,1\}^{n+1}$ of size 2^n, the word $w_S \cdot w_{\bar{S},n}$ belongs to $\mathcal{L}(\nu(e_n))$, for every possible valuation $\nu : \Sigma \to \{x_1, \ldots x_{n+1}\}$. Let then S be an arbitrary subset of $\{0,1\}^{n+1}$ of size 2^n, and let ν be an arbitrary valuation from Σ to $\{x_1, \ldots, x_{n+1}\}$. Define $u = \nu(x_1) \cdots \nu(x_{n+1})$. Then u is a substring of either w_S or $w_{\bar{S},n}$. Assume the former is true (the other case is analogous). Then the word $w_S \cdot w_{\bar{S},n}$ can be written in the form $v \cdot u \cdot v' \cdot w_{\bar{S},n}$, with $v, v' \in \mathcal{L}((0 \mid 1)^{n+1})$. This shows that $w_S \cdot w_{\bar{S},n}$ belongs to $\mathcal{L}(\nu(e_n))$. Since ν was arbitrarily chosen, we have that $w_S \cdot w_{\bar{S},n}$ belongs to $\mathcal{L}_\Box(e_n)$.

2. Assume for the sake of contradiction that there are distinct subsets S_1, S_2 of $\{0,1\}^{n+1}$ of size 2^n such that $w_{S_1} \cdot w_{\bar{S}_2,n}$ belongs to $\mathcal{L}_\Box(e_n)$. Since S_1 and S_2 are distinct, proper subsets of $\{0,1\}^{n+1}$ (they are of size 2^n), there must be a word in $\{0,1\}^{n+1}$ that belongs to S_2 but not to S_1. Let s be such word. Given that the word $w_{S_1} \cdot w_{\bar{S}_2,n}$ belongs to $\mathcal{L}_\Box(e_n)$, by Lemma 7.3.8 we have that s is a subword of $w_{S_1} \cdot w_{\bar{S}_2,n}$ that matches $w_{S_1} \cdot w_{\bar{S}_2,n}$ in a position j such that $j = 1 \mod n+1$. There are two possibilities. First, it could be that $j < |w_{S_1}|$. But since $j = 1 \mod n+1$, this means that s corresponds to one of the words in S_1, that gives form to w_{S_1}, which is a contradiction. On the other hand, if $j \geq |w_{S_1}|$, using essentially the same argument we conclude that s does not belong to S_2, which is also a contradiction.

We use essentially the same technique to address the $\Diamond$-semantics. To show the exponential lower bound for $\mathcal{L}_\Diamond$, define $e_n = (x_1 \cdots x_n)^*$, and let $P_n = \{(w, w) \mid w \in \{0,1\}^n\}$. Clearly, P_n contains 2^n pairs. All that is left to do is to show that $\mathcal{L}_\Diamond(e_n)$ and P_n satisfy properties (1) and (2) in Theorem 7.3.9.

1. From the fact that $\mathcal{L}_{\diamond}(e_n) = \bigcup_{w \in \{0,1\}^n} w^*$, we have that for each $u \in \{0,1\}^n$ the word uu belongs to $\mathcal{L}_{\diamond}(e_n)$.

2. The same fact shows that for every $u, v \in \{0,1\}^n$, if $u \neq v$, then $uv \notin \bigcup_{w \in \{0,1\}^n} w^*$, and thus $uv \notin \mathcal{L}_{\diamond}(e_n)$.

This finishes the proof of the theorem. □

Chapter 8

Decision Problems for Incomplete Automata

The goal of this chapter is to determine the exact complexity of key problems related to languages $\mathcal{L}_\Box(\mathcal{A})$ and $\mathcal{L}_\Diamond(\mathcal{A})$. We consider standard language-theoretic decision problems, such as membership of a word in the language, language nonemptiness, universality, and containment. Motivated by the applications of incomplete automata, we also study the problem of computing the intersection of $\mathcal{L}_\Box(\mathcal{A})$ and $\mathcal{L}_\Diamond(\mathcal{A})$ with a regular language. For all of these problems, upper bounds shall be given in terms of incomplete automaton, and we give matching lower bounds that hold even when the input to the problem is restricted to parameterized regular expressions.

8.1 Definition of the Problems

We now describe the decision problems we study in this Section. We study both semantics, so for each problem we shall have two versions, depending on which semantics – $\mathcal{L}_\Box$ or $\mathcal{L}_\Diamond$ – is used. So each problem will have a subscript $*$ that can be interpreted as $\Box$ or $\Diamond$. Note that these problems are motivated from the applications mentioned in Section 7.2

NONEMPTINESS$_*$ Given an incomplete automaton $\mathcal{A}$, is $\mathcal{L}_*(\mathcal{A}) \neq \emptyset$?

MEMBERSHIP$_*$ Given an incomplete automaton $\mathcal{A}$ and a word $w \in \Sigma^*$, is $w \in \mathcal{L}_*(\mathcal{A})$?

UNIVERSALITY$_*$ Given an incomplete automaton $\mathcal{A}$, is $\mathcal{L}_*(\mathcal{A}) = \Sigma^*$?

CONTAINMENT$_*$ Given incomplete automata $\mathcal{A}_1$ and $\mathcal{A}_2$, is $\mathcal{L}_*(\mathcal{A}_1) \subseteq \mathcal{L}_*(\mathcal{A}_2)$?

NONEMPTYINTREG$_*$ Given an incomplete automata $\mathcal{A}$, and a regular language L over Σ, is $L \cap \mathcal{L}_*(\mathcal{A}) \neq \emptyset$?

Problem \ Semantics	Certainty □	Possibility ◇
NONEMPTINESS	EXPSPACE-complete	NLOGSPACE-complete (for automata)
MEMBERSHIP	coNP-complete	NP-complete
CONTAINMENT	EXPSPACE-complete	EXPSPACE-complete†
UNIVERSALITY	PSPACE-complete	EXPSPACE-complete†
NONEMPTYINTREG	EXPSPACE-complete	NP-complete†

†: For parameterized automata

Figure 8.1: Summary of complexity results

The last problem is motivated by the task of computing certain paths over graph patterns, as seen in Section 7.2.

The table in Fig. 8.1 summarizes the main results in Sections 8.2 to 8.6.

8.2 Nonemptiness

The problem NONEMPTINESS$_\Diamond$ has a trivial algorithm for parameterized regular expressions. So we consider regular expressions for the certainty semantics only; for the possibility semantics, we give the lower bound for incomplete automata.

Theorem 8.2.1
- *The problem* NONEMPTINESS$_\Box$ *is* EXPSPACE*-complete. It remains* EXPSPACE*-hard even if the input is a parameterized regular expression.*
- *The problem* NONEMPTINESS$_\Diamond$ *is* NLOGSPACE*-complete.*

The result for the possibility semantics is by a standard reachability argument. Note that the bound is the same here as in the case of infinite alphabets studied in [Grumberg et al., 2010]. To see the upper bound for NONEMPTINESS$_\Box$, note that by Lemma 7.3.2 we can focus instead on $\mathcal{A}^{fin}$. For these automata we have shown that there are exponentially many valuations ν, and each automaton $\nu(\mathcal{A}^{fin})$ is of polynomial size, so we can use the standard algorithm for checking nonemptiness of the intersection of a family of regular languages which can be solved in polynomial space in terms of the size of its input; since the input to this problem is of exponential size in terms of the original input, the EXPSPACE bound follows.

The hardness is by a generic (Turing machine) reduction; we show it even for the case of parameterized regular expressions. In the proof we use the following property of the certainty semantics, that shows a striking difference between incomplete automata and NFAs:

Lemma 8.2.2 *Given a set $e_1, \ldots, e_k$ of parameterized expressions of size at most $n \geq k$, it is possible to build, in time $O(|\Sigma| \cdot k^2 \cdot n)$ an expression e' such that $\mathcal{L}_\Box(e')$ is empty if and only if $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k)$ is empty.*

Of course, since we can construct an equivalent parameterized automata from each parameterized regular expression, Lemma 8.2.2 holds as well for parameterized automata. But one can also show that this result holds even for incomplete automata, i.e. given incomplete automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ it is also possible to construct, in polynomial time, an incomplete automaton $\mathcal{A}'$ such that $\mathcal{L}_\Box(\mathcal{A}')$ is empty if and only if $\mathcal{L}_\Box(\mathcal{A}_1) \cap \cdots \cap \mathcal{L}_\Box(\mathcal{A}_k)$ is empty. For completeness, we provide the proof of this remark in the Appendix.

The reason the case of the $\mathcal{L}_\Box(e)$ semantics is so different from the usual semantics of regular languages is as follows. It is well known that checking whether the intersection of the languages defined by a finite set S of regular expressions is nonempty is PSPACE-complete [Kozen, 1977], and hence under widely held complexity-theoretical assumptions no regular expression r can be constructed in polynomial time from S such that $\mathcal{L}(r)$ is nonempty if and only if $\bigcap_{s \in S} \mathcal{L}(s)$ is nonempty. Lemma 8.2.2, on the other hand, says that such a construction is possible for parameterized regular expressions under the certainty semantics. Next we prove Lemma 8.2.2:

Proof: Assume first that Σ has at least two symbols. Let $e_1, \ldots, e_k$ be parameterized regular expressions as stated in the Lemma, and let a, b be different symbols in Σ. We use $(\Sigma - a)$ as a shorthand for the expression whose language is the union of every symbol in Σ different from a, and define $A^i = \left((\Sigma - a)^* \cdot a \cdot (\Sigma - a)^*\right)^i$, for $1 \leq i \leq k-1$. Finally, let $x_1, \ldots, x_{k-1}$ be fresh variables. We define e' as

$$(\Sigma - a)^* \cdot x_1 \cdot (\Sigma - a)^* \cdot x_2 \cdot (\Sigma - a)^* \cdots x_{k-1} \cdot (\Sigma - a)^* \cdot \\ \left(ba^k b \cdot e_1 \mid b \cdot A^1 \cdot ba^k b \cdot e_2 \mid b \cdot A^2 \cdot ba^k b \cdot e_3 \mid \cdots \mid b \cdot A^{k-1} \cdot ba^k b \cdot e_k\right)$$

We prove next that $\mathcal{L}_\Box(e') \neq \emptyset$ if and only if $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k) \neq \emptyset$. For the *if* direction, consider a word $w \in \Sigma^*$ that belongs to $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k)$. Then it can be observed from the construction of e' that the word $(\bar{c}^k ab)^{k-1} ba^k bw$ belongs to $\mathcal{L}_\Box(e')$ where $\bar{c}$ is the concatenation (say, in lexicographical order) of all the symbols in Σ different from a.

On the other hand, assume that a word w belongs to $\mathcal{L}_\Box(e')$. It is clear that w must contain the substring $ba^k b$. Thus, there are words $u, v \in \Sigma^*$ such that $w = u \cdot ba^k b \cdot v$, and u does not contain the word $ba^k b$ as a substring. Our goal is to prove that v belongs to $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k)$. But first we need to show that u contains exactly $k-1$ appearances of the symbol a. We prove this statement by contradiction. Assume first that u contains less than $k-1$ appearances of the symbol a. Then consider a valuation ν that maps each variable in e' to the

symbol a. Since $\nu(e')$ is of the form

$$((\Sigma - a)^* a)^{k-1}(\Sigma - a)^* \cdot \big(ba^k b \cdot \nu(e_1) \mid b \cdot A^1 \cdot ba^k b \cdot \nu(e_2) \mid b \cdot A^2 \cdot ba^k b \cdot \nu(e_3) \mid \cdots \mid b \cdot A^{k-1} \cdot ba^k b \cdot \nu(e_k)\big),$$

we conclude that the language of $\nu(e')$ cannot contain any word that starts with $u \cdot ba^k b$, since we have assumed that u contains less than $k-1$ appearances of the symbol a. Next, assume that u contains more than $k-1$ appearances of the symbol a, and consider a valuation ν' that maps each variable in e' to the symbol b. Then $\nu'(e')$ is of the form

$$((\Sigma - a)^* b)^{k-1}(\Sigma - a)^*(ba^k b \cdot \nu'(e_1) \mid b \cdot A^1 \cdot ba^k b \cdot \nu'(e_2) \mid b \cdot A^2 \cdot ba^k b \cdot \nu'(e_3) \mid \cdots \mid b \cdot A^{k-1} \cdot ba^k b \cdot \nu'(e_k)).$$

Recall that we define A^i as $A^i = \big((\Sigma - a)^* \cdot a \cdot (\Sigma - a)^*\big)^i$. Then notice that any word in $\mathcal{L}(\nu'(e'))$ is such that the symbol a cannot appear more than $k-1$ times before the substring $ba^k b$. We conclude that $\mathcal{L}(\nu'(e'))$ cannot contain a word starting with $u \cdot ba^k b$.

We have just proved that w can be decomposed into $u \cdot ba^k b \cdot v$, where u does not contain the substring $ba^k b$ and has exactly $k-1$ appearances of the symbol a. With this observation, and the assumption that w belongs to $\mathcal{L}_\square(e')$, it is not difficult to show the following fact. If ν is a valuation for e' that assigns the symbol a to exactly j variables in $\{x_1, \ldots, x_{k-1}\}$ $(0 \leq j \leq k-1)$, then the word v must belong to $\mathcal{L}_\square(e_{k-j})$. This proves that v belongs to $\mathcal{L}_\square(e_1) \cap \cdots \cap \mathcal{L}_\square(e_k)$, which was to be shown.

For the case when Σ contains a single symbol a, notice that for each $1 \leq i \leq k$ it is the case that $\mathcal{L}_\square(e_i) = \mathcal{L}(e_i')$, where e_i' is the expression resulting from replacing all parameters in e_i with the symbol a. We perform this replacement, and afterwards augment Σ with a fresh new symbol. The construction previously explained can be then used on input $e_1', \ldots, e_k'$. The correctness of this algorithm follows directly from the proof of the previous case, and the fact that the expressions $e_1', \ldots, e_k'$ contain no variables.

Regarding the size of the expression e', we have that the size of the first part of e', corresponding to $(\Sigma - a)^* \cdot x_1 \cdot (\Sigma - a)^* \cdot x_2 \cdot (\Sigma - a)^* \cdots x_{k-1} \cdot (\Sigma - a)^*$, is $O(|\Sigma| \cdot k)$. Furthermore, the second part comprises of a union of k expressions, each of them of size $O(|\Sigma| \cdot k \cdot n)$. Thus, the size of e' is $O(|\Sigma| \cdot k^2 \cdot n)$. □

Lower bound for certainty semantics: To complete the proof of Theorem 8.2.1, we prove an EXPSPACE lower bound for NONEMPTINESS$_\square$, using a reduction from the acceptance problem for deterministic Turing machines that work in exponential space. Along the proof we use the shorthand $[i]$ to denote the binary representation of the number $i < 2^n$ as a string of n symbols from $\{0,1\}$. For example, $[0]$ corresponds to the word 0^n, and $[2]$ corresponds to the word $0^{n-2}10$.

Let $L \subseteq \Sigma^*$ be a language that belongs to EXPSPACE, and let $\mathcal{M}$ be a 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 parameterized regular expression $e_{\mathcal{M},\bar{a}}$ such that $\mathcal{L}_\Box(e_{\mathcal{M},\bar{a}}) \neq \emptyset$ if and only if $\mathcal{M}$ accepts $\bar{a}$.

Assume that $\mathcal{M} = (Q, \Gamma, q_0, \{q_m\}, \delta)$, where $Q = \{q_0, \ldots, q_m\}$ is the set of states, $\Gamma = \{0, 1, B\}$ is the tape alphabet (B is the *blank* symbol), the initial state is q_0, q_m is the unique final state, and $\delta : (Q \setminus \{q_m\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Notice that we assume without loss of generality that no transition is defined on the final state q_m. Furthermore, we also assume without loss of generality that every accepting run of $\mathcal{M}$ ends after an odd number of computations. 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 8.2.2, it suffices to construct a set E of parameterized regular expressions, such that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ is empty if and only if $\mathcal{M}$ accepts on input $\bar{a}$.

Consider the alphabet $\Sigma = \{0, 1\}$. The idea of the reduction is to code the run of $\mathcal{M}$ on input $\bar{a}$ into a word in Σ^*, in such a way that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ contains precisely the words that code an accepting run τ for $\mathcal{M}$ on input $\bar{a}$. In intuitive terms, such a word w represents the sequence of "instant descriptions" of $\mathcal{M}$ with respect to run τ. We do it as follows.

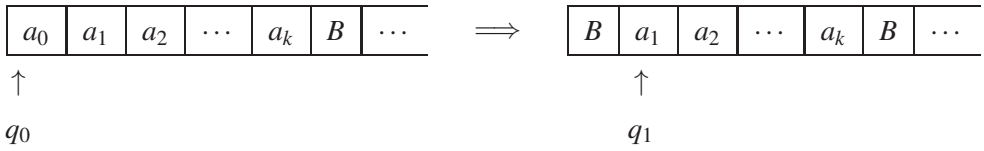
Assume that $\mathcal{M}$ performs m computations according to the run τ. With each $1 \leq i \leq 2^n$ and $i \leq j \leq m$, we associate a symbol $b_{i,j} \in \Gamma \cup (\Gamma \times Q)$, in such a way that $b_{i,j}$ corresponds to the symbol in the i-th cell of the tape in the j-th step of the run τ, if the head of $\mathcal{M}$ in the j-th step of the computation is not pointing into such cell, and otherwise as the pair (c, q), where c is the symbol in the i-th cell of the tape in the j-th step of the run τ, and q is the state of $\mathcal{M}$ in the j-th step of τ. We need each $b_{i,j}$ to be coded as a string over $\{0, 1\}$. In order to do this, let $p = |\Gamma \cup (\Gamma \times Q)|$. We shall code each symbol in $\Gamma \cup (\Gamma \times Q)$ in unary, i.e. as a p-bit string. We denote by $[b_{i,j}]$ the unary representation of the symbol $b_{i,j}$.

We also need to include information about the *action* that was performed in each cell of $\mathcal{M}$ at each step of the computation (i.e. read the cell, point the cell after moving the pointer, or nothing). More precisely, let $[nothing] = 100$, $[read] = 101$ and $[head] = 111$, and define, for each $1 \leq i \leq 2^n$ and $1 \leq j \leq m$, the string $[\mathrm{ac}_{i,j}]$ as $[read]$ if $\mathcal{M}$ is to read the content of the i-th cell at the j-th step of the computation; $[head]$, if after the j-th computation $\mathcal{M}$ moves the head to point into the i-th cell of the tape, and $[nothing]$ otherwise.

Roughly speaking, the idea is to define $w = w_1 \cdot w_2 \cdots w_m$, where each w_j is of the form:

$$\begin{aligned}
&[\mathrm{ac}_{(0,j)}] \cdot [b_{(0,j)}] \cdot [0] \cdot [b_{(0,j+1)}] \cdot \\
&\qquad [\mathrm{ac}_{(1,j)}] \cdot [b_{(1,j)}] \cdot [1] \cdot [b_{(1,j+1)}] \cdot \\
&\qquad\qquad \vdots \\
&\qquad [\mathrm{ac}_{(2^n-1,j)}] \cdot [b_{(2^n-1,j)}] \cdot [2^n-1] \cdot [b_{(2^n-1,j+1)}] \qquad (8.1)
\end{aligned}$$

For example, assume that in the first step of the computation, $\mathcal{M}$ reads the first cell of the tape, writes a blank symbol, changes from state q_0 to q_1, and advances to the right. That is, the first and second configurations of $\mathcal{M}$ are as depicted in the following figure:



Then w_1 corresponds to the string

$$\begin{array}{cccccccc}
[\mathit{read}] & \cdot & [(a_0,q_0)] & \cdot & [0] & \cdot & [B] & \cdot \\
[\mathit{head}] & \cdot & [a_1] & \cdot & [1] & \cdot & [(a_1,q_1)] & \cdot \\
[\mathit{nothing}] & \cdot & [a_2] & \cdot & [2] & \cdot & [a_2] & \cdot \\
 & & \vdots & & & & & \\
[\mathit{nothing}] & \cdot & [a_k] & \cdot & [k] & \cdot & [a_k] & \cdot \\
[\mathit{nothing}] & \cdot & [B] & \cdot & [k+1] & \cdot & [B] & \cdot \\
 & & \vdots & & & & & \\
[\mathit{nothing}] & \cdot & [B] & \cdot & [2^n-1] & \cdot & [B] & \cdot
\end{array}$$

Essentially, we use 4 substrings to describe the action on each cell of the tape. The first substring, of length 3, refers to the action performed in that computation. In this case, an action $[\mathit{read}]$ accompanies the first cell, since it was the cell read in the first step of the computation, and an action $[\mathit{head}]$ accompanies the second cell, since as a result of the computation the head of $\mathcal{M}$ is now pointing into that cell. As expected, all other actions in w_1 are set to $[\mathit{nothing}]$, since nothing was done to those cells in the first step of the computation. The second substring (of length p) refers to the content of the cell before the computation, the third is of length n, and contains the number identifying a particular cell as the i-th cell, from left to right, where i is binary, and the fourth string, of length p, is the content of that cell right after the computation.

Finally, we also need to explicitly distinguish *even* and *odd* computations of $\mathcal{M}$. Formally, let $[\mathit{even}] = 000$ and $[\mathit{odd}] = 001$. We construct E in such a way that if there is a word w in $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ then it is of the form:

$$[\mathit{even}] \cdot w_1 \cdot [\mathit{even}] \cdot [\mathit{odd}] \cdot w_2 \cdot [\mathit{odd}] \cdot [\mathit{even}] \cdot w_3 \cdot [\mathit{even}] \cdots [\mathit{odd}] \cdot w_m \cdot [\mathit{odd}],$$

where each w_j is of the form (8.1), as explained above.

The rest of the proof is devoted to construct such set E. We divide the set E into sets E^1, E^2, E^3, E^4 and E^5.

First, E^1 contains only the expression

$$\Big([even]\big([action](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)^*[even][odd]\big([action](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)^*[odd]\Big)^*,$$

where $[action]$ is just a shorthand for the expression $([read] \mid [head] \mid [nothing])$. In intuitive terms, it ensures that all words accepted by $\bigcap_{e\in E} \mathcal{L}_\square(e)$ are repetitions of sequences of subwords of length $3+2p+n$, contained between $[even]$ or $[odd]$ strings.

So far, we only have that all words in $\bigcap_{e\in E} \mathcal{L}_\square(e)$ must be of the above form. The next step is to ensure that the number of substrings of the form $\big([action](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)$ between any two strings $[even]$ or $[odd]$ has to be precisely 2^n (one for each cell used in the tape) and, furthermore, the numbers in binary representation used to code the position of the cell in each of these substrings (i.e., the part corresponding to $(0 \mid 1)^n$) have to be arranged in numerical order. To ensure this we use a set of regular expressions E^2. It is defined in such a way that the language $\bigcap_{e\in E^2} L(e)$ corresponds to the language accepted by the expression:

$$\begin{aligned}\big(([even] \mid [odd]) \cdot [action] \cdot (0 \mid 1)^p \cdot [0] \cdot (0 \mid 1)^p \cdot \\ [action] \cdot (0 \mid 1)^p \cdot [1] \cdot (0 \mid 1)^p \cdot \\ \vdots \\ [action] \cdot (0 \mid 1)^p \cdot [2^n-1] \cdot (0 \mid 1)^p \cdot ([even] \mid [odd])\big)^*\end{aligned}$$

The definition of the set E^2 is standard, but very technical, and it is therefore omitted. It is based on the idea of representing the string $[0]\cdot[1]\cdots[2^n-1]$ as an intersection of a polynomial number of regular expressions stating all together that, for each even $i \leq 2^n-1$, the string $[i]$ has to be followed by the string $[i+1]$, and likewise for each odd number (see e.g. [Kozen, 1977]).

Next, we ensure that the state and contents of the cells are carried along the descriptions. More precisely, E^3 must ensure that, if for some $1 \leq i < 2^n$ and $1 \leq j \leq m$, the word w_j features a substring of the form:

$$[even]\cdots[action]\cdot(0 \mid 1)^p \cdot [i] \cdot [b_{(i,j)}]\cdots[even],$$

with $b_{(i,j)} \in \Gamma \cup (\Gamma \times Q)$, then it must be directly followed by a string of form

$$[odd]\cdots[action]\cdot[b_{(i,j)}]\cdot[i]\cdots[odd],$$

so that the slots representing the content of the i-th cell after the j-th computation coincide with the slots representing the content of the i-th cell before the $j+1$-th computation.

It is straightforward to state such a condition by enumerating all cases, for each $0 \leq i \leq 2^n - 1$, but this would yield exponentially many equations. Instead, we exploit the use of parameters in our expression. We include in E^3 parameterized expressions E^3_1 and E^3_2, where E^3_1 (and, correspondingly, E^3_2) force that the content of the cell $x_1 \cdot x_2 \cdots x_n$ after an even (correspondingly, odd) computation corresponds exactly to the state before the next computation. To define E^3_1, consider the following expressions, for each $b \in \Gamma \cup (\Gamma \times Q)$:

$$E^3_{(1,b,\text{even})} = [even] \cdot ([action] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [action] \cdot (0 \mid 1)^p \cdot x_1 \cdots x_n \cdot [b] \cdot ([action] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [even]$$

$$E^3_{(1,b,\text{odd})} = [odd] \cdot ([action] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [action] \cdot [b] \cdot x_1 \cdots x_n \cdot (0 \mid 1)^p \cdot ([action] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [odd]$$

Then we define E^3_1 as follows:

$$E^3_1 = \left(\bigcup_{b \in \Gamma \cup (\Gamma \times Q)} \left(E^3_{(1,b,\text{even})} \cdot E^3_{(1,b,\text{odd})} \right) \right)^*$$

Expression E^3_2 is defined accordingly, simply by interchanging the order of $[even]$ and $[odd]$ strings, carefully checking that the first step of the computation is even, and allowing for the possibility that a word representing a computation ends in an odd configuration (that is, an odd configuration may be followed by an even configuration with the aforementioned properties, or may be the last configuration of the computation).

All that is left to do is to construct regular expressions that ensure that each of the substrings w_j $(1 \leq j \leq m)$ of the word w in $\bigcap_{e \in E} \mathcal{L}_\square(e)$ represent valid computations of $\mathcal{M}$. This is done by set E^4 of expressions, accepting all words such that:

- Between each two consecutive $[even]$ or $[odd]$ strings there is exactly one $[read]$ and one $[head]$ in the slots devoted to $[action]$ in form (8.1).
- No other cell can change its context, except for those marked with $[read]$ or $[head]$, and
- The content that changes in the cells marked by $[read]$ and $[head]$ respects the transition function δ of $\mathcal{M}$.

Moreover, we also add a set of expressions E^5, accepting words such that:

- The initial configuration of $\mathcal{M}$ is encoded as the first step of the computation represented by w.

- The last computation ends in a final state of $\mathcal{M}$.

It is a tedious, but straightforward task to define the sets E^4 and E^5 of expressions. Furthermore, the fact that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ is empty if and only if $\mathcal{M}$ accepts on input $\bar{a}$ follows immediately from the remarks given along the construction. This finishes the proof. □

The generic reduction used in the proof of EXPSPACE-hardness of NONEMPTINESS$_\Box$ also provides lower bounds on the minimal sizes of words in languages $\mathcal{L}_\Box(e)$ (note that the language $\mathcal{L}_\Diamond(e)$ always contains a word of linear size in $|e|$).

Corollary 8.2.3 *There exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a sequence of parameterized regular expressions $\{e_n\}_{n \in \mathbb{N}}$ such that each e_n is of size at most $p(n)$, and every word in the language $\mathcal{L}_\Box(e_n)$ has size at least 2^{2^n}.*

The single exponential bound was already hinted in Example 7.2.2. Let us now explain the doubly exponential bound.

Proof: Clearly, for each $n \in \mathbb{N}$, it is possible to construct a deterministic Turing machine $\mathcal{M}_n$ over alphabet $\Sigma = \{0, 1\}$ that on input 1^n works for exactly 2^{2^n} steps, using 2^n cells. Furthermore, it is possible to specify this machine using polynomial size with respect to n.

Next, using the construction in the reduction of Theorem 8.2.1, construct a set of parameterized regular expressions $E_{(\mathcal{M}_n, 1^n)}$ such that the single word $w_n \in \bigcap_{e \in E_{(\mathcal{M}_n, 1^n)}} \mathcal{L}_\Box(e)$ represents a run (or, more precisely, a sequence of configurations) of $\mathcal{M}_n$ on input 1^n. Note that each set $E_{(\mathcal{M}_n, 1^n)}$ is of size polynomial with respect to n. Moreover, according to the reduction in the proof of Theorem 8.2.1, $\bigcap_{e \in E_{(\mathcal{M}_n, 1^n)}} \mathcal{L}_\Box(e)$ contains a single word, of length greater than 2^{2^n}, representing the single run of $\mathcal{M}_n$ on input 1^n.

For each n, we define e_n as the expression such that $\mathcal{L}_\Box(e_n)$ is empty if and only if $\bigcap_{e \in E_{(\mathcal{M}_n, 1^n)}} \mathcal{L}_\Box(e)$ is empty, constructed as in the proof of Lemma 8.2.2, so that e_n is of size polynomial with respect to $E_{(\mathcal{M}_n, 1^n)}$. It follows from the proof of such lemma that every word accepted by $\mathcal{L}_\Box(e_n)$ has size at least 2^{2^n}. □

8.3 Membership

We now turn to the membership problem. We can easily show coNP-hardness for incomplete automata using the results in Chapter 4 and the remark in Proposition 7.2.1. However, upper bounds for both semantics, and lower bounds for parameterized regular expressions, remain to be shown.

Theorem 8.3.1

- *The problem* MEMBERSHIP$_\Box$ *is* coNP*-complete. It remains* coNP*-hard even if the input is a parameterized regular expression.*
- *The problem* MEMBERSHIP$_\Diamond$ *is* NP*-complete. It remains* NP*-hard even if the input is a parameterized regular expression.*

Proof: We only show the upper bounds, the lower bounds follow from a much more stronger result that we present afterwards.

We begin with MEMBERSHIP$_\Box$ 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 7.3.3 shows that each NFA $\nu(\mathcal{A}^{fin})$ is of polynomial size w.r.t. $\mathcal{A}$, 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 $*$ in a non-trivial way), 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 7.3.2.

For the case of MEMBERSHIP$_\Diamond$, one just needs to guess a valuation $\nu(\eta, \theta)$ for $\mathcal{A}$ such that $\nu(\mathcal{A})$ accepts the word w. Of course, θ needs not send any transition in $\mathcal{A}$ of form (q_1, L, q_2) to a transition (q_1, w', q_2) where w' is of size greater that w, so there always exists one such valuation ν of polynomial size. □

The similarities between incomplete automata and certain answers for graph pattern are even more evident with this next result, showing that the lower bounds for the membership problem are very resilient. To show this, we adapt the notion of codd patterns and acyclic underlying graphs for the case of parameterized regular expression, as follows:

We say that a parameterized regular expression is *simple* if each variable occurs only once in the expression. Moreover, an expression has star height 0 if it does not make use of the Kleene star: these denote finite languages, and each finite language is denoted by such an expression.

Note that these restrictions are actually stronger than the equivalent restrictions for parameterized automata. In fact, it is not difficult to construct an incomplete automata with a single label variable, used only once, that is not equivalent to any simple parameterized regular expression.

Proposition 8.3.2 *The complexity of the membership problem remains as in Theorem 8.3.1 over the classes of simple expressions, and expressions of star-height* 0*. Over the class of simple expressions of star-height* 0*,* MEMBERSHIP$_\Diamond$ *can be solved in polynomial time (actually, in time* $O(nm\log^2 n)$*, where n is the size of the expression and m is the size of the word).*

Proof: For the sake of readability, in this proof we use $\cup$ – instead of $|$ – for representing the operation of union between regular expressions.

1) $\diamond$-semantics: We first consider the $\diamond$-semantics. We start by showing NP-hardness of MEMBERSHIP$_\diamond$, for regular expressions of star-height 0. We use a reduction from POSITIVE 1-3 3-SAT, which is the following NP-hard decision problem: Given a conjunction φ of clauses, with exactly three literals each, and in which no negated variable occurs, is there a truth assignment to the variables so that each clause has exactly one true variable?

The reduction is as follows. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a formula in CNF, where each C_i ($1 \leq i \leq m$) is a clause consisting of exactly tree positive literals. Let $\{p_1, \ldots, p_n\}$ be the variables that appear in φ. With each propositional variable p_i ($1 \leq i \leq n$) we associate a different variable $x_i \in \mathbf{V}$. We show next how to construct, in polynomial time from φ, a parameterized regular expression e over alphabet $\Sigma = \{a, 0, 1\}$ and a word w over the same alphabet, such that there is an assignment to the variables of φ for which each clause has exactly one true variable if and only if $w \in \mathcal{L}_\diamond(e)$.

The parameterized regular expression e is defined as $ae_1ae_2a\cdots ae_ma$, where the regular expression e_i, for $1 \leq i \leq m$, is defined as follows: Assume that $C_i = (p_j \vee p_k \vee p_\ell)$, where $1 \leq j,k,\ell \leq n$. Then e_i is defined as $(x_jx_kx_\ell \,|\, x_jx_\ell x_k \,|\, x_kx_jx_\ell \,|\, x_kx_\ell x_j \,|\, x_\ell x_jx_k \,|\, x_\ell x_kx_j)$. That is, e_i is just the union of all the possible forms in which the variables in $\mathbf{V}$ that correspond to the propositional variables that appear in C_i can be ordered. Further, the word w is defined as $(a100)^m a$. Clearly, e and w can be constructed in polynomial time from φ. Next we show that there is an assignment for variables $\{p_1, \ldots, p_n\}$ for which each clause has exactly one true variable if and only if $w \in \mathcal{L}_\diamond(e)$.

Assume first that $w \in \mathcal{L}_\diamond(e)$. Then there exists a valuation $\nu : \{x_1, \ldots, x_n\} \to \Sigma$ such that $w \in \mathcal{L}(\nu(e))$. Thus, it must be the case that the word $a100$ belongs to $\nu(ae_i)$, for each $1 \leq i \leq m$. But this implies that if $C_i = (x_j \vee x_k \vee x_\ell)$, then ν assigns value 1 to exactly one of the variables in the set $\{x_j, x_k, x_\ell\}$ and it assigns value 0 to the other two variables. Let us define now a propositional assignment $\sigma : \{p_1, \ldots, p_n\} \to \{0,1\}$ such that $\sigma(p_i) = \nu(x_i)$, for each $1 \leq i \leq n$. It is not hard to see then that for each clause C_j, $1 \leq j \leq m$, σ assigns value 1 to exactly one of its propositional variables.

Assume, on the other hand, that there is a propositional assignment $\sigma : \{p_1, \ldots, p_n\} \to \{0,1\}$ that assigns value 1 to exactly one variable in each clause C_i, $1 \leq i \leq m$. Let us define ν as a valuation from $\{x_1, \ldots, x_n\}$ into $\{0,1\}$ such that $\nu(x_i) = 1$ if and only if $\sigma(p_i) = 1$. Clearly then $100 \in \mathcal{L}(\nu(e_i))$, for each $1 \leq i \leq m$. Thus, $(a100)^m a \in \mathcal{L}(\nu(e))$. We conclude that $w \in \mathcal{L}_\diamond(e)$.

Next we prove NP-hardness of MEMBERSHIP$_\diamond$ for simple expressions. We use a reduction from 3-SAT. Let $\varphi = \bigwedge_{1 \leq i \leq n}(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1, \ldots, p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for

$1 \leq k \leq m$. Next we show how to construct in polynomial time from φ, a simple regular expression e over alphabet $\Sigma = \{a,b,c,d,0,1\}$ and a word w over the same alphabet such that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$.

The regular expression e is defined as f^*, where $f := a(f_1 \cup g_1 \cup \cdots \cup f_m \cup g_m)b$, and the regular expressions f_i and g_i are defined as follows: Intuitively, f_i (resp. g_i) codifies p_i (resp. $\neg p_i$) and the clauses in which p_i (resp. $\neg p_i$) appears. Formally, we define f_i ($1 \leq i \leq m$) as

$$\Big((c^i \cup \bigcup_{\{1\leq j\leq n | p_i = \ell_j^1 \text{ or } p_i = \ell_j^2 \text{ or } p_i = \ell_j^3\}} d^j) \cdot x_i\Big),$$

where x_i is a fresh variable in $\mathbf{V}$. In the same way we define g_i as

$$\Big((c^i \cup \bigcup_{\{1\leq j\leq n | \neg p_i = \ell_j^1 \text{ or } \neg p_i = \ell_j^2 \text{ or } \neg p_i = \ell_j^3\}} d^j) \cdot \bar{x}_i\Big),$$

where $\bar{x}_i$ is a fresh variable in $\mathbf{V}$. The variable x_i (resp. $\bar{x}_i$) is said to be *associated* with p_i (resp. $\neg p_i$) in e. Clearly, e is a simple regular expression and can be constructed in polynomial time from φ.

The word w is defined as:

$$ac1bac0bacc1bacc0b\cdots ac^m1bac^m0bad1badd1b\cdots ad^n1b.$$

Clearly, w can be constructed in polynomial time from φ. Next we show that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$.

Assume first that $w \in \mathcal{L}_\diamond(e)$. That is, there is a valuation ν for the variables in the set $\{x_1,\bar{x}_1,\ldots,x_m,\bar{x}_m\}$ over Σ such that $w \in \mathcal{L}(\nu(e))$. But then, given the form of w, it is clear that ac^i1b and ac^i0b belong to $\mathcal{L}(\nu(f))$, for each $1 \leq i \leq m$. Notice that the only way for this to happen is that both $\nu(x_i)$ and $\nu(\bar{x}_i)$ take its value in the set $\{0,1\}$, and, further, $\nu(x_i) \neq \nu(\bar{x}_i)$. For the same reasons, $ad^j1b \in \mathcal{L}(\nu(f))$, for each $1 \leq j \leq n$. But the only way for this to happen is that for each $1 \leq j \leq n$ it is the case that either the variable associated with ℓ_j^1 or with ℓ_j^2 or with ℓ_j^3 in e is assigned value 1 by ν. Thus, the propositional assignment $\sigma : \{p_1,\ldots,p_m\} \to \{0,1\}$, defined as $\sigma(p_i) = 1$ if and only if $\nu(x_i) = 1$, is well-defined and satisfies φ.

Assume, on the other hand, that there is a satisfying propositional assignment $\sigma : \{p_1,\ldots,p_m\} \to \{0,1\}$ for φ. Consider the following valuation ν for e: For each $1 \leq i \leq m$ it is the case that $\nu(x_i) = \sigma(p_i)$ and $\nu(\bar{x}_i) = 1 - \sigma(x_i)$. Using essentially the same techniques as in the previous paragraph it is possible to show that $w \in \mathcal{L}(\nu(e))$, and, therefore, that $w \in \mathcal{L}_\diamond(e)$.

Next we show that MEMBERSHIP$_\diamond$ can be solved in time $O(mn \cdot \log^2 n)$ for simple expressions of star-height 0. Given a regular expression $e \in \mathrm{REG}(\Sigma,\mathbf{V})$ that is simple and of star-height 0, one can construct in time $O(n \cdot \log^2 n)$ [Hagenah and Muscholl, 1998] an ε-free NFA $\mathcal{A}$ over $\Sigma \cup \mathbf{V}$ that accepts precisely $\mathcal{L}(e)$, and satisfies the following two properties: (1) Its underlying directed graph is acyclic (this is because e does not mention the Kleene star),

and (2) for each $x \in \mathbf{V}$ that is mentioned in e there is at most one pair (q,q') of states of $\mathcal{A}$ such that $\mathcal{A}$ contains a transition from q to q' labeled x (this is because e is simple). From Lemma 7.1.4, checking whether $w \in \mathcal{L}_\diamond(e)$, for a given word $w \in \Sigma^*$, is equivalent to checking whether $w \in \mathcal{L}(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$. We show how the latter can be done in polynomial time.

First, construct in time $O(m)$ a deterministic finite automaton (DFA) $\mathcal{B}$ over Σ such that $\mathcal{L}(\mathcal{B}) = \{w\}$. We assume without loss of generality that the set Q of states of $\mathcal{A}$ is disjoint from the set P of states of $\mathcal{B}$. Next we construct, the following NFA $\mathcal{A}'$ over the alphabet $\Sigma \cup (\mathbf{V} \times \Sigma)$ as follows: The set of states of $\mathcal{A}'$ is $Q \times P$. The initial state of $\mathcal{A}'$ is the pair (q_0, p_0), where q_0 is the initial state of $\mathcal{A}$ and p_0 is the initial state of $\mathcal{B}$. The final states of $\mathcal{A}'$ are precisely the pairs $(q,p) \in Q \times P$ such that q is a final state of $\mathcal{A}$ and p is a final state of $\mathcal{B}$. Finally, there is a transition in $\mathcal{A}'$ from state (q,p) to state (q',p') labeled $a \in \Sigma$ if and only if there is a transition in $\mathcal{A}$ from q to q' labeled a and there is a transition in $\mathcal{B}$ from p to p' labeled a. There is a transition in $\mathcal{A}'$ from state (q,p) to state (q',p') labeled $(x,a) \in \mathbf{V} \times \Sigma$ if and only there is a transition in $\mathcal{A}$ from q to q' labeled x and there is a transition in $\mathcal{B}$ from p to p' labeled a. Clearly, such construction can be performed by checking all combinations of transitions of both $\mathcal{A}$ and $\mathcal{B}$, and thus it can be performed in time $O(mn \cdot \log^2 n)$. Checking whether $\mathcal{L}(\mathcal{A}') \neq \emptyset$ can easily be done in linear time with respect to the size of $\mathcal{A}'$, thus obtaining the $O(mn \cdot \log^2 n)$ bound. We prove next that checking this is equivalent to checking whether $w \in L(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$, which finishes the proof of the proposition in terms of the $\diamond$-semantics.

Assume first that $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Let $(q_0,p_0) \xrightarrow{u_1} (q_1,p_1) \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} (q_{n-1},p_{n-1}) \xrightarrow{u_n} (q_n,p_n)$ be an accepting run of $\mathcal{A}'$. That is, $u_1u_2\cdots u_n \in (\Sigma \cup (\mathbf{V} \times \Sigma))^*$ and (q_n,p_n) is a final state of $\mathcal{A}'$. Since the underlying directed graph of $\mathcal{A}$ is acyclic, and each variable x mentioned in e appears in at most one transition of $\mathcal{A}$, it must be the case that for each $1 \leq i < j \leq n$, if $u_i = (x_i,a_i) \in \mathbf{V} \times \Sigma$ and $u_j = (x_j,a_j) \in \mathbf{V} \times \Sigma$ then $x_i \neq x_j$. This implies that we can define a mapping $\nu : \mathcal{W} \to \Sigma$, where $\mathcal{W}$ is the set of variables used in transitions of $\mathcal{A}$, such that $\nu(x) = a$, if $u_i = (x,a)$ for some $1 \leq i \leq n$, and $\nu(x)$ is an arbitrary element $a' \in \Sigma$, otherwise. It is not hard to see that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ is also an accepting run of $\mathcal{L}(\nu(\mathcal{A}))$ and that $a_1a_2\cdots a_n = w$. The latter can be proved as follows: Let $f : \{u_1,\ldots,u_n\} \to \Sigma$ be the mapping such that $f(u_i) = u_i$, if $u_i = a \in \Sigma$, and $f(u_i) = a$, if $u_i = (x,a) \in \mathbf{V} \times \Sigma$. Then clearly $p_0 \xrightarrow{f(u_1)} p_1 \xrightarrow{f(u_2)} \cdots \xrightarrow{f(u_{n-1})} p_{n-1} \xrightarrow{f(u_n)} p_n$ is an accepting run of $\mathcal{B}$, and, therefore, $w = f(u_1)\cdots f(u_n)$. Further, let $g : \{u_1,\ldots,u_n\} \to \Sigma$ be the mapping such that $g(u_i) = u_i$, if $u_i = a \in \Sigma$, and $g(u_i) = \nu(x) = a$, if $u_i = (x,a) \in \mathbf{V} \times \Sigma$. Then clearly $f(u_i) = g(u_i)$, for each $1 \leq i \leq n$, and, further, $q_0 \xrightarrow{g(u_1)} q_1 \xrightarrow{g(u_2)} \cdots \xrightarrow{g(u_{n-1})} q_{n-1} \xrightarrow{g(u_n)} q_n$ is an accepting run of $\mathcal{L}(\nu(\mathcal{A}))$. We conclude that $w \in \mathcal{L}(\nu(\mathcal{A}))$.

Assume, on the other hand, that $w \in \mathcal{L}(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$. Suppose that $w = a_1a_2\cdots a_n$, where each $a_i \in \Sigma$ $(1 \leq i \leq n)$, and let $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ be an

accepting run of $\mathcal{L}(\nu(\mathcal{A}))$; i.e. q_n is a final state of $\mathcal{A}$. Assume that $i_1 < i_2 < \cdots < i_m$ are the only indexes in the set $\{0,1,\ldots,n-1\}$ such that, for each $1 \leq j \leq m$, there is no transition labeled a_{i_j+1} from q_{i_j} to q_{i_j+1} in $\mathcal{A}$. Then there must be a transition in $\mathcal{A}$ from q_{i_j} to q_{i_j+1} labeled $x_{i_j} \in \mathbf{V}$. Consider an arbitrary accepting run $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} p_{n-1} \xrightarrow{a_n} p_n$ of $\mathcal{B}$; i.e. p_n is a final state of $\mathcal{B}$. Then it is clear that

$$(q_0,p_0) \xrightarrow{a_1} (q_1,p_1)\cdots(q_{i_1},p_{i_1}) \xrightarrow{(x_{i_1},a_{i_1+1})} (q_{i_1+1},p_{i_1+1})\cdots$$
$$(q_{i_m},p_{i_m}) \xrightarrow{(x_{i_m},a_{i_m+1})} (q_{i_m+1},p_{i_m+1})\cdots(q_{n-1},p_{n-1}) \xrightarrow{a_n} (q_n,p_n)$$

is an accepting run of $\mathcal{A}'$. Thus, $\mathcal{L}(\mathcal{A}') \neq \emptyset$.

2) $\Box$-semantics: Now we deal with the $\Box$-semantics. That MEMBERSHIP$_\Box$ is coNP-hard, even over the class of expressions of star-height 0, follows from the second part of Theorem 5.1.1. Indeed, we constructed there a pattern π whose underlying graph is a DAG and an RPQ of form (x,w,y), where w is a word such that computing the certain answers of Q over π was coNP-hard. The pattern π can be seen as an incomplete automata, and we can transform it into a parameterized regular expression of star height 0 in polynomial time. In fact, such theorem proves something stronger: MEMBERSHIP$_\Box$ is coNP-hard for the class of expressions of star-height 0, even for a fixed word w. Next we prove that MEMBERSHIP$_\Box$ is coNP-hard, even over the class of simple regular expressions.

We use a reduction from 3-SAT to the complement of MEMBERSHIP$_\Box$ over the class of simple expressions. Let $\varphi = \bigwedge_{1\leq i\leq n}(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1,\ldots,p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for $1 \leq k \leq m$. Next, we show how to construct in polynomial time from φ, a simple regular expression e over alphabet $\Sigma = \{a,b,0,1\}$ and a word w over the same alphabet such that φ is satisfiable if and only if $w \notin \mathcal{L}_\Box(e)$.

We start by defining w as the following word, where we have distinguished several prefixes that will be used in the rest of the proof:

$$\underbrace{1111a\,11111a\,b\,1110a\,11110a\,b\,111111a\,1111111a\,b\,111110a\,1111110a\,b\,\cdots}_{w',w'',w_i^+,w_i^-,w_j^u}$$

$$\underbrace{\overbrace{\cdots 1^{2i+1}1a\,1^{2i+2}1a}^{w_i^-}\,b\,1^{2i+1}0a\,1^{2i+2}0a\,b\cdots 1^{2m+1}1a\,1^{2m+2}1a\,b\,1^{2m+1}0a\,1^{2m+2}0a\,b}_{w',w'',w_j^u}$$

$$\underbrace{1^{3(m+1)+1}0a\,1^{3(m+1)+2}0a\,1^{3(m+1)+3}0a\,b\,1^{6(m+1)+1}0a\,1^{6(m+1)+2}0a\,1^{6(m+1)+3}0a\,b\cdots}_{w',w'',w_j^u}$$

$$\underbrace{\overbrace{\cdots 1^{3j(m+1)+1}0a\,1^{3j(m+1)+2}0a\,1^{3j(m+1)+3}0a}^{w_j^u}\,b\cdots}_{w',w''}$$

$$\underbrace{\overbrace{\cdots 1^{3n(m+1)+1}0a\,1^{3n(m+1)+2}0a\,1^{3n(m+1)+3}0a\,b}^{w'}\,b\,aa}_{w''}. \quad (8.2)$$

As it is shown above, we denote by w' the prefix of w such that $w = w'aa$ and by w'' the prefix of w such that $w = w''baa$. Clearly, w can be constructed in polynomial time from φ.

The regular expression e is defined as $(\Sigma^* b \cup \varepsilon) f (b\Sigma^* \cup \varepsilon)$, where f is defined as:

$$\big((f_1 \cup g_1 \cup \cdots f_m \cup g_m)(a \cup \varepsilon)\big)^*.$$

Intuitively f_i (resp. g_i) codifies p_i (resp. $\neg p_i$) and the clauses in which p_i (resp. $\neg p_i$) appears. Formally, we define f_i ($1 \leq i \leq m$) as

$$\Big(\big(\{w'\} \cup \{w''\} \cup 1^{2i+1} \cup \bigcup_{\{1\leq j\leq n | p_i = \ell_j^1\}} 1^{3j(m+1)+1} \cup \bigcup_{\{1\leq j\leq n | p_i = \ell_j^2\}} 1^{3j(m+1)+2} \cup \bigcup_{\{1\leq j\leq n | p_i = \ell_j^3\}} 1^{3j(m+1)+3}\big) \cdot x_i a\Big),$$

where x_i is a fresh variable in $\mathbf{V}$. In the same way we define g_i as

$$\Big(\big(\{w'\} \cup \{w''\} \cup 1^{2i+2} \cup \bigcup_{\{1\leq j\leq n | \neg p_i = \ell_j^1\}} 1^{3j(m+1)+1} \cup \bigcup_{\{1\leq j\leq n | \neg p_i = \ell_j^2\}} 1^{3j(m+1)+2} \cup \bigcup_{\{1\leq j\leq n | \neg p_i = \ell_j^3\}} 1^{3j(m+1)+3}\big) \cdot \bar{x}_i a\Big),$$

where $\bar{x}_i$ is a fresh variable in $\mathbf{V}$. The variable x_i (resp. $\bar{x}_i$) is said to be *associated* with p_i (resp. $\neg p_i$) in e. Clearly, e is a simple regular expression and can be constructed in polynomial time from φ.

Next we show that φ is satisfiable if and only if $w \notin \mathcal{L}_\square(e)$.

We prove first that if $w \notin \mathcal{L}_\square(e)$ then φ is satisfiable. Assume that $w \notin \mathcal{L}_\square(e)$. Then there exists a valuation $\nu : \{x_1, \bar{x}_1, \ldots, x_m, \bar{x}_m\} \to \Sigma$ such that $w \notin \mathcal{L}(\nu(e))$. First of all, we prove

that for each $1 \leq i \leq m$ both $\nu(x_i)$ and $\nu(\bar{x}_i)$ belong to the set $\{0,1\}$. Assume, for the sake of contradiction, that this is not the case. Suppose first that $\nu(x_i) = a$, for some $1 \leq i \leq m$. Then it is clear that $\mathcal{L}(w'aa) \subseteq \mathcal{L}(\nu(e))$ (because $\mathcal{L}(w'\nu(x_i)a) \subseteq \mathcal{L}(\nu(e))$). But $w = w'aa$, and, therefore, $w \in \mathcal{L}(\nu(e))$, which is a contradiction. Suppose now that $\nu(x_i) = b$, for some $1 \leq i \leq m$. Then, again, it is clear that $\mathcal{L}(w''baa) \subseteq \mathcal{L}(\nu(e))$ (because $\mathcal{L}(w''\nu(x_i)aa) \subseteq \mathcal{L}(\nu(e))$). As in the previous case, $w = w''baa$, and, therefore, $w \in \mathcal{L}(\nu(e))$, which is a contradiction. The other case, when $\nu(\bar{x}_i) \in \{a,b\}$, for some $1 \leq i \leq m$, is completely analogous.

Next we prove that for each $1 \leq i \leq m$ it is the case that $\nu(x_i) = 1 - \nu(\bar{x}_i)$. Assume otherwise. Then for some $1 \leq i \leq m$ it is the case that $\nu(x_i) = \nu(\bar{x}_i)$. Suppose first that $\nu(x_i) = \nu(\bar{x}_i) = 1$. Consider the unique prefix w_1 of w that is of the form $u1^{2i+1}1a1^{2i+2}1a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f)(b\Sigma^* \cup \varepsilon))$. It follows that $w_1 \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f))$. But since w_1 is of the form $u1^{2i+1}1a1^{2i+2}1a$, it follows that $u = \varepsilon$ or $u = u'b$, for some $u' \in \Sigma^*$. In any case it must hold that $1^{2i+1}1a1^{2i+2}1a \notin \mathcal{L}(\nu(f))$. Notice, however, that $\mathcal{L}(1^{2i+1}\nu(x_i)a1^{2i+2}\nu(\bar{x}_i)a) \subseteq \mathcal{L}(\nu(f))$. Hence, $1^{2i+1}1a1^{2i+2}1a \in \mathcal{L}(\nu(f))$, which is a contradiction. Suppose, on the other hand, that $\nu(x_i) = \nu(\bar{x}_i) = 0$. Consider the unique prefix w_1 of w that is of the form $u1^{2i+1}0a1^{2i+2}0a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f)(b\Sigma^* \cup \varepsilon))$. It follows that $w_1 \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f))$. But since w_1 is of the form $u1^{2i+1}0a1^{2i+2}0a$, it follows that $u = \varepsilon$ or $u = u'b$, for some $u' \in \Sigma^*$. In any case it must hold that $1^{2i+1}0a1^{2i+2}0a \notin \mathcal{L}(\nu(f))$. Notice, however, that $\mathcal{L}(1^{2i+1}\nu(x_i)a1^{2i+2}\nu(\bar{x}_i)a) \subseteq \mathcal{L}(\nu(f))$. Hence, $1^{2i+1}0a1^{2i+2}0a \in \mathcal{L}(\nu(f))$, which is a contradiction.

We can then define a propositional assignment $\sigma : \{p_1, \ldots, p_m\} \to \{0,1\}$ such that $\sigma(p_i) = \nu(x_i)$, for each $1 \leq i \leq m$. Notice, from our previous remarks, that $\sigma(\neg p_i) = 1 - \nu(x_i) = \nu(\bar{x}_i)$, for each $1 \leq i \leq m$. We prove next that σ satisfies φ. Assume this is not the case. Then for some $1 \leq j \leq n$ it is the case that $\sigma(\ell_j^1) = \sigma(\ell_j^2) = \sigma(\ell_j^3) = 0$. Consider now the unique prefix w_1 of w such that w_1 is of the form $ub1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}(\Sigma^*b\nu(f)b\Sigma^*)$. It follows that $w_1 \notin \mathcal{L}(\Sigma^*b\nu(f))$. But since w_1 is of the form $ub1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a$, it is the case that

$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a \notin \mathcal{L}(\nu(f)).$$

Let q_1, q_2 and q_3 be the variables in e associated with ℓ_j^1, ℓ_j^2 and ℓ_j^3, respectively. Then it cannot be the case that $\nu(q_1) = \nu(q_2) = \nu(q_3) = 0$. Assume otherwise. It is clear that $\mathcal{L}(1^{3j(m+1)+1}\nu(q_1)a1^{3j(m+1)+2}\nu(q_2)a1^{3j(m+1)+3}\nu(q_3)a) \subseteq \mathcal{L}(\nu(f))$, and, therefore,

$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a \in \mathcal{L}(\nu(f)),$$

which is a contradiction. Thus, either $\nu(q_1) = \sigma(\ell_j^1) = 1$ or $\nu(q_2) = \sigma(\ell_j^2) = 1$ or $\nu(q_3) = \sigma(\ell_j^3) = 1$. This is our desired contradiction.

We prove second that if φ is satisfiable then $w \notin \mathcal{L}_{\Box}(e)$. Assume that φ is satisfiable. Then there exists a propositional assignment $\sigma : \{p_1, \ldots, p_m\} \to \{0,1\}$ that satisfies φ. We define a valuation $\nu : \{x_1, \bar{x}_1, \ldots, x_m, \bar{x}_m\} \to \{0,1\}$ for e as follows: For each $1 \leq i \leq m$ it is the case that $\nu(x_i) = \sigma(p_i)$ and $\nu(\bar{x}_i) = 1 - \sigma(p_i)$. We prove next that $w \notin \mathcal{L}(\nu(e))$.

Clearly, $w \notin \mathcal{L}(\nu(e))$ if and only if for each words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1 w_2 w_3$ it is the case that $w_1 \notin \mathcal{L}(\Sigma^* b \cup \varepsilon)$ or $w_2 \notin \mathcal{L}(\nu(f))$ or $w_3 \notin \mathcal{L}(b\Sigma^* \cup \varepsilon)$. Thus, in order to prove that $w \notin \mathcal{L}(\nu(e))$ it is enough to prove that for each words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1 w_2 w_3$,

$$(*) \text{ if } w_1 \in \mathcal{L}(\Sigma^* b \cup \varepsilon) \text{ and } w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon) \text{ then } w_2 \notin \mathcal{L}(\nu(f)).$$

Take arbitrary words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1 w_2 w_3$. We consider several cases:

1. Either $w_1 \notin \mathcal{L}(\Sigma^* b \cup \varepsilon)$ or $w_3 \notin \mathcal{L}(b\Sigma^* \cup \varepsilon)$. Then (*) is trivially true.

2. It is the case that $w_1 \in \mathcal{L}(\Sigma^* b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form $1^{2i+1}1a1^{2i+2}1au$, for some $1 \leq i \leq m$ and $u \in \Sigma^*$. Assume, for the sake of contradiction, that $w_2 \in \mathcal{L}(\nu(f))$. Since clearly there is no word accepted by $\mathcal{L}(\nu(f))$ with prefix baa, it must be the case that w_3 is not the empty word, and, therefore, that $w_3 \in \mathcal{L}(b\Sigma^*)$. Thus, the only possibility for w_2 to belong to $\mathcal{L}(\nu(f))$ is that $1^{2i+1}1a \in \mathcal{L}(\nu(f_i))$ and $1^{2i+2}1a \in \mathcal{L}(\nu(g_i))$. But this can only happen if $\nu(x_i) = 1$ and $\nu(\bar{x}_i) = 1$, which is our desired contradiction (since $\nu(x_i) = 1 - \nu(\bar{x}_i)$).

3. It holds that $w_1 \in \mathcal{L}(\Sigma^* b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form $1^{2i+1}0a1^{2i+2}0au$, for some $1 \leq i \leq m$ and $u \in \Sigma^*$. This case is completely analogous to the previous one.

4. It is the case that $w_1 \in \mathcal{L}(\Sigma^* b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form

$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0au,$$

for some $1 \leq j \leq n$ and $u \in \Sigma^*$. Assume, for the sake of contradiction, that $w_2 \in \mathcal{L}(\nu(f))$. It is easy to see that the only way in which this can happen is that $\nu(q_1) = \nu(q_2) = \nu(q_3) = 0$, where q_1, q_2 and q_3 are the variables in e that are associated with ℓ_j^1, ℓ_j^2 and ℓ_j^3, respectively. Thus, $\sigma(\ell_j^1) = \sigma(\ell_j^2) = \sigma(\ell_j^3) = 0$, which is or desired contradiction.

This finishes the proof of the proposition. □

8.4 Universality

Somewhat curiously, the universality problem is more complex for the possibility semantics $\mathcal{L}_{\Diamond}$, even for parameterized automata. Indeed, consider an incomplete automaton $\mathcal{A}$ over Σ and

$\mathcal{W}$. For the certainty semantics, it suffices to construct the equivalent automata $\mathcal{A}^{fin}$ without edges labelled with expressions denoting an infinite alphabet, and just guess a word w and a valuation such that $w \notin \mathcal{L}(\nu(A^{fin}))$. We have shown in Section 7.3 that all such $\nu(A^{fin})$ are of polynomial size, which results in a PSPACE upper bound for this problem. This is the best that we can do, as the universality problem is PSPACE-hard even for standard regular expressions. On the other hand, a simple adaptation to the proof of Theorem 10 in [Freydenberger, 2013] shows that this problem is undecidable for incomplete automata under possibility semantics. But even if one considers only parameterized automata, in order to solve universality one can expect that all possible valuations for $\mathcal{A}$ will need to be analyzed, which increases the complexity by one exponential. (In fact, when one moves to infinite alphabets, this problem becomes undecidable even for parameterized regular expressions [Grumberg et al., 2010]). The lower bound proof for parameterized expressions is again by a generic reduction.

Theorem 8.4.1

- *The problem* UNIVERSALITY$_\Box$ *is* PSPACE*-complete. It remains* PSPACE*-hard even if restricted to standard regular expressions.*
- *The problem* UNIVERSALITY$_\Diamond$ *is undecidable, and* EXPSPACE*-complete when restricted to parameterized automata. It remains* EXPSPACE*-hard for parameterized regular expressions.*

Proof: We only need to show the EXPSPACE bound of the second part. We begin with the upper bound. It is well known that there is an algorithm to decide whether the language of a standard NFA is universal, that requires polynomial space with respect to the size of the input NFA. Given an incomplete automaton $\mathcal{A}$, we construct the equivalent NFA $\mathcal{B}$ such that $L(\mathcal{B}) = \mathcal{L}_\Diamond(\mathcal{A})$, and then decide universality of $L(\mathcal{B})$. Since the NFA $\mathcal{B}$ is of size exponential with respect to the original incomplete automaton $\mathcal{A}$, the above procedure runs in EXPSPACE.

The lower bound is again shown for parametrized regular expressions. We present a reduction from the complement of the acceptance problem of a Turing machine. Let L be a language that belongs to EXPSPACE, and let $\mathcal{M}$ be a Turing machine that decides L in EXPSPACE. Given an input $\bar{a}$ we construct in polynomial time with respect to $\mathcal{M}$ and $\bar{a}$ a parameterized regular expression $e_{\mathcal{M},\bar{a}}$ over some alphabet Δ such that $\mathcal{L}_\Diamond(e_{\mathcal{M},\bar{a}})$ consists of all the strings over Δ if and only if $\mathcal{M}$ does not accept input $\bar{a}$.

Just as in section 8.2, we assume that $\mathcal{M} = (Q, \Gamma, q_0, \{q_m\}, \delta)$, where $Q = \{q_0, \ldots, q_m\}$ is the set of states, Γ is the tape alphabet (that contains the distinguished *blank* symbol B), the initial state is q_0, q_m is the unique final state, and $\delta : (Q \setminus \{q_m\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Notice that we assume without loss of generality that no transition is defined on the final state q_m. 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 Γ). For notational convenience we will assume from now on that $S(|\bar{a}|) = n$.

We also find it convenient to introduce the following notation. For an alphabet $\Sigma = \{b_1, \ldots, b_p\}$ of symbols, we abuse notation and denote by Σ the regular expression $(b_1 \mid \cdots \mid b_p)$. Thus, for example, assume that $\Gamma = \{b_1, \ldots, b_p\} \cup \{B\}$. Then, when we write $(\Gamma \cup (\Gamma \times Q))$ we represent the language given by $(b_1 \mid \cdots \mid b_p \mid B \mid (b_1, q_0) \mid \cdots \mid (B, q_m))$. Furthermore, we reuse the notation introduced in Section 8.2, and write the shorthand $[i]$ to denote the binary representation of the number i as a string of n characters (i.e., $[0]$ corresponds to the word 0^n, and $[2]$ corresponds to the word $0^{n-2}10$).

Our parameterized expression $e_{\mathcal{M},\bar{a}}$ uses the alphabet $\Delta = \{0, 1, \#, \&, \%\} \cup \Gamma \cup (\Gamma \times Q)$. The idea of the reduction is as follows. Using Δ, we represent a configuration of $\mathcal{M}$ by words in Δ^* of the form:

$$\begin{aligned} \#\cdot[0]\cdot(\Gamma\cup(\Gamma\times Q))\cdot\&\cdot & \\ [1]\cdot(\Gamma\cup(\Gamma\times Q))\cdot\&\cdot & \\ [2]\cdot(\Gamma\cup(\Gamma\times Q))\cdot\& & \\ \vdots & \\ [2^n-1]\cdot(\Gamma\cup(\Gamma\times Q))\cdot\&\cdot\% & \end{aligned} \tag{8.3}$$

Intuitively, the strings $[0], [1], \ldots, [2^n - 1]$ indicate each one of the 2^n cells of $\mathcal{M}$, and the symbol following these strings represents either the content of the cell, or the content of the cell plus the state of $\mathcal{M}$, if $\mathcal{M}$ is pointing into that particular cell of the tape in the configuration that is being encoded.

Since each word of the form (8.3) represents a configuration of $\mathcal{M}$, we can represent a run of $\mathcal{M}$ on input $\bar{a}$ as a sequence of concatenations of words of the form (8.3), as long as each one of these configurations is consistent with the computation of $\mathcal{M}$. The idea of the reduction is to construct a parameterized regular expression $e_{\mathcal{M},\bar{a}}$ that represent all words w in Δ^* that are either *not* valid concatenations of subwords of the form (8.3), or, in case that they are, that the sequence of configurations represented by w is not a valid run of $\mathcal{M}$ on input $\bar{a}$. In other words, any word in Δ^* that does not belong to $\mathcal{L}_\diamond(e_{\mathcal{M},\bar{a}})$ is bound to represent a valid run of $\mathcal{M}$ over input $\bar{a}$, thus obtaining that $\mathcal{L}_\diamond(e) \neq \Delta^*$ if and only if $\mathcal{M}$ accepts on input $\bar{a}$.

We split the definition of $e_{\mathcal{M},\bar{a}}$ into five parts: $e_{\mathcal{M},\bar{a}} = e^1 \mid e^2 \mid e^3 \mid e^4 \mid e^5$, where:

- e^1 describes all the words that are not concatenations of subwords of form (8.3).
- e^2 describes all the words that, even if they are concatenations of subwords of form (8.3), these subwords do not represent valid configurations of $\mathcal{M}$.
- e^3 describes words that do not start with a subword of form (8.3) correctly describing the initial configuration of $\mathcal{M}$ over input $\bar{a}$.

- e^4 describes words whose final subword of form (8.3) does not contain any final states (and is therefore not a final configuration of $\mathcal{M}$).
- e^5 describes words that contains two consecutive subwords of form (8.3) that represent configurations α and β for $\mathcal{M}$ such that α and β do not agree on δ.

Next we show how to construct expressions e^1, e^2, e^3, e^4, e^5. We do not provide the precise details of e^1, since it is straightforward to define it from (8.3). Expression e^2 is defined as the union of the following two expressions, stating that:

- Between a symbol # and % there is no symbol in $(\Gamma \times Q)$ (in other words, the machine is pointing at none of the cells in that configuration):

$$e_1^2 = \Delta^* \cdot \# \cdot (\Delta \setminus (\{\%\} \cup \Gamma \times Q))^* \cdot \% \cdot \Delta^*$$

- Between a symbol # and % there is more than one symbol in $(\Gamma \times Q)$ (a configuration features two positions being read by the machine):

$$e_2^2 = \Delta^* \cdot \# \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times Q) \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times Q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \Delta^*$$

Expression e^3 is the union of the following expressions, describing that:

- The first configuration does not contain the initial state in the first position of the tape, reading the first symbol of the output:

$$e_1^3 = \# \cdot [0] \cdot \Gamma \cup \big((\Gamma \times Q) \setminus \{(q_0, a_0)\}\big) \cdot \Delta^*$$

- The rest of the $k-1$ symbols of the tape do not agree with the input:

$$\begin{aligned} e_2^3 &= \# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot [1] \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{a_1\}) \cdot \Delta^* \\ \vdots &= \vdots \\ e_k^3 &= \# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [k-1] \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{a_{k-1}\}) \cdot \Delta^* \end{aligned}$$

- The rest of the symbols of the first configuration are not blank symbols:

$$e_{k+1}^3 = \# \cdot [0] \cdot \Delta \cdot \& \cdots [k-1] \cdot \Delta \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot (0 \mid 1)^n \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{B\}) \cdot \Delta^*$$

Expression e^4 describes words whose final configuration is not in a final state:

$$e^4 = \Delta^* \cdot \# \cdot (\Delta \setminus \{\%\})^* \cdot \big(\Gamma \times (Q \setminus \{q_m\})\big) \cdot (\Delta \setminus \{\%\})^* \cdot \%$$

Finally, we describe expression e^5. Intuitively, it describes words that feature two subsequent configurations that are not consistent with each other. More precisely, it is the union of the following expressions, stating that:

- A cell not pointed by the head changed its content from one configuration to the subsequent one:

$$e_1^5 = \bigcup_{a\in\Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

- A configuration that is not final features a pair in $Q \times \Sigma$ for which no transition is defined (the symbol # states the configuration is not the final one):

$$e_2^5 = \bigcup_{\{(a,q) \mid \delta(q,a) \text{ is not defined}\}} \Delta^* \cdot (a,q) \cdot \Delta^* \cdot \# \cdot \Delta^*$$

- The change of state does not agree with δ:

$$e_3^5 = \bigcup_{\{(a,q) \mid \delta(q,a)=(q',a',\{L,R\})\}} \Delta^* \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times (Q \setminus \{q'\})) \cdot \Delta^*$$

- The symbol written in a given step does not agree with δ:

$$e_4^5 = \bigcup_{\{(a,q) \mid \delta(q,a)=(q',a',\{L,R\})\}} \Delta^* \cdot y_1 \cdots y_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ (\Delta \setminus \{\%\})^* \cdot y_1 \cdots y_n \cdot (\Gamma \setminus \{a'\}) \cdot \Delta^*$$

- The movement of the head does not agree with δ:

$$e_5^5 = \bigcup_{\{(a,q) \mid \delta(q,a)=(q',a',R)\}} \Delta^* \cdot z_1 \cdots z_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ (\Delta \setminus \{\%\})^* \cdot z_1 \cdots z_n \cdot a' \cdot \& \cdot \big(\varepsilon \mid ((0|1)^n \cdot \Gamma \cdot (\Delta \setminus \{\%\}))^*\big) \cdot \% \cdot \Delta^*$$

$$e_6^5 = \bigcup_{(a,q) \mid \delta(q,a)=(q',a',L)} \Delta^* \cdot w_1 \cdots w_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot \big(\varepsilon \mid ((\Delta \setminus \{\%\})^* \cdot (0|1)^n \cdot \Gamma \cdot \&)\big) \cdot w_1 \cdots w_n \cdot \Delta^*$$

Having defined $e_{\mathcal{M},\bar{a}}$, it is now straightforward to show that $\mathcal{L}_\diamond(e_{\mathcal{M},\bar{a}}) = \Delta^*$ if and only if $\mathcal{M}$ does not accept on input $\bar{a}$. This finishes the proof of the EXPSPACE lower bound. □

One can also show that the EXPSPACE bound for UNIVERSALITY$_\diamond$ holds even for simple expressions (note that it makes no sense to study expressions of star-height 0, as they denote finite languages and thus cannot be universal).

Proposition 8.4.2 *The problem* UNIVERSALITY$_\diamond$ *remains* EXPSPACE*-hard over the class of simple parameterized regular expressions.*

Proof: We sketch how to adapt the reduction of Theorem 8.4.1 to hold for simple parameterized regular expressions (i.e. without repetitions of variables).

Recall that the previous reduction used the alphabet $\{0,1,\%,\#,\&\} \cup \Gamma \cup (\Gamma \times Q)$. In this case, we need a slightly bigger alphabet. Let $\Delta = \{0,1,\&,\#_{\text{even}},\%_{\text{even}},\#_{\text{odd}},\%_{\text{odd}}\} \cup \Gamma \cup (\Gamma \times Q)$. The idea is to modify the way configurations are represented.

Previously, we had that runs of $\mathcal{M}$ were represented by words in the language:

$$\big(\# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%\big)^*.$$

We modify the coding, so that configurations are represented in the following way:

$$\big(\#_{\text{even}} \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%_{\text{even}} \cdot \\ \#_{\text{odd}} \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%_{\text{odd}}\big)^*$$

The intuition is that configurations using $\#_{\text{even}}$ and $\%_{\text{even}}$ represent an even step of the computation of the Turing machine, whereas configurations using $\#_{\text{odd}}$ and $\%_{\text{odd}}$ represent an odd step. Notice that one can assume, without loss of generality, that the run of $\mathcal{M}$ over input $\bar{a}$ ends after an odd number of computations.

All that remain to do is to adapt the definition of the expression $e_{\mathcal{M},\bar{a}} = e^1 \mid \cdots \mid e^5$ so that it works under this modified coding, and such that $e_{\mathcal{M},\bar{a}}$ is simple. We omit most of the details, since most of the expressions in $e^1,\ldots,e^5$ do not use parameters, and thus are not difficult to modify.

To see how the expressions using parameters can be modified so that they are simple, we show how to adapt the expression e^5_1, that intuitively accepts all words describing two configurations in which a cell not pointed by the head changed its content. It was defined previously as

$$e^5_1 = \bigcup_{a \in \Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

A straightforward idea is to explicitly state even - odd and odd - even cases, that is, redefine e^5_1 as $e^5_{1,e} \mid e^5_{1,o}$, where

$$e^5_{1,e} = \bigcup_{a \in \Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}},\%_{\text{odd}}\})^* \cdot \%_{\text{even}} \cdot \\ \#_{\text{odd}} \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

$$e^5_{1,o} = \bigcup_{a\in\Gamma} \Delta^* \cdot y_1 \cdots y_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}}, \%_{\text{odd}}\})^* \cdot \%_{\text{odd}} \cdot$$
$$\#_{\text{even}} \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot y_1 \cdots y_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

The problem is that these expressions are not simple: they reuse the variables $x_1, \ldots, x_n$ or $y_1, \ldots, y_n$. The solution is instead a bit more technical. We redefine e^5_1 as $e^6_{1,e} \mid e^6_{1,o}$, where:

$$e^6_{1,e} = \bigcup_{a\in\Gamma} \Delta^* \#_{\text{even}} \cdot (\Delta \setminus \{\%_{\text{even}}\})^* \cdot$$
$$\Bigg(x_1 \cdots x_n \cdot \Bigg(\big(a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}}\})^* \cdot \%_{\text{even}} \cdot \#_{\text{odd}} (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot \&\big) \mid$$
$$\big(((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)) \cdot (\Delta \setminus \{\%_{\text{odd}}, \#_{\text{even}}, \%_{\text{even}}\})^*\big)\Bigg)\Bigg)^* \cdot \%_{\text{odd}} \cdot \Delta^*$$

$$e^6_{1,o} = \bigcup_{a\in\Gamma} \Delta^* \#_{\text{odd}} \cdot (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot$$
$$\Bigg(y_1 \cdots y_n \Bigg(\big(a \cdot \& \cdot (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot \%_{\text{odd}} \cdot \#_{\text{even}} (\Delta \setminus \{\#_{\text{even}}\})^* \cdot \&\big) \mid$$
$$\big(((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)) \cdot (\Delta \setminus \{\%_{\text{even}}, \#_{\text{odd}}, \%_{\text{odd}}\})^*\big)\Bigg)\Bigg)^* \cdot \%_{\text{even}} \cdot \Delta^*$$

Notice then that $e^6_{1,e} \mid e^6_{1,o}$ is a simple parameterized regular expression. In order to see that the intended meaning of these expressions remains the same, notice that $\mathcal{L}_\diamond(e^5_{1,e}) \subseteq \mathcal{L}_\diamond(e^6_{1,e})$ and $\mathcal{L}_\diamond(e^5_{1,o}) \subseteq \mathcal{L}_\diamond(e^6_{1,o})$. Moreover, it is not difficult to check that none of the words that belong to $\mathcal{L}_\diamond(e^6_{1,e})$ but not to $\mathcal{L}_\diamond(e^5_{1,e})$ represent a valid run of $\mathcal{M}$, and neither does any word in $\mathcal{L}_\diamond(e^6_{1,o})$ but not in $\mathcal{L}_\diamond(e^5_{1,o})$. Thus, the words in $\mathcal{L}_\diamond(e^6_{1,e})$ but not in $\mathcal{L}_\diamond(e^5_{1,e})$ (respectively, in $\mathcal{L}_\diamond(e^6_{1,o})$ but not in $\mathcal{L}_\diamond(e^5_{1,o})$) are not harmful for our purposes, since these extra words already belong to the language of some other disjunction in $e_{\mathcal{M},\bar{a}}$.

With these observations, it is not difficult to modify the remainder of the reduction of the proof of Theorem 8.4.1 so that every expression is simple. The proof then follows along the same lines as the proof of Theorem 8.4.1. □

8.5 Containment

The bounds for the containment problem are easily obtained from the fact that both nonemptiness and universality can be cast as its versions. That is, we have:

Theorem 8.5.1

- *The problem* CONTAINMENT$_\square$ *is* EXPSPACE*-complete. It remains* EXPSPACE*-hard when restricted to parameterized regular expressions.*
- *The problem* CONTAINMENT$_\diamond$ *is undecidable. It is* EXPSPACE*-complete when restricted to parameterized automata, and remains* EXPSPACE*-hard when restricted to parameterized regular expressions.*

Proof: Since $\Sigma^* \subseteq \mathcal{L}_\diamond(\mathcal{A})$ iff UNIVERSALITY$_\diamond(\mathcal{A})$ is true, and $\mathcal{L}_\square(\mathcal{A}) \subseteq \emptyset$ iff NONEMPTINESS$_\square(\mathcal{A})$ is false, we get EXPSPACE-hardness and undecidability from Theorems 8.2.1 and 8.4.1.

To check whether $\mathcal{L}_\square(\mathcal{A}_1) \subseteq \mathcal{L}_\square(\mathcal{A}_2)$, we focus once again in automata without transitions labelled with infinite alphabets $\mathcal{A}_1^{fin}$ and $\mathcal{A}_2^{fin}$. We must check that $\bigcap_\nu \mathcal{L}(\nu(\mathcal{A}_1^{fin})) \cap \overline{\mathcal{L}(\nu'(\mathcal{A}_2^{fin}))} = \emptyset$ for each valuation ν' on $\mathcal{A}_2^{fin}$. This is doable in EXPSPACE, since one can construct exponentially many automata for $\mathcal{L}(\nu(A_1^{fin}))$ in EXPTIME, as well as the automaton for the complement $\overline{\mathcal{L}(\nu'(\mathcal{A}_2^{fin}))}$, and checking nonemptiness of the intersection of those is done in polynomial space in terms of their size, i.e., in EXPSPACE. Since this needs to be done for exponentially many valuations ν', the overall EXPSPACE bound follows. The proof for the $\mathcal{L}_\diamond$ semantics is similar, having in mind that we restrict once again to parameterized automata $\square$

Containment with one fixed expression. We look at two variations of the containment problem, when one of the automata is fixed: CONTAINMENT$_*(\mathcal{A}_1, \cdot)$ asks, for an incomplete automaton $\mathcal{A}_2$, whether $\mathcal{L}_*(\mathcal{A}_1) \subseteq \mathcal{L}_*(\mathcal{A}_2)$; and CONTAINMENT$_*(\cdot, \mathcal{A}_2)$ is defined similarly. The reductions proving Theorem 8.5.1 show that CONTAINMENT$_\square(\cdot, \mathcal{A}_2)$ and CONTAINMENT$_\diamond(\mathcal{A}_1, \cdot)$ remain EXPSPACE-hard and undecidable, respectively. For the other two versions of the problem, the proposition below shows that the complexity is lowered by at least one exponential. For CONTAINMENT$_\diamond(\cdot, \mathcal{A}_2)$ we focus solely on parameterized automata. The complexity for the general case when the input is an arbitrary incomplete automaton is left open.

Proposition 8.5.2
- CONTAINMENT$_\square(\mathcal{A}_1, \cdot)$ *is* PSPACE*-complete.*
- CONTAINMENT$_\diamond(\cdot, \mathcal{A}_2)$ *is* coNP*-complete when restricted to parameterized automata.*

Proof: For **Part 1**, it is well known that CONTAINMENT$_\square(\mathcal{A}_1, \cdot)$ is PSPACE-hard even for standard NFA or regular expressions. For the upper bound, let $\mathcal{A}_1'$ be an NFA such that $L(\mathcal{A}_1') = \mathcal{L}_\square(\mathcal{A}_1)$. Since $\mathcal{A}_1$ is fixed, the expression $\mathcal{A}_1'$ can be computed in constant time. Then, it suffices to guess a valuation ν and a word w such that $w \in L(\mathcal{A}_1')$, but $w \notin \mathcal{L}(\nu(\mathcal{A}_2))$, which can clearly be done in PSPACE by considering the equivalent automata $\mathcal{A}_2^{fin}$ without transitions labelled with infinite languages (since we know that all $\nu(\mathcal{A}_2^{fin})$ are of polynomial size).

For **Part 2**, we begin with the upper bound for the problem CONTAINMENT$_\diamond(\cdot, \mathcal{A}_2)$. Assume that the input is a parameterized automata $\mathcal{A}_1$ over Σ, and that $\mathcal{W} \subset \mathbf{V}$ is the set of variables mentioned in $\mathcal{A}_1$. The following coNP algorithm solves the problem CONTAINMENT$_\diamond(\cdot, \mathcal{A}_2)$. First, construct an NFA $\mathcal{A}_2'$ such that $L(\mathcal{A}_2') = \mathcal{L}_\diamond(\mathcal{A}_2)$, and then construct $\mathcal{A}_2^C$, the NFA that accepts the complement of $L(\mathcal{A}_2')$. Since $\mathcal{A}_2$ is fixed, $\mathcal{A}_2^C$ can be constructed in constant time. Next, guess a valuation $\nu : \mathcal{W} \to \Sigma$, and, check that $\nu(\mathcal{A}_1) \cap \mathcal{A}_2^C \neq \emptyset$, which can be performed in polynomial time using a standard reachability test over the product of $\nu(\mathcal{A}_1)$ and $\mathcal{A}_2^C$. It is not hard to see that this algorithm is sound and complete for the problem.

In fact, if $\nu(\mathcal{A}_1) \cap \mathcal{A}_2^C \neq \emptyset$, then there is a word $w \in \mathcal{L}(\nu(\mathcal{A}_1'))$, and hence in $\mathcal{L}_\diamond(\mathcal{A}_1)$, that does not belong to $\mathcal{L}_\diamond(\mathcal{A}_2)$. This implies that $\mathcal{L}_\diamond(\mathcal{A}_1)$ is not contained in $\mathcal{L}_\diamond(\mathcal{A}_2)$. On the other hand, it is clear that if $\nu(\mathcal{A}_1) \cap \mathcal{A}_2^C \neq \emptyset$ for all possible valuations ν from $\mathcal{W}$ to Σ, then $\mathcal{L}_\diamond(\mathcal{A}_1)$ is contained in $\mathcal{L}_\diamond(\mathcal{A}_2)$.

The lower bound is given again for the case when the input are parameterized regular expressions. It is established via a reduction from 3-SAT to the complement of $\text{CONTAINMENT}_\square(\cdot, e_2)$, where e_2 is the following regular expression over the alphabet $\Sigma = \{0,1,\#\}$:

$$e_2 := \big((10 \mid 01)^*\#((0 \mid 1)^3)^*000((0 \mid 1)^3)^*\big) \mid \big(((0 \mid 1)^2)^*(00 \mid 11)\Sigma^*\big) \mid \big(\Sigma^*\#\Sigma^*\#\Sigma^*\big).$$

Notice that e_2 mentions no variables, and hence $\mathcal{L}_\diamond(e_2) = \mathcal{L}(e_2)$.

Let $\varphi = \bigwedge_{1 \leq i \leq n}(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1, \ldots, p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for $1 \leq k \leq m$. Next we show how to construct in polynomial time from φ a parameterized regular expression e_1 over the alphabet $\Sigma = \{0,1,\#\}$ such that φ is satisfiable if and only if $\mathcal{L}_\diamond(e_1) \not\subseteq \mathcal{L}(e_2)$.

Let $\mathcal{W} = \{x_i, \hat{x}_i \mid 1 \leq i \leq m\}$. Intuitively, each x_i represents the value assigned to p_i, and $\hat{x}_i$ represents the value of $\neg p_i$. Moreover, assume that h is a mapping from the literals ℓ_i^j ($1 \leq i \leq n$ and $1 \leq j \leq 3$) to $\mathcal{W}$, defined as expected: $h(\ell_i^j) = x_k$ if ℓ_i^j is p_k, for some $1 \leq k \leq m$, and $h(\ell_i^j) = \hat{x}_k$ if ℓ_i^j is $\neg p_k$.

Define e_1 as follows:

$$e_1 = x_1\hat{x}_1 \cdots x_m\hat{x}_m\#h(\ell_1^1)h(\ell_1^2)h(\ell_1^3)\cdots h(\ell_n^1)h(\ell_n^2)h(\ell_n^3).$$

We show that φ is satisfiable if and only if $\mathcal{L}_\diamond(e_1) \not\subseteq \mathcal{L}(e_2)$.

($\Rightarrow$): Assume that φ is satisfiable by valuation σ. Let ν be a valuation from $\mathcal{W}$ to Σ, defined as follows:

- For each $1 \leq k \leq m$, $\nu(x_k) = 1$ if $\sigma(p_k) = 1$, and $\nu(x_k) = 0$ otherwise.
- For each $1 \leq k \leq m$, $\nu(\hat{x}_k) = 0$ if $\sigma(p_k) = 1$, and $\nu(\hat{x}_k) = 1$ otherwise.

Notice that $L(\nu(e_1))$ consists of the single word:

$$\nu(x_1)\nu(\hat{x}_1)\cdots\nu(x_m)\nu(\hat{x}_m)\#\nu(h(\ell_1^1))\nu(h(\ell_1^2))\nu(h(\ell_1^3))\cdots\nu(h(\ell_n^1))\nu(h(\ell_n^2))\nu(h(\ell_n^3)).$$

We shall abuse notation and denote by $\nu(e_1)$ both this word and the aforementioned expression. It is clear that $\nu(e_1)$ contains a single symbol #, and starts with a prefix in $(01 \mid 10)^*\#$. Thus, if $\mathcal{L}_\diamond(e_1) \subseteq \mathcal{L}(e_2)$ it must be that $\nu(e_1)$ is defined by the expression $(10 \mid 01)^*\#((0 \mid 1)^3)^*000((0 \mid 1)^3)^*$. But this implies that there are literals ℓ_i^1, ℓ_i^2 and ℓ_i^3, for some $1 \leq i \leq n$, such that $\nu(h(\ell_i^1))\nu(h(\ell_i^2))\nu(h(\ell_i^3)) = 000$. By construction of ν, it must

be the case that σ falsifies the i-th clause of φ, which contradicts the fact that σ is a satisfying assignment.

($\Leftarrow$): Assume on the other hand that $\mathcal{L}_{\diamond}(e_1) \not\subseteq \mathcal{L}(e_2)$. From the definition of the $\diamond$-semantics, there is at least one valuation ν from $\mathcal{W}$ to Σ such that $\mathcal{L}(\nu(e_1)) \not\subseteq \mathcal{L}(e_2)$. Notice again that, by construction of e_1, $\nu(e_1)$ consists of the single word:

$$\nu(x_1)\nu(\hat{x}_1)\cdots\nu(x_m)\nu(\hat{x}_m)\#\nu(h(\ell_1^1))\nu(h(\ell_1^2))\nu(h(\ell_1^3))\cdots\nu(h(\ell_n^1))\nu(h(\ell_n^2))\nu(h(\ell_n^3)).$$

Again, we shall denote this word also by $\nu(e_1)$. Then if $\mathcal{L}(\nu(e_1)) \not\subseteq \mathcal{L}(e_2)$ it must be the case that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$. This immediately entails that $\nu(e_1)$ cannot have two or more copies of the symbol #, and thus we conclude that ν assigns to each variable $\mathcal{W}$ a symbol in $\{0,1\}$. From this it follows that the following valuation σ for the variables in φ is well defined:

- $\sigma(p_i) = 1$ if $\nu(x_i) = 1$, and $\sigma(p_i) = 0$ if $\nu(x_i) = 0$

Next we show that for each $1 \leq i \leq m$, it is the case that $\nu(x_i) \neq \nu(\hat{x}_i)$. Assume for the sake of contradiction that for some $i \leq i \leq m$ we have $\nu(x_i) = \nu(\hat{x}_i)$. From the construction of e_1 it must be the case that $\nu(e_1)$ is denoted by the expression $((0 \mid 1)^2)^*(00 \mid 11)\Sigma^*$, which contradicts the fact that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$. Finally, we claim that φ is satisfiable by the valuation σ. Assume the contrary. Then there is a clause $(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$, for $1 \leq i \leq n$, such that, for each $1 \leq j \leq 3$, if ℓ_i^j is the literal p_k, for some $1 \leq k \leq m$, then σ assigns the value 0 to p_k, and if ℓ_i^j is the literal $\neg p_k$, for some $1 \leq k \leq m$, then σ assigns the value 1 to p_k. It is now straightforward to conclude that this fact contradicts the assumption that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$, by studying all of the 8 possible cases.

□

8.6 Intersection With a Regular Language

This problem is a natural analog of the standard decision problem solved in automata-based verification; we also saw in the introduction that it arises when one computes certain paths for queries over graph patterns.

Checking whether $L \cap \mathcal{L}_{\Box}(\mathcal{A}) \neq \emptyset$ can be done in EXPSPACE using the same brute-force algorithm as for the nonemptiness problem (intersection of exponentially many regular languages). Since the nonemptiness problem is a special case with $L = \Sigma^*$, we get the matching lower bound by Theorem 8.2.1. For possibility semantics, languages need not be regular, and thus to maintain regularity we only focus on parameterized automata. For $\mathcal{L}_{\diamond}(\mathcal{A})$, an NP upper bound is easy: one just guesses a valuation so that $L \cap \mathcal{L}(\nu(\mathcal{A})) \neq \emptyset$. If L denotes a single word w, we have an instance of the membership problem, and hence there is a matching lower bound, by Theorem 8.3.1. Summing up, we have:

Corollary 8.6.1

- *The problem* NONEMPTYINTREG$_\Box$ *is* EXPSPACE*-complete.*
- *The problem* NONEMPTYINTREG$_\Diamond$ *is* NP*-complete when restricted to parameterized automata.*

Chapter 9

Conclusions and Future Work

The work in this dissertation is relevant to graph-based applications that need to deal with patterns. We have studied structural properties of graph patterns, how to query them, how to use them as queries, and we have also studied several of their applications. We looked at three main features of patterns: node variables, label variables, and regular expressions specifying paths, and showed that each of these features strictly increases the expressiveness of patters. Moreover, we studied the problem of querying graph patterns, and looked at data and combined complexity of CRPQs and other queries (both extensions and restrictions of CRPQs) over graph patterns. This line of work has direct applications in scenarios where the need for querying patterns is present, such as social networks, online retailing, search result classification, crime detection and plagiarism detection [Fan et al., 2010b, Fan et al., 2010a, Natarajan, 2000, Kanza et al., 2002], to name a few.

The main conclusion is that, without carefully chosen restrictions, querying graph patterns is computationally harder than querying relational or XML patterns. This of course has implications on ongoing work in applications that need to query patterns. For example, we took them into account when studying schema mappings and data exchange in Chapter 6. We have also identified rather robust classes with tractable query answering, as well as classes of reasonable combined complexity for which query answering is naturally viewed as a constraint satisfaction property.

Based on our work on graph patterns, we have studied interoperability issues for graph databases, and shown how patterns allow for a natural definition of different classes of schema mappings. We took into account the existence of scenarios in which the size of the graph forbids any algorithm that is not linear - or at most quadratic - in terms of the database. For these applications it is not advisable to define mappings in terms of patterns, but if one focuses instead on binary queries, such as NPQs, then it is still possible to perform data exchange and query answering in such restricted complexity bounds.

Finally, we defined and studied incomplete automata, a model of automata with applica-

tions not only in graph databases, but in many other areas linked to databases or verification, such as program analysis. Regarding graph databases, we have shown that this model of automata can indeed capture query answering over graph patterns, both for queries returning nodes and queries returning paths. We also studied the basic properties of the languages generated by incomplete automata, such as regularity, emptiness, membership and universality. We believe that this is just the start in this direction, and that the model of incomplete automata provides an engaging new area for future research.

9.1 Future Work

The work in this dissertation opens up many interesting new ideas for research, and several questions remain unanswered.

Patterns. We have formalized our graph patterns in terms of node variables, label variables, and regular expressions; the latter in order to specify paths between nodes. Our features were motivated by real graph databases scenarios, but in the end a graph pattern is just a concise representation of an infinite set of graph databases, and these features are of course not the only possibilities to specify them. However, other areas different from databases may need different specifications for patterns. For example, in areas such as verification one is interested in properties that cannot be expressed by regular expressions, but can be defined with other formalisms, such as modal logics. Examples of these are the existence of loops (so call liveness properties) or multiple recursion (to specify, for example, that a certain part of the graph has the shape of a finite but unbounded tree).

It would be interesting to look at the behavior of other types of graph patterns, defined perhaps using different formalisms to specify paths between nodes; and study the applications of this work in the context of verification of transition systems. It would also be interesting to study the more expressive models of patterns that result of adding extra features to patterns, apart from the three features studied in this dissertation. Here a natural choice is to add *data values* to the nodes –or edges– of patterns, as what has been done, for example, for patterns in XML databases [Barceló et al., 2010b] or XML data exchange [Arenas and Libkin, 2008, Amano et al., 2009].

Another possible extension of patterns is to add features capable of expressing *negation*. The patterns that we have defined are positive, in the sense that they can only specify that some node, label or path must exist in the graphs represented by the pattern. However, one could also specify that a certain feature does not exist in these graphs. Of course, adding unrestricted negation would inevitably lead to the intractability of query answering, since the problem becomes undecidable already for relational conjunctive queries with negation [Libkin, 2004]. But nothing is telling us that we cannot find milder forms of negation which will not significantly

increase the complexity of query answering. For instance, in relational and XML scenarios one can add limited atomic negation [Arenas et al., 2011a] or evaluate boolean combinations of patterns [Gheerbrant et al., 2012, David et al., 2013] without any major computational expense.

On the other hand, the problems one encounters with certain answers and patterns with negation are deeply related with the semantics of patterns. For instance, it was shown in [Gheerbrant et al., 2013] that, for relational databases, if one restrict to different versions of *closed world semantics*, then one can perform naive evaluation for positive queries with universal –as well as existential– quantification, and even for more expressive queries. This means that the certain answers of this class of queries can be computed in polynomial time, contrasting what is known for the default *open world semantics* that we have chosen in this dissertation. It would thus be interesting to study these forms of closed world semantics in the context of graph patterns. This would yield immediate results for answering queries with universal quantification over patterns in $\mathcal{P}^{\text{nv}}$, which are essentially relational naive tables, but one might also find better bounds for patterns using label variables, or regular expressions.

Other applications of patterns. When studying how graph patterns represent data we have so far assumed that no other information is known about the graphs that are represented by these patterns. A natural continuation of our work is to study the case when the representation is given in form of a pattern, plus a set of additional *constraints*. If the constraints are not chosen carefully, we could easily face undecidability of query answering. But the field of description logics (DLs) has successfully identified a good number of logics to specify constraints that have good query answering properties, such as those in the DL-LITE family [Calvanese et al., 2007]. These families of logics are a good starting point to study how to query graph patterns under additional constraints, and indeed this problem has already been studied in [Calvanese et al., 2011b] for the case when patterns are just RPQs. Furthermore, [Arenas et al., 2011b] has already studied the problem of schema mapping and data exchange using instances represented by DL constraints. It would be interesting to see how can these results be applied in the context of graph data exchange under DL constraints.

Another possible application domain for graph patterns are workflow models such as *electronic services* or business processes (see [Hull et al., 2003] for a good theoretical introduction on the subject). While generally these are modelled with Petri nets, one might use graph patterns, or an adaptation thereof, to represent services that are incomplete or unknown; or perhaps as a query language, to certify that a workflow satisfy certain criteria. Alternatively, one could study which properties of e-services can be modelled by graph databases (for instance, provenance [Acar et al., 2010, Belhajjame et al., 2011]), and then naturally apply patterns in this context.

Query answering. With respect to querying graph patterns, a natural continuation of this

work is to carry on with the task of finding tractable fragments for query answering, at least in data complexity. There are two possible directions in this research. The most obvious one is to continue the search for tractable restrictions for the general problem of querying graph patterns. Due to the connection between query answering and constraint satisfaction that we presented in Chapter 5, this line of research is also of interest in that field, since identifying tractable fragments for query answering might transfer in different, unknown tractable restrictions for the constraint satisfaction problem.

But one could also analyze the tractability of query answering specifically from the point of view of applications that need to do query graph patterns. For example, it deserves to be studied how to link our positive results for graph patterns to the context of computing mapping-based certain answers, that is, deciding CERTAIN-DE$_{\mathcal{M}}(Q, G_{\mathbf{S}})$. One trivial way of doing so, is to just consider the class of all mappings $\mathcal{M}$ and graph databases $G_{\mathbf{S}}$, such that the universal representative $\pi_{\mathbf{T}}$ for $G_{\mathbf{S}}$ under $\mathcal{M}$ constructed in the proof of Proposition 6.4.3, belongs to the one of the tractable classes for query answering identified in Chapter 5. Nevertheless, it would be fairly more interesting to search for natural structural conditions imposed over mappings and graph databases, that ensures that the universal representative $\pi_{\mathbf{T}}$ always belong to this class. We believe this to be an interesting and challenging topic for future research.

Finally, we would like to continue with the study of the complexity of query answering under the notion of *parameterized complexity* [Downey and Fellows, 1999, Flum and Grohe, 2006]. We have already shown results for fixed parameter tractability using bounded treewidth, and we believe that we can also obtain results on the same direction if using the out-degree of patterns or their meaningful functions as parameters of the problem.

Schema mappings. Continuing with the topic of schema mappings for graph databases, a particularly interesting issue has to do with the level of synchronization between source and target paths allowed in graph mappings. For now, we do not allow any synchronization at all between paths, but one could think of mapping rules that could specify, for instance *copying* entire paths from the source graph to the target graphs. One could even impose much milder restrictions, such as forcing that each source path has to be transferred as a target path of the same length. Numerous notions of synchronization have been studied, such as regular relations, or rational relations [Elgot and Mezei, 1965, Frougny and Sakarovitch, 1993], and query languages capable of doing this have already appeared in the literature [Barceló et al., 2010a].

Incomplete automata. Finally, the study of incomplete automata opens up many possibilities for future work. There are several open problems involving decision problems for incomplete automata for possibility semantics, and even the issue of characterizing what type of language is produced by them is still unknown. Furthermore, for most bounds on the decision problems (except of course universality and containment), the complexity under the possibility semantics is reasonable, while for the certainty semantics it is quite high (i.e., double-exponential in

practice). At the same time, the concept of $\mathcal{L}_{\Box}(e)$ captures many query answering scenarios over graph databases with incomplete information. One of the future directions of this work is to devise better algorithms for problems related to the certainty semantics under restrictions arising in the context of querying graph databases, and specifically for queries returning paths.

Another line of work has to do with closure properties: we know that the language generated by incomplete automata under certainty semantics, and for parameterized automata under possibility semantics, are regular. Therefore, results of Boolean operations on languages $\mathcal{L}_{\Box}(\mathcal{A})$ and $\mathcal{L}_{\Diamond}(e)$ remain to be regular and can be represented by NFAs; the bounds on sizes of such NFAs follow from the results shown in this dissertation. However, it is conceivable that such NFAs can be succinctly represented by parameterized regular expressions. To be concrete, one can easily derive from results in Chapter 7 that there is a doubly-exponential size NFA A so that $\mathcal{L}(A) = \mathcal{L}_{\Box}(\mathcal{A}_1) \cap \mathcal{L}_{\Box}(\mathcal{A}_2)$, and that this bound is optimal. However, it leaves open a possibility that there is a much more succinct incomplete automata $\mathcal{A}$ so that $\mathcal{L}_{\Box}(\mathcal{A}) = \mathcal{L}_{\Box}(\mathcal{A}_1) \cap \mathcal{L}_{\Box}(\mathcal{A}_2)$; in fact, nothing that we have shown contradicts the existence of a polynomial-size expression with this property. We plan to study bounds on such regular expressions in the future; we imagine this study to be relevant in applications that need to successively operate and complement different NFAs, since it might be less costly to perform these operations at an incomplete automata level, and transform back these automata into NFAs only when all of these operations have been performed.

Bibliography

[Abiteboul et al., 1999] Abiteboul, S., Buneman, P., and Suciu, D. (1999). *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman.

[Abiteboul and Duschka, 1998] Abiteboul, S. and Duschka, O. M. (1998). Complexity of answering queries using materialized views. In *PODS*, pages 254–263.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[Abiteboul et al., 1991] Abiteboul, S., Kanellakis, P. C., and Grahne, G. (1991). On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187.

[Acar et al., 2010] Acar, U., Buneman, P., Cheney, J., Van Den Bussche, J., Kwasnikowska, N., and Vansummeren, S. (2010). A graph model of data and workflow provenance. In *Proceedings of the 2nd conference on Theory and practice of provenance*, pages 8–8. USENIX Association.

[Aho, 1990] Aho, A. (1990). *Handbook of Theoretical Computer Science*.

[Amano et al., 2009] Amano, S., Libkin, L., and Murlak, F. (2009). XML schema mappings. In *PODS*, pages 33–42.

[Angles, 2012] Angles, R. (2012). A comparison of current graph database models. In *ICDE Workshops*, pages 171–177.

[Angles and Gutierrez, 2008] Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, 40(1).

[Arenas et al., 2010] Arenas, M., Barceló, P., Libkin, L., and Murlak, F. (2010). *Relational and XML Data Exchange*. Morgan & Claypool.

[Arenas et al., 2011a] Arenas, M., Barceló, P., and Reutter, J. L. (2011a). Query languages for data exchange: Beyond unions of conjunctive queries. *Theory Comput. Syst.*, 49(2):489–564.

[Arenas and Libkin, 2008] Arenas, M. and Libkin, L. (2008). Xml data exchange: Consistency and query answering. *J. ACM*, 55(2).

[Arenas et al., 2011b] Arenas, M., Pérez, J., and Reutter, J. L. (2011b). Data exchange beyond complete data. In *PODS*, pages 83–94.

[Arenas et al., 2009] Arenas, M., Pérez, J., Reutter, J. L., and Riveros, C. (2009). Inverting schema mappings: Bridging the gap between theory and practice. *PVLDB*, 2(1):1018–1029.

[Barceló, 2009] Barceló, P. (2009). Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58.

[Barceló et al., 2010a] Barceló, P., Hurtado, C. A., Libkin, L., and Wood, P. T. (2010a). Expressive languages for path queries over graph-structured data. In *PODS*, pages 3–14.

[Barceló et al., 2010b] Barceló, P., Libkin, L., Poggi, A., and Sirangelo, C. (2010b). Xml with incomplete information. *Journal of the ACM*, 58(1):1–62.

[Barceló et al., 2011a] Barceló, P., Libkin, L., and Reutter, J. L. (2011a). Parameterized regular expressions and their languages. In *FSTTCS*, pages 351–362.

[Barceló et al., 2011b] Barceló, P., Libkin, L., and Reutter, J. L. (2011b). Querying graph patterns. In *PODS*, pages 199–210.

[Barceló et al., 2012] Barceló, P., Pérez, J., and Reutter, J. L. (2012). Relative expressiveness of nested regular expressions. In *AMW*, pages 180–195.

[Barceló et al., 2013a] Barceló, P., Pérez, J., and Reutter, J. L. (2013a). Schema mappings and data exchange for graph databases. In *to appear in ICDT*, page TBD.

[Barceló et al., 2013b] Barceló, P., Reutter, J. L., and Libkin, L. (2013b). Parameterized regular expressions and their languages. *Theor. Comput. Sci.*, 474:21–45.

[Belhajjame et al., 2011] Belhajjame, K., Cheney, J., Garijo, D., Lebo, T., Soiland-Reyes, S., and Zednik, S. (December 2011). The prov ontology: Model and formal semantics. *W3C Working Draft*.

[Belleau et al., 2008] Belleau, F. o., Nolin, M.-A., Tourigny, N., Rigault, P., Morissette, J., et al. (2008). Bio2rdf: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716.

[Bernstein, 2003] Bernstein, P. (2003). Applying model management to classical meta data problems. In *CIDR*.

[Bhadra et al., 2005] Bhadra, J., Martin, A. K., and Abraham, J. A. (2005). A formal framework for verification of embedded custom memories of the motorola mpc7450 microprocessor. *Formal Methods in System Design*, 27(1-2):67–112.

[Björklund et al., 2007] Björklund, H., Martens, W., and Schwentick, T. (2007). Conjunctive query containment over trees. In *11th International Symposium on Database Programming Languages (DBPL)*, pages 66–80.

[Calvanese et al., 2007] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning*, 39(3):385–429.

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

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

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

[Calvanese et al., 2002] Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. (2002). Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443–465.

[Calvanese et al., 2008] Calvanese, D., Giacomo, G. D., and Lenzerini, M. (2008). Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3).

[Calvanese et al., 2011a] Calvanese, D., Giacomo, G. D., Lenzerini, M., and Vardi, M. Y. (2011a). Simplifying schema mappings. In *ICDT*, pages 114–125.

[Calvanese et al., 2011b] Calvanese, D., Ortiz, M., and Simkus, M. (2011b). Containment of regular path queries under description logic constraints. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 805–812. AAAI Press.

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

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

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

[David et al., 2013] David, C., Gheerbrant, A., Libkin, L., and Martens, W. (2013). Containment of pattern-based queries over data trees. In *ICDT*, pages 201–212.

[DBLP, 2013] DBLP (2013). D2R DBLP bibliography database hosted at L3S research center. `http://dblp.l3s.de/d2r/`.

[de Moor et al., 2003] de Moor, O., Lacey, D., and Wyk, E. V. (2003). Universal regular path queries. *Higher-Order and Symbolic Computation*, 16(1-2):15–35.

[Dechter, 2003] Dechter, R. (2003). *Constraint processing*. Morgan Kauffman.

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

[Diestel, 2005] Diestel, R. (2005). *Graph Theory*. Springer.

[Donato et al., 2007] Donato, D., Laura, L., Leonardi, S., and Millozzi, S. (2007). The web as a graph: How far we are. *ACM Trans. Internet Techn.*, 7(1).

[Downey and Fellows, 1999] Downey, R. G. and Fellows, M. R. (1999). *Parameterized complexity*, volume 127. springer Heidelberg.

[Elgot and Mezei, 1965] Elgot, C. and Mezei, J. (1965). On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68.

[Fagin, 2007] Fagin, R. (2007). Inverting schema mappings. *TODS*, 32(4).

[Fagin et al., 2005a] Fagin, R., Kolaitis, P., Miller, R., and Popa, L. (2005a). Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124.

[Fagin et al., 2005b] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005b). Data exchange: semantics and query answering. *TCS*, 336(1):89–124.

[Fagin et al., 2005c] Fagin, R., Kolaitis, P. G., Popa, L., and Tan, W.-C. (2005c). Composing schema mappings: Second-order dependencies to the rescue. *TODS*, 30(4):994–1055.

[Fan et al., 2010a] Fan, W., Li, J., Ma, S., Tang, N., and Wu, Y. (2010a). Graph pattern matching: from intractable to polynomial time. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):264–275.

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

[Fan et al., 2010b] Fan, W., Li, J., Ma, S., Wang, H., and Wu, Y. (2010b). Homomorphism revisited for graph matching. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):1161–1172.

[Florescu et al., 1998] Florescu, D., Levy, A., and Suciu, D. (1998). Query containment for conjunctive queries with regular expressions. In *17th ACM Symposium on Principles of Database Systems (PODS)*, pages 139–148.

[Flum and Grohe, 2006] Flum, J. and Grohe, M. (2006). *Parameterized complexity theory*, volume 3. Springer Heidelberg.

[Freydenberger, 2011] Freydenberger, D. D. (2011). Extended regular expressions: Succinctness and decidability. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 507–518.

[Freydenberger, 2013] Freydenberger, D. D. (2013). Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*. Special Issue STACS 2011. To appear.

[Frougny and Sakarovitch, 1993] Frougny, C. and Sakarovitch, J. (1993). Synchronized rational relations of finite and infinite words. TCS, 108:45–82.

[G. Klyne, 2004] G. Klyne, J. J. C. (2004). RDF concepts and abstract syntax, W3C recommendation.

[Gheerbrant et al., 2013] Gheerbrant, A., Libkin, L., and Sirangelo, C. (2013). When is naive evaluation possible? In *To appear in PODS*.

[Gheerbrant et al., 2012] Gheerbrant, A., Libkin, L., and Tan, T. (2012). On the complexity of query answering over incomplete xml documents. In *ICDT*, pages 169–181.

[Glaister and Shallit, 1996] Glaister, I. and Shallit, J. (1996). A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75–77.

[Glimm et al., 2006] Glimm, B., Horrocks, I., and Sattler, U. (2006). Conjunctive query answering for description logics with transitive roles. In *Description Logics*.

[Grahne, 1991] Grahne, G. (1991). *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer.

[Grumberg et al., 2010] Grumberg, O., Kupferman, O., and Sheinvald, S. (2010). Variable automata over infinite alphabets. In *Proceedings of the 4th International Conference on Language and Automata Theory and Applications (LATA)*, pages 561–572.

[Gutierrez et al., 2011] Gutierrez, C., Hurtado, C., Mendelzon, A. O., , and Pérez, J. (2011). Foundations of semantic web databases. *Journal of Computer and System Sciences*, 77(3):520–541.

[Gyssens et al., 1994] Gyssens, M., Paredaens, J., Van den Bussche, J., and Van Gucht, D. (1994). A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):572–586.

[Hagenah and Muscholl, 1998] Hagenah, C. and Muscholl, A. (1998). Computing epsilon-free nfa from regular expressions in o(n $\log^2$(n)) time. In *Proceedings of the 23rd International Symposium on the Mathematical Foundations of Computer Science (MFCS)*, pages 277–285.

[Harris and Seaborne, 2013] Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. W3C recommendation. `http://www.w3.org/TR/sparql11-query/`.

[Hull et al., 2003] Hull, R., Benedikt, M., Christophides, V., and Su, J. (2003). E-services: a look behind the curtain. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–14. ACM.

[Imielinski and Lipski, 1984] Imielinski, T. and Lipski, W. (1984). Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791.

[InfiniteGraph, 2013] InfiniteGraph (2013). Infinitegraph release 3.1 by objectivity inc. `http://www.objectivity.com/infinitegraph`.

[Johnson and Klug, 1984] Johnson, D. and Klug, A. (1984). Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189.

[Kaminski and Zeitlin, 2010] Kaminski, M. and Zeitlin, D. (2010). Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760.

[Kanehisa and Goto, 2000] Kanehisa, M. and Goto, S. (2000). Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30.

[Kanza et al., 2002] Kanza, Y., Nutt, W., and Sagiv, Y. (2002). Querying incomplete information in semistructured data. *Journal of Computer and System Sciences*, 64(3):655–693.

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

[Kolaitis, 2005] Kolaitis, P. G. (2005). Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75.

[Kolaitis et al., 2006] Kolaitis, P. G., Panttaja, J., and Tan, W.-C. (2006). The complexity of data exchange. In *PODS*, pages 30–39.

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

[Krentel, 1988] Krentel, M. W. (1988). The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, 36(3):490–509.

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

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: a theoretical perspective. In *PODS*, pages 233–246.

[Leser, 2005] Leser, U. (2005). A query language for biological networks. *Bioinformatics*, 21(2):ii33–ii39.

[Libkin, 2004] Libkin, L. (2004). *Elements of Finite Model Theory*. Springer.

[LinkedIn, 2013] LinkedIn (2013). `http://www.linkedin.com/`.

[Liu et al., 2004] Liu, Y. A., Rothamel, T., Yu, F., Stoller, S. D., and Hu, N. (2004). Parametric regular path queries. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 219–230.

[Liu and Stoller, 2006] Liu, Y. A. and Stoller, S. D. (2006). Querying complex graphs. In *Proceedings of 8th International Symposium on the Practical Aspects of Declarative Languages (PADL)*, pages 199–214.

[Madhavan and Halevy, 2003] Madhavan, J. and Halevy, A. Y. (2003). Composing mappings among data sources. In *VLDB*, pages 572–583.

[Manku et al., 2004] Manku, G. S., Naor, M., and Wieder, U. (2004). Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In *STOC*, pages 54–63.

[Matono et al., 2003] Matono, A., Amagasa, T., Yoshikawa, M., and Uemura, S. (2003). An efficient pathway search using an indexing scheme for rdf. *GENOME INFORMATICS SERIES*, pages 374–375.

[Melnik, 2004] Melnik, S. (2004). *Generic Model Management: concepts and Algorithms*, volume 2967 of *LNCS*. Springer.

[Milo et al., 2002] 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.

[Nash et al., 2005] Nash, A., Bernstein, P. A., and Melnik, S. (2005). Composition of mappings given by embedded dependencies. In *PODS*, pages 172–183.

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

[Neo4j, 2013] Neo4j (2013). Neo4j, the graph database. `http://www.neo4j.org/`.

[Ogata et al., 2000] Ogata, H., Fujibuchi, W., Goto, S., and Kanehisa, M. (2000). A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic acids research*, 28(20):4021–4028.

[Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems*, 34(3).

[Pérez et al., 2010] Pérez, J., Arenas, M., and Gutierrez, C. (2010). nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270.

[Pesant, 2004] Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In *Proceedings of the 10th International Conference on the Principles and Practice of Constraint Programming (CP)*, pages 482–495.

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

[San Martín and Gutierrez, 2009] San Martín, M. and Gutierrez, C. (2009). Representing, querying and transforming social networks with rdf/sparql. In *6th European Semantic Web Conference (ESWC)*, pages 293–307.

[Schaefer, 1978] Schaefer, T. J. (1978). The complexity of satisfiability problems. In *STOC*, pages 216–226.

[Shen et al., 2007] Shen, X., Zhong, Y., and Ding, C. (2007). Predicting locality phases for dynamic memory optimization. *J. Parallel Distrib. Comput.*, 67(7):783–796.

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

[Vardi, 1995] Vardi, M. Y. (1995). On the complexity of bounded-variable queries. In *PODS*, pages 266–276.

[W3C Consortium, 2013] W3C Consortium (2013). Semantic web: The w3c consortium's vision of the web of linked data. `http://www.w3.org/standards/semanticweb/`.

[Weikum et al., 2009] Weikum, G., Kasneci, G., Ramanath, M., and Suchanek, F. (2009). Database and information-retrieval methods for knowledge discovery. *Communications of the ACM*, 52(4):56–64.

[Wood, 2012] Wood, P. T. (2012). Query languages for graph databases. *Sigmod Record*, 41(1):50–60.

[Yannakakis, 1981] Yannakakis, M. (1981). Algorithms for acyclic database schemes. In *VLDB*, pages 82–94.

Appendix A

Proofs and Additional Results

A.1 Proof of Claim 5.2.2

Let $s_1, \ldots, s_p$ be an arbitrary enumeration of the states in S, f be a mapping from S to 2^S, $S' \subseteq S$ and $\mathfrak{S} \subseteq 2^S$. In order to determine whether $(f, S', \mathfrak{S})$ is realized in $\mathcal{A}_2$ we do the following. First, construct in constant time a deterministic NFA $\mathcal{A}_3$ that is equivalent to $\mathcal{A}_1$ using the standard powerset construction. Thus, the states of $\mathcal{A}_3$ are precisely the subsets S' of S. Assume that $\delta' : 2^S \times \Sigma \to 2^S$ is the transition function of $\mathcal{A}_3$. Then, from $\mathcal{A}_3$ we construct, in constant time, two deterministic NFAs $\mathcal{A}_4$ (without initial/final states) and $\mathcal{A}_5$ as follows:

- The set of states of $\mathcal{A}_4$ is the disjoint union between $\{S' \mid S' \subseteq S\}$ and $\{S' \cup \{s_{aux}\} \mid S' \subseteq S\}$, where s_{aux} is a new auxiliary state. The transition function of $\mathcal{A}_4$ is constructed from the transition function δ' of $\mathcal{A}_3$ using the following rules: (1) If transition from $S'_1 \subseteq S$ into $S'_2 \subseteq S$ labeled a exists in $\mathcal{A}_3$, add the same transition to $\mathcal{A}_4$, and add also a transition labeled a from $S'_1 \cup \{s_{aux}\}$ into $S'_2 \cup \{s_{aux}\}$. (2) Delete every transition labeled a from a state $S'_1 \subseteq S$, such that $S'_1 \cap F = \emptyset$, into a state $S'_2 \subseteq S$ such that $S'_2 \cap F \neq \emptyset$, and replace it by a transition labeled a from S'_1 into $S'_2 \cup \{s_{aux}\}$.

  Intuitively, s_{aux} works as a flag for states of $\mathcal{A}_3$ that contain some final state in F. Thus, the transition function of $\mathcal{A}_4$ leads from a state $\{s\}$ to a state $f(s) \cup \{s_{aux}\}$ over a word w if and only if $\delta'(\{s\}, w) = f(s)$, and there is a prefix w' of w such that $\delta'(\{s\}, w')$ contains some state in F. On the contrary, the transition function of $\mathcal{A}_4$ leads from a state $\{s\}$ to a state $f(s)$ over a word w if and only if $\delta'(\{s\}, w) = f(s)$, and there is no prefix w' of w such that $\delta'(\{s\}, w')$ contains some state in F.

- The set of states of $\mathcal{A}_5$ is 2^{2^S}. Its initial state is $\{\{\}\}$ and its final state is $\mathfrak{S}$. There is a transition labeled a from $\{S_1, \ldots, S_\ell\}$ into $\{S'_1, \ldots, S'_{\ell'}\}$ if and only $\{S'_1, \ldots, S'_{\ell'}\}$ is precisely the set that contains (1) all states of $\mathcal{A}_3$ of the form $\delta'(S_i, a)$, for $1 \leq i \leq \ell$, and (2) state $\delta(\{s_0\}, a)$.

Intuitively, the transition function of $\mathcal{A}_5$ leads from state $\{\{\}\}$ into state $\mathfrak{S}$ over a word w if and only if the set $\mathfrak{S}$ consists of exactly those states S'' of $\mathcal{A}_3$ such that for some suffix w' of w it is the case that $\delta'(\{s_0\}, w') = S''$.

Then we construct, in constant time, p copies $\mathcal{A}_4^1, \ldots, \mathcal{A}_4^p$ of $\mathcal{A}_4$, and set the initial state of $\mathcal{A}_4^i$ ($1 \leq i \leq p$) to be $\{s_i\}$ and its final state to be $f(s_i) \cup \{s_{aux}\}$, if $s_i \in S'$, and $f(s_i)$ otherwise. Afterwards, we construct in polynomial time the NFA $\mathcal{A}_6 := \mathcal{A}_2 \times \mathcal{A}_4^1 \times \cdots \times \mathcal{A}_4^p \times \mathcal{A}_5$. Clearly, the tuple $(f, S', \mathfrak{S})$ is realized in $\mathcal{A}_1$ if and only if $\mathcal{A}_6$ is nonempty. But the latter can be checked in polynomial time in the size of $\mathcal{A}_6$, and, thus, in the size of $\mathcal{A}_2$. Now the claim follows from the fact that there is a constant number of tuples of the form $(f, S', \mathfrak{S})$. □

A.2 Proof of Lemma 6.5.5

Part 1. Let us first assume that query $Q = (\xi, x_1, x_2)$ belongs to $\mathcal{P}^{\text{nv,re}}$. We show afterwards how to deal with label variables. We use a technique which is similar to the *roll-up* of an acyclic conjunctive query, which has been used to provide decidability results for query containment in the context of description logics (see [Glimm et al., 2006, Calvanese et al., 2008]). In this case we roll-up the acyclic graph query into an NPQ.

First we need to construct the graph $\mathcal{G}_{(\xi,x_1,x_2)}$ as follows. The nodes of $\mathcal{G}_{(\xi,x_1,x_2)}$ are the nodes of ξ, and for each edge of form (u, R, v) in ξ, add an undirected edge in $\mathcal{G}_{(\xi,x_1,x_2)}$ with two labels $\ell_{(u,v)}(\{u,v\}) = R$ and $\ell_{(v,u)}(\{u,v\}) = R^{-1}$.

Notice that $\mathcal{G}_{(\xi,x_1,x_2)}$ is a tree (since it is acyclic and connected).

Now, for every node u in $\mathcal{G}_{(\xi,x_1,x_2)}$ we define an NRE $\nu(u)$ as follows. First consider the graph $\mathcal{G}'_{(\xi,x_1,x_2)}$ obtained from $\mathcal{G}_{(\xi,x_1,x_2)}$ by deleting all the edges in the unique path between x_1 and x_2. Notice that $\mathcal{G}'_{(\xi,x_1,x_2)}$ is no longer a tree, but a *forest*, and every node of $\mathcal{G}_{(\xi,x_1,x_2)}$ belongs to a unique tree in $\mathcal{G}'_{(\xi,x_1,x_2)}$. Let T_u be the tree in $\mathcal{G}'_{(\xi,x_1,x_2)}$ to which node u belongs, and assume that T_u is a rooted tree, with root u. Then for every node v in T_u construct an expression $\tau_u(v)$ recursively as follows:

- if v is a leaf in T_u then $\tau_u(v) = \varepsilon$
- else, if v has $v_1, \ldots, v_k$ as children in T_u then

$$\tau_u(v) \;=\; [\ell_{(v,v_1)}(\{v,v_1\}) \cdot \tau_u(v_1)] \cdot \cdots \cdot [\ell_{(v,v_k)}(\{v,v_k\}) \cdot \tau_u(v_k)]$$

Finally we say that $\nu(u) = \tau_u(u)$. Notice that the size of $\nu(u)$ is linear in the size of the tree T_u. Also notice that the fact that $\mathcal{G}_{(\xi,x_1,x_2)}$ is actually a tree is crucial for defining $\nu(u)$ for every node in $\mathcal{G}_{(\xi,x_1,x_2)}$.

Now, with $\nu(u)$ for every possible node of ξ we are ready to describe the NPQ that defines (ξ, x_1, x_2). Let $x_1, u_1, u_2, \ldots, u_k, x_2$ be the unique path in $\mathcal{G}_{(\xi,x_1,x_2)}$ from x_1 to x_2. Then we

consider the expression

$$R \;=\; \nu(x)\cdot \ell_{(x,u_1)}(\{x,u_1\})\cdot \nu(u_1)\cdot \ell_{(u_1,u_2)}(\{u_1,u_2\})\cdot \nu(u_2)\cdot \cdots \\ \cdots \cdot \nu(u_k)\cdot \ell_{(u_k,y)}(\{u_k,y\})\cdot \nu(y).$$

It is not difficult to see that R is of size linear in the size of ξ. Moreover, it is not difficult to prove by induction that for every graph G it holds that $[\![R]\!]_G = Q(G)$.

Essentially, expression R is testing for the existence of the unique path (in the query (ξ,x_1,x_2)) between x_1 and x_2, and also existentially testing for all the branches that start from the variables in this path as described in (ξ,x_1,x_2). In particular, let $\psi_u(u)$ be the formula obtained by considering the conjunction of all formulas (v,R,v') obtained from the edges of graph T_u with all variables except for u existentially quantified. It is not difficult to prove that for every graph G and node v of G, it holds that $G \models \psi_u(v)$ if and only if $(v,v) \in [\![\nu(u)]\!]_G$.

Moreover, it is easy to see that if $(x_1,R_0,u_1),(u_1,R_1,u_2),\ldots,(u_k,R_k,x_2)$ are the edges defining the single path from x_1 to x_2 in ξ, then (ξ,x,y) is equivalent to the formula

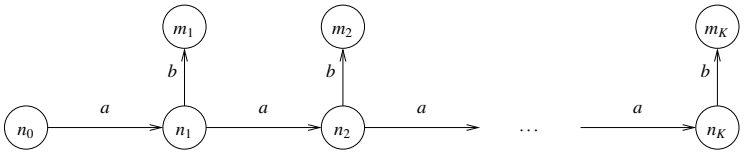
$$\exists u_1 \cdots \exists u_k \Big(\psi_x(x_1) \wedge (x_1,R_0,u_1) \wedge \psi_{u_1}(u_1) \wedge (u_1,R_1,u_2) \wedge \psi_{u_2}(u_2) \cdots \\ \cdots \wedge \psi_{u_k}(u_k) \wedge (u_k,R_k,y) \wedge \psi_{x_2}(x_2) \Big)$$

From this we obtain that our construction of R is correct and then the NPQ $Q' = (x_1,R,x_2)$ is equivalent to (ξ,x_1,x_2).

Now if we start with a query $Q = (\xi,x_1,x_2)$ with label variables from $\mathcal{W}$, we first need to transform such query into a union of queries $Q' = \bigcup(\eta(\xi),x_1,x_2)$ for each valuation η from $\mathcal{W}$ to Σ. Note that queries Q and Q' are equivalent. For each of these subqueries we construct an NRE R as explained above, and then we just consider the NPQ given by the disjunction of all of them.

Part 2

Let $Q = (x,a[b])^+,y)$, and assume for the sake of contradiction that there is a graph query (ξ,x,y) using label variables from $\mathcal{W}$ that is equivalent to Q over Σ. Let N be the number of edges in ξ, and consider the following graph G



where $K > N+1$. Now, since we are assuming that (ξ,x,y) is equivalent to Q and given that $(n_0,n_K) \in [\![R]\!]_G$, we have that there is a valuation η from $\mathcal{W}$ to Σ such that $\eta(\xi)[x/n_0,y/n_K] \models G$.

Assume without loss of generality that the edges of $\eta(\xi)$ are of the form $(z_1, r_1, z'_1) \wedge \cdots \wedge (z_N, r_N, z'_N)$, where now each expression $r_1, \ldots, r_N$ is a regular expression over Σ. Then by the semantics of patterns, we know that there exists a mapping h from $\{z_1, z'_1, \ldots, z_N, z'_N\}$ to $\{n_0, n_1, m_1, \ldots, n_K, m_K\}$ such that $h(x) = n_0$, $h(y) = n_K$ and for every $i \in \{1, \ldots, N\}$ there is a path ρ_i^h between $h(z_i)$ and $h(z'_i)$ such that the sequence of edge labels λ_i^h associated to ρ_i^h, satisfies the expression r_i. We use the mapping h and the fact that K is sufficiently large compared with N to obtain a contradiction.

From h and ξ we construct a graph $G_{h(\xi)}$ as follows. Initially consider the values $h(z_1), h(z'_1), \ldots, h(z_N), h(z'_N)$ as nodes in $G_{h(\xi)}$. Now, for every edge (z_i, r_i, z'_i) of ξ let $\lambda_i^h = a_0 a_1 \ldots a_k$ be the sequence of edges in the path between $h(z_i)$ and $h(z'_i)$ in G that satisfies the regular expression r_i when evaluated over G. Then, we include k fresh nodes $s_1, s_2, \ldots, s_k$ to $G_{h(\xi)}$ and the following edges. Assuming that $s_0 = h(z_i)$ and that $s_{k+1} = h(z'_i)$, then for every $j \in \{1, \ldots, k+1\}$:

- if $a_j = a$ or $a_j = b$ then add edge (s_{j-1}, a_j, s_j) to $G_{h(\xi)}$, and
- if $a_j = a^-$ or $a_j = b^-$ then add edge (s_j, a_j, s_{j-1}) to $G_{h(\xi)}$.

By the construction of $G_{h(\xi)}$ it is easy to conclude that $\eta(\xi)[x/n_0, y/n_K] \models G_{h(\xi)}$, and thus $\xi[x/n_0, y/n_K] \models G_{h(\xi)}$. We prove below that $(n_0, n_K) \notin [\![R]\!]_{G_{h(\xi)}}$, which shows that $(n_0, n_K) \notin Q(G_{h(\xi)})$ and suffices for the proof.

First notice that if n_0 and n_K are in different connected components in $G_{h(\xi)}$, then clearly $(n_0, n_K) \notin [\![R]\!]_{G_{h(\xi)}}$. Thus, assume that n_0 and n_K are in the same connected component of $G_{h(\xi)}$. Notice that since n_0 and n_K are at distance K in G, then any path connecting n_0 and n_K in $G_{h(\xi)}$ should have at least K edges. Now, given that ξ has N edges and $K > N + 1$ we know that each one of the paths connecting n_0 and n_K in $G_{h(\xi)}$ should contain a portion that was constructed by converting a sequence of edge labels of G into a *linear* path in $G_{h(\xi)}$. All this implies that every possible path from n_0 to n_K in $G_{h(\xi)}$ should contain some intermediate nodes such that the sum of the out and in-degrees of the node is at most 2 (they are part of a line). On the other hand, given the semantics of NREs, every path that satisfies the expression $(a \cdot [b])^+$ should be such that all intermediate nodes (that is without considering the first and last node), should have in-degree at least 1 (one edge going into the node with label a) and out-degree at least 2 (one edge going out with label a, and another going out with label b), which implies that no path from n_0 to n_K in $G_{h(\xi)}$ satisfies expression $(a \cdot [b])^+$.

A.3 Proof of Claim 6.6.6

We prove the Claim using induction. Note that we only need to show that $[\![R]\!]_{G_{\mathbf{T}}} = [\![\mathrm{rew}(R)]\!]_{G_{\mathbf{S}}}$.

- If $R = \varepsilon$, then a pair (n_1, n_1) of nodes belong to $[\![R]\!]_{G_\mathbf{T}}$ if and only if there is a pair of nodes containing node n_1 that satisfies the left hand side of the dependencies in $\mathcal{M}$. If n_1 is the first component of that pair, then this must mean that $(n_1, n_1) \in [\![R']\!]_{G_\mathbf{S}}$, where $R' = [R_1 \mid \cdots \mid R_n]$. On the other hand, if n_1 is in the second component of such pair, then $(n_1, n_1) \in [\![R'']\!]_{G_\mathbf{S}}$, where $R'' = [R_1 \mid \cdots \mid R_n]$.

- If $R = a$, then since $\mathcal{M}$ is a GAV mapping, a pair $(n_1, n_2) \in [\![R]\!]_{G_\mathbf{T}}$ if and only if (n_1, n_2) satisfies one of the left hand side of the dependencies that produce a labelled edges, according to $\mathcal{M}$, which suffices for the proof.

- The case when Ra^- is analogous to the previous one.

- If $R = e_1 \cdot e_2$, then $\mathrm{rew}(R) = \mathrm{rew}(R_1) \cdot \mathrm{rew}(R_2)$. Assume first that $(n_1, n_2) \in [\![R]\!]_{G_\mathbf{T}}$. Then there is a node n_3 such that $(n_1, n_3) \in [\![e_1]\!]_{G_\mathbf{T}}$ and $(n_3, n_2) \in [\![e_2]\!]_{G_\mathbf{T}}$. From the induction hypothesis, we have that $(n_1, n_3) \in [\![\mathrm{rew}(e_1)]\!]_{G_\mathbf{S}}$ and $(n_3, n_2) \in [\![\mathrm{rew}(e_2)]\!]_{G_\mathbf{S}}$, which suffices for the proof. The other direction is completely symmetrical.

- The rest of the proof follows along the same line as this previous cases.

A.4 Proof of Lemma 8.2.2 for Incomplete Automata

We show the following:

Lemma A.4.1 *Given a set $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of incomplete automata, it is possible to build, in polynomial time, an automaton $\mathcal{A}'$ such that $\mathcal{L}_\Box(\mathcal{A}')$ is empty if and only if $\mathcal{L}_\Box(\mathcal{A}_1) \cap \cdots \cap \mathcal{L}_\Box(\mathcal{A}_k)$ is empty.*

Proof: We only focus on the case when Σ contains at least two symbols. The case when Σ contains a single symbol is treated in the same way as in the proof of Lemma 8.2.2 in Chapter 8.

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.

Pick up two arbitrary symbols S and R from Σ. The incomplete 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 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 disjoint and $\{q_1', \ldots, q_k', p_1', \ldots, p_k'\} \cap \bigcup_{j \leq k} Q_j = \emptyset$;

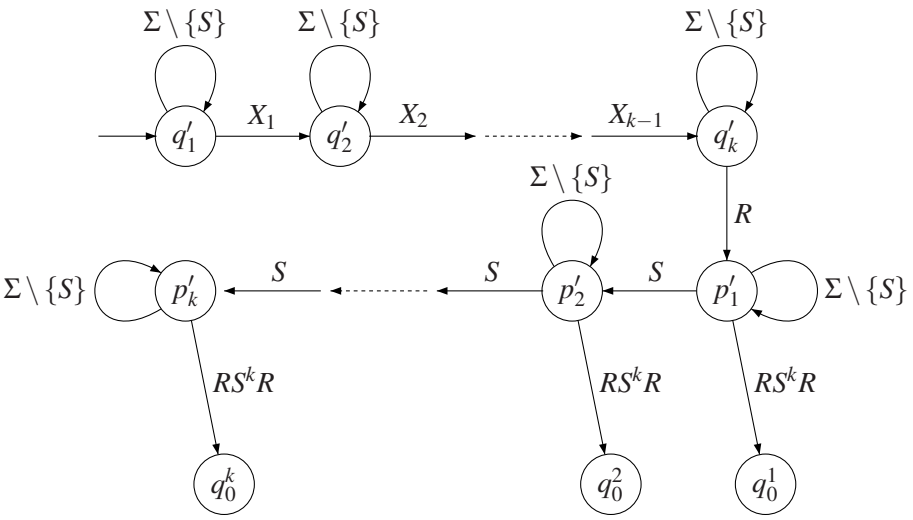


Figure A.1: Control section of incomplete automaton $\mathcal{A}'$.

- $F' = \bigcup_{j\leq k} F_j$;
- 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_0^j), for every $j \in [1,k]$

The *control* part of automaton $\mathcal{A}'$ is depicted in Figure A.1. Notice that the states $q_0^1, \ldots, q_0^k$ 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 $\bigcap_{j \leq k} L_s(\mathcal{A}_j)$, which contradicts the fact that $\bigcap_{j \leq k} L_s(\mathcal{A}_j)$ is empty.

Assume for the sake of contradiction that there exists $1 \leq j \leq k$ 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 $1 \leq i \leq k$ and $i \neq j$.

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 $\bigcap_{j \leq k} L_s(\mathcal{A}_j)$. 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 $(\bar{c}^k SR)^{k-1} RS^k Rw$, which is a contradiction.

Let $\nu = (\eta, \theta)$ be an arbitrary valuation for $\mathcal{A}'$. We show that $(\bar{c}^k RS)^{k-1} RS^k Rw$ 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 being 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 $(\bar{c}^k RS)^{k-2} RS^k R$. 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 $\mathcal{A}'$. □