UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

NEW COMPLEXITY BOUNDS FOR EVALUATION AND CONTAINMENT OF GRAPH QUERY LANGUAGES

TESIS PARA OPTAR AL GRADO DE DOCTOR EN CIENCIAS
MENCIÓN COMPUTACIÓN

MIGUEL ÁNGEL ROMERO ORTH

PROFESOR GUÍA:
PABLO BARCELÓ BAEZA

MIEMBROS DE LA COMISIÓN:
AIDAN HOGAN
JORGE PÉREZ ROJAS
MAURIZIO LENZERINI

Este trabajo ha sido financiado por CONICYT
y el Centro de Investigación de la Web Semántica

SANTIAGO DE CHILE
2016

Resumen

Las bases de datos de grafos han recibido mucho interés en los últimos años, debido a sus aplicaciones en temas como las redes sociales o la Web Semántica. En esta tesis estudiamos lenguajes de consultas que poseen las características navegacionales fundamentales para las distintas aplicaciones de bases de datos de grafos. Estos incluyen la clase de *consultas regulares de caminos* (RPQs), las cuales chequean si dos nodos están conectados por un camino cuya etiqueta satisface una expresión regular; la clase de las RPQs con *inversos* (2RPQs), que adicionalmente permiten navegar arcos en la dirección reversa; la clase de las *uniones de conjunciones* de 2RPQs (UC2RPQs), que resulta de cerrar las 2RPQs bajo las operaciones de *join*, *projección* y *unión*; y la clase de las *consultas regulares* (RQs) que adicionalmente cierra las UC2RPQs bajo *clausura transitiva*.

En esta tesis demostramos nuevos resultados de complejidad para UC2RPQs y RQs. En la primera parte, estudiamos evaluación de UC2RPQs. Este problema es computacionalmente difícil: NP-completo en general y W[1]-completo en complejidad parametrizada. Esto ha motivado la búsqueda de restricciones que hacen que la evaluación sea tratable o tratable de *parámetro fijo*. Las más importantes son las clases de UC2RPQs de *treewidth acotado*, que se pueden evaluar en tiempo polinomial, pero hasta la fecha no se conocen otras restricciones que sean tratables o tratables de parámetro fijo. El resultado principal de esta parte es que la evaluación de UC2RPQs de treewidth acotado *módulo equivalencia* es tratable de parámetro fijo. Más precisamente, demostramos que, para cada $k \geq 1$ fijo, existe un algoritmo tratable de parámetro fijo que evalúa UC2RPQs que son equivalentes a alguna UC2RPQ de treewidth a lo más k. También estudiamos el caso cuando la cota k es 1, esto es, la clase de UC2RPQs *semánticamente acíclicas*, y obtenemos aún más resultados. En particular, demostramos que chequear acaso una UC2RPQ es semánticamente acíclica es decidible y Expspace-completo.

En la segunda parte, estudiamos contención de RQs. Las RQs han emergido recientemente como un lenguaje natural para bases de datos de grafos. Las RQs tienen propiedades naturales de clausura, no como las UC2RPQs que no son cerradas bajo clausura transitiva. Además, evaluar RQs no es más díficil que evaluar UC2RPQs (NP-completo). Sin embargo, el problema de contención de RQs aún está abierto. Este problema es decidible, pero sólo cotas no *elementales* son conocidas. En contraste, es sabido que contención de UC2RPQs es elemental, específicamente, Expspace-completo. El resultado principal de esta parte es que contención de RQs es 2Expspace-completo, y luego, tiene complejidad elemental tal como las UC2RPQs. También estudiamos restricciones de RQs que mejoran la complejidad de evaluación o contención, y algunas extensiones. En particular, demostramos que contención para una generalización natural de RQs para bases de datos relacionales es 2Expspace-completo.

Abstract

Graph databases have gained renewed interest in recent years, due to their application in areas such as social networks and the Semantic Web. We study graph query languages that provide the fundamental navigational features needed in different graph database applications. This includes the class of *regular path queries* (or RPQs for short), which check whether two nodes are connected by a path whose label satisfies a regular expression; the class of *two-way RPQs* (2RPQs), which additionally enables backward navigation of edges; the class of *unions of conjunctive* 2RPQs (UC2RPQs), which results from closing 2RPQs under the operations of *join*, *projection*, and *union*; and the class of *regular queries* (RQs), which additionally closes UC2RPQs under *transitive closure*.

In this thesis, we provide new complexity results for UC2RPQs and RQs. In the first part, we study query evaluation for UC2RPQs. This problem is known to be computationally hard: NP-complete in combined complexity and W[1]-complete in parameterized complexity. This has motivated the search for restrictions that lead to (fixed-parameter) tractable evaluation. The most prominent restrictions are the classes of UC2RPQs of *bounded treewidth*, which can be evaluated in polynomial time, but no other tractable or fixed-parameter tractable restrictions are known to date. Our main result in this part is that evaluation for UC2RPQs of bounded treewidth *modulo equivalence* is fixed-parameter tractable. More precisely, we show that, for each fixed $k \geq 1$, there is a fixed-parameter tractable algorithm that evaluates UC2RPQs that are equivalent to some UC2RPQ of treewidth at most k. We also study the case when the bound k equals 1, that is, the class of *semantically acyclic* UC2RPQs, and provide further results. In particular, we show that checking whether a UC2RPQ is semantically acyclic is decidable and EXPSPACE-complete.

In the second part, we study query containment for RQs. The class of RQs has emerged only recently as a natural graph query language. RQs have natural closure properties, unlike UC2RPQs that are not closed under transitive closure. Moreover, RQs are not harder to evaluate than UC2RPQs (NP-complete). Nevertheless, the containment problem for RQs has been open so far. This problem is decidable, but only *nonelementary* complexity upper bounds are known. In contrast, query containment for UC2RPQs is known to be elementary, specifically, EXPSPACE-complete. Our main result in this part is that containment of RQs is 2EXPSPACE-complete, and therefore, it has elementary complexity just like UC2RPQs. We also study restrictions of RQs that help to alleviate the complexity of evaluation or containment, and also some extensions. In particular, we show that containment of a natural generalization of RQs for relational databases is still 2EXPSPACE-complete.

Acknowledgements

First and foremost, I would like to thank my supervisor Pablo Barceló who gave me the opportunity to do research and introduced me to the subject of database theory. I am eternally grateful to Pablo for his support, collaboration and advice during my studies. Many thanks to Moshe Vardi for sharing his research ideas with Pablo and me. The completion of this work would not be possible without his close collaboration. I also would like to thank Juan Reutter whose collaboration was fundamental for developing the work presented in the second part of this thesis. I am also very grateful to Juan for many useful academic and life advices. Many thanks as well to Gaëlle Fontaine for helpful discussions about semantically acyclic UC2RPQs. Finally, I would like to thank Leonid Libkin for proposing (together with Pablo) the idea of query approximation as a fruitful line of research. This idea was the starting point of this thesis project.

I also would like to thank the members of the thesis committee, Aidan Hogan, Jorge Pérez and Maurizio Lenzerini, for providing useful comments that help to improve this thesis.

Countless thanks to all my friends from the DCC and the Database/Semantic Web research group who made this experience so great: Aidan (a.k.a Elliot), (el gurú) Claudio, Cristián, Cristóbal, Domagoj, Gaëlle, Jorge, Jota, Juan, Marcelo, Martín, Pablo, Pablo (Estefó) and Pablo (Muñoz), to name just a few. Many thanks also to Angélica and Sandra for making my life easier (although sometimes harder) during my studies, and for their administrative support during these last weeks, which help me to defend my Ph.D. on time.

Finalmente, quiero agradecer a mi familia por todo el cariño y apoyo que me dieron durante este proceso, y a la Pauli por alegrarme los días y acompañarme en estos últimos meses :-)

This work was supported by CONICYT Ph.D. Grant and the Millennium Nucleus Center for Semantic Web Research Grant NC120004.

Contents

List of Figures

Chapter 1

Introduction

Since the introduction of the relational model by Codd in 1970 [49, 50], relational databases have largely dominated the database market. However, researchers soon realized that new database systems were needed to handle new types of applications, for which the relational model was too restrictive. During the 80s and 90s, several database models emerged to enhance and complement relational databases. One of these models is the graph database model, which attempts to capture the inherent graph structure of data appearing in applications such as social networks or geographical information systems, where the interconnectivity of data is an important aspect.

In the late eighties, graph database research was first motivated by hypertext systems [51, 52]. During the 1990s, the model gained popularity due to its connections with object-oriented databases [93, 104] and semistructured data [3, 5, 31]. But it was not until the mid 2000s that graph databases gained significant interest due to new applications such as the Semantic Web [91, 105, 156], social networks [66, 139, 142], biological databases [113, 116, 129], crime detection networks [68, 67], and several others. Managing graph-structured data is currently one of the most active research topics in the database community, and there are several vendors offering graph database products [60, 101, 127]. One prominent example is Neo4j [127], whose presence in the database market has been constantly growing.

Each application domain requires a particular graph data model, which has led to the proposal of several models for graph databases (see [8] for a survey). In its simplest form, a graph database consists of a finite set of nodes representing objects and labeled edges between nodes representing relationships between objects. For instance, Figure 1.1 shows a graph database representing a social network. The labels `helps` and `knows` represent that one person helps (resp. knows) another person in the network. In this thesis, we focus on this simple model as it is expressive enough to capture graph data in several scenarios, and most theoretical issues related to graph databases already appear in this model.

A fundamental issue related to any database model is the design of declarative query languages. A query language has to be capable of expressing a wide variety of relevant queries while keeping the complexity of main database tasks low. For example, in the social network from Figure 1.1, one might be interested in finding all the *collaborators*, that is, all

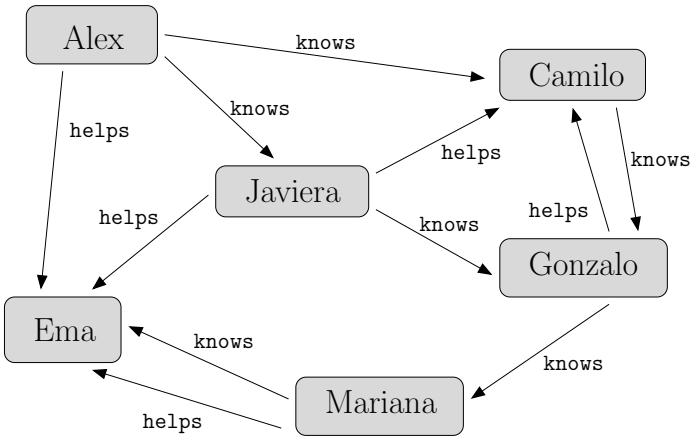

Figure 1.1: A social network represented as a graph database

pairs of people that help the same person. This can be expressed by the following *conjunctive query*:

$$\theta(x, y) \leftarrow \texttt{helps}(x, z), \texttt{helps}(y, z)$$

Observe how the variable z, which is existentially quantified, appears in both atoms of θ. Note that the pair $(Alex, Mariana)$ is in the answer as we can assign z to Ema, but $(Alex, Gonzalo)$ is not. The class of conjunctive queries (CQs) is one of the most fundamental and well-studied class of database queries [4], as they are the most frequently used in practice. CQs correspond to the *select-project-join* fragment of the relational algebra [4], or equivalently, to the SELECT-FROM-WHERE SQL queries that only have equality of attributes in the WHERE condition. Intuitively, CQs check for the existence of a fixed pattern in the database.

Graph databases differ from relational databases in that the topology of the data is as important as the data itself. Therefore, users are not only interested in traditional relational queries, such as CQs, but also in *navigational* queries that navigate along paths of unspecified length. One of the most basic classes of navigational queries is the class of *regular path queries* (or RPQs for short) [57], which check whether two nodes are connected by a path that satisfies a regular expression. For instance, using the example of Figure 1.1, the RPQ $\texttt{knows}^*$ selects all pairs of persons connected by a path of `knows`-labeled edges, that is, persons that indirectly know other persons. If we are interested in persons that indirectly know other persons by at least 2 intermediate persons, we can use the RPQ $\texttt{knows} \cdot \texttt{knows} \cdot \texttt{knows} \cdot \texttt{knows}^*$. Observe that $(Alex, Mariana)$ belongs to the answer as witnessed by the path $Alex, Camilo, Gonzalo, Mariana$. On the other hand, $(Alex, Gonzalo)$ and $(Alex, Camilo)$ are not in the answer as any path of `knows`-labeled edges between these persons has length at most 2.

The class of RPQs is an important building block for most proposed graph query languages. For instance, RPQs were present in one of the first graph query languages, the

so-called G, defined by Cruz et al. [57] in the late 80s. Since then, several graph query languages have been proposed, for instance, in the context of graph-based object databases [92, 93, 132], semistructured data [5, 71], social networks [139, 142] and the Semantic Web [95, 112]. RPQs are an integral part of most of them. On the practical side, the latest version of the W3C recommendation to query RDF documents, namely SPARQL 1.1 [95], includes *property paths*, which are based on RPQs. Also, the query language *Cypher* [58] provided by Neo4j – a major graph database vendor – includes the ability to match paths against regular expressions of a restricted form. Thus RPQs also feature in the most prominent graph query languages found in practice.

In this thesis, we study the most basic navigational query languages for graph databases. We focus on languages that provide fundamental features needed in different graph database applications. This includes the class of RPQs and several extensions defined below, such as the class of *conjunctive* RPQs that enable us to perform joins of RPQs and projection of variables. The languages we consider, for instance conjunctive RPQs, have been the topic of a vast amount of research over the last 20 years [13, 17, 33, 34, 37, 73, 137]. (For a survey of graph query languages, see [12, 157].)

The first extension of RPQs we consider includes the *inverse* operator, which enables backward navigation of edges. Let us consider again the example in Figure 1.1. Recall that two persons are collaborators if they help the same person and that these can be defined by the CQ $\theta(x, y) \leftarrow \texttt{helps}(x, z), \texttt{helps}(y, z)$. These can also be defined by the 2RPQ $\texttt{helps} \cdot \texttt{helps}^-$. Note how the expression $\texttt{helps}^-$ indicates that a helps-labeled edge must be traversed in the backward direction. To define all the indirect collaborators, we can use the 2RPQ $(\texttt{helps} \cdot \texttt{helps}^-)^*$. In this case, the pair $(Mariana, Gonzalo)$ is in the answer as witnessed by the sequence *Mariana*, *Ema*, *Javiera*, *Camilo*, *Gonzalo*.

We can combine the expressive power of CQs and 2RPQs to define the class of *conjunctive* 2RPQs (C2RPQs). In other words, C2RPQs are the closure of 2RPQs under joins and projections of variables. For example, let us say that a person is a *potential friend* of another person if he knows and helps this person at the same time through a chain of people. Thus the set of all persons that have a potential friend can be defined by the following C2RPQ:

$$\gamma(x) \leftarrow \texttt{knows}^*(x, y), \texttt{helps}^*(x, y)$$

Note that *Javiera* belongs to the answer as *Ema* is a potential friend. In contrast with CQs, which check the existence of a fixed pattern, C2RPQs look for "flexible" patterns in the database. Finally, we can close C2RPQs under *unions*, which leads to the class of *unions* of C2RPQs (UC2RPQs). For example, to define all persons that have either a potential friend or an indirect collaborator, we can use the UC2RPQ $\{\gamma, \gamma'\}$, where γ' is defined by the rule $\gamma'(x) \leftarrow (\texttt{helps} \cdot \texttt{helps}^-)^*(x, y)$. UC2RPQs can be seen as the analog of the *unions* of CQs (UCQs) from the relational world.

The main contributions of this thesis are new complexity bounds for evaluation and containment of UC2RPQs and a natural extension of these, namely, the *regular queries*, which we introduce in the following sections. The main results of this work are the following:

1. The evaluation problem is *fixed-parameter tractable* for UC2RPQs of *bounded treewidth*

modulo equivalence.

2. The containment problem for regular queries is elementary and 2EXPSPACE-complete.

In the following sections, we present the content of this thesis in more detail. In Section 1.1, we explain our main goals and formulate the research questions we address in this work. In Section 1.2, we present our main contributions and some related work.

1.1 Main goals

This work is a theoretical study of the complexity of UC2RPQs and some extensions thereof. It has two main parts, each one with different goals and research questions, as we explain in the next sections.

1.1.1 Fixed-parameter tractable evaluation of UC2RPQs

A fundamental issue in every query language is the complexity of query evaluation. In order to formally analyze its complexity, one typically studies how difficult is to recognize whether a given tuple belongs to the answer of a query over a database. In other words, we look at the problem as a *decision* problem, instead of looking at it as the *computation* problem of enumerating all the tuples in the answer. This approach is common in the database literature [12, 131, 151], as the decision problem is the core problem underlying query evaluation and it can be studied in terms of standard complexity classes. We formally denote this decision problem as the *evaluation* problem. A key issue in database theory then is to find efficient algorithms for the evaluation problem of relevant query languages.

The widely adopted theoretical notion for efficiency is that of polynomial time solvability. For the case of query evaluation, this means a running time that is bounded by a polynomial in $|q| + |\mathcal{D}|$, where $|q|$ and $|\mathcal{D}|$ denote the size of the query and database, respectively. Unfortunately, the evaluation problem for most relevant query languages is computationally hard, and thus a general polynomial time evaluation algorithm for them is very unlikely to exists. For instance, evaluating UCQs and UC2RPQs is NP-complete [40], while evaluating SQL or *first-order* queries is PSPACE-complete [4]. However, it was noted rather early by Vardi [151], and later by Papadimitriou and Yannakakis [131, 161], that this complexity measure is rather restrictive and unrealistic as it makes no distinction between the size of the query and the size of the database, whereas typically the query is significantly smaller than the database. For instance, suppose we can solve the evaluation problem in $O(2^{|q|} \cdot |\mathcal{D}|)$ time. Although, this running time is not polynomial but exponential in the combined size $|q| + |\mathcal{D}|$, it is only exponential in $|q|$, which is very small in comparison with $|\mathcal{D}|$. Hence, a running time of $O(2^{|q|} \cdot |\mathcal{D}|)$ may be acceptable in cases where queries are small, which often holds true in practice.

Parameterized complexity theory [63, 75] provides a framework to analyze these types of problems. This theory distinguishes a part of the input, called the *parameter*, which is

significantly smaller than the total input. A central notion in this theory is *fixed-parameter tractability* that relaxes classical polynomial time tractability by allowing exponential or higher behavior only with respect to the parameter. As discussed above, the natural parameter for the evaluation problem is the size of the query. Then a fixed-parameter tractable evaluation algorithm has a running time of the form $O(f(|q|) \cdot |\mathcal{D}|^c)$, where f is some function depending only on $|q|$, and c is a constant independent of $|q|$. Understanding when it is possible to obtain fixed-parameter tractable evaluation algorithms is a fundamental issue in database theory and it has motivated a large body of research over the last 20 years [46, 47, 64, 83, 84, 86, 89, 87, 131, 161].

In this first part, our main goal is to find fixed-parameter tractable evaluation algorithms for UC2RPQs. Unfortunately, it is widely believed that there is no such an algorithm that works for every possible input. This follows from the fact that the parameterized complexity of UC2RPQ evaluation is W[1]-complete [131] (this is an intractability result analogous to the NP-completeness result mentioned above, but relative to the parameterized world [75]). However, real-world queries commonly have a particular structure depending on the application domain. This structure can be exploited in order to obtain better evaluation algorithms. Thus by considering a particular class or *restriction* of UC2RPQs the evaluation problem could become fixed-parameter tractable. This motivates the main research questions of this part:

Can we find relevant restrictions of UC2RPQs with a fixed-parameter tractable evaluation problem? Or ultimately, what are precisely the restrictions of UC2RPQs with a fixed-parameter tractable evaluation problem?

To the best of our knowledge, structural restrictions of UC2RPQs that lead to fixed-parameter tractability have only been studied recently by Barceló et al. [13]. It follows from [13] that UC2RPQs of *bounded treewidth* can be evaluated in polynomial time; in particular, they are fixed-parameter tractable. The notion of treewidth is a well-known notion in graph theory [61] and it measures the similarity of a graph to a tree. This measure can be applied to UC2RPQs by considering the *underlying graph* of the query, which capture how variables interact with each other in the query.

Of course, the idea of restricting the treewidth of database queries is not new, and for instance, there is a large body of research applying this idea to the class of (U)CQs in the context of relational databases [45, 59, 80, 81, 85, 89, 86, 107]. Interestingly, it has been shown that there is an intimate connection between bounded treewidth and tractable evaluation of UCQs. One of the first results by Chekuri and Rajaraman [45] shows that CQs of bounded treewidth can be evaluated in polynomial time. To be more precise, a class $\mathcal{C}$ of CQs has bounded treewidth if there is a constant $k \geq 1$ such that each query in $\mathcal{C}$ has treewidth at most k.

Dalmau et al. [59] showed that tractability still holds for the classes of CQs of bounded treewidth *modulo equivalence*. Formally, we say that two queries q and q' are *equivalent* if the answer of q and q' is the same, over *all* databases. Then, a class $\mathcal{C}$ has bounded treewidth modulo equivalence if there is a constant $k \geq 1$ such that each CQ in $\mathcal{C}$ is equivalent to a CQ with treewidth at most k. After this, Grohe [86] proved this result to be optimal under

certain assumptions. Specifically, if $\mathcal{C}$ is a class of CQs with *bounded arity*, then under widely believed complexity theoretical assumptions, the following are equivalent: (i) evaluation for $\mathcal{C}$ is in polynomial time, (ii) evaluation for $\mathcal{C}$ is fixed-parameter tractable, and (iii) $\mathcal{C}$ has bounded treewidth modulo equivalence. A class of CQs has bounded arity if there is a constant $r \geq 1$ that bounds the *arity* of each CQ in the class, that is, the maximum arity of a predicate appearing in the query.

All the previous results can be easily extended to UCQs. Observe also that in the context of graph databases, a class of UCQs always has bounded arity, since the arity of any query is precisely 2. As a consequence, the notion of bounded treewidth modulo equivalence precisely characterizes fixed-parameter tractable evaluation of UCQs over graph databases. Thus, to address our main research question and understand when UC2RPQ evaluation is fixed-parameter tractable it is natural to start by understanding what is the role of bounded treewidth in this context. This motivates our specific research questions:

- *Is evaluation fixed-parameter tractable for UC2RPQs of bounded treewidth modulo equivalence?*
- *Does bounded treewidth modulo equivalence characterize fixed-parameter tractability for UC2RPQs (as in the case of UCQs)?*
- *Does fixed-parameter tractability and polynomial time tractability coincide for UC2RPQs (as in the case of UCQs)?*

The search for efficient restrictions of database queries is motivated by practical applications. Consider for instance the important class of UCQs over relational databases. Several tractable and fixed-parameter tractable restrictions for these queries have been introduced, e.g., acyclicity [160], bounded treewidth [45, 59], bounded *hypertree* width [81, 79], bounded *fractional* hypertree width [88], and bounded *submodular* width [126], amongst others. Although these specialized algorithms are not implemented in popular commercial systems (as they are based on different optimization techniques, such as *cost-based optimization* [4, 41], and simple general algorithms), there is much research trying to integrate these algorithms into query optimizers of existing and future database systems [1, 2, 6, 7, 77, 78] (see [79] for a discussion of the practical relevance of hypertree decompositions). For instance, the EmptyHeaded relational engine uses hypertree decompositions in its query planner [1, 2]. The system has been tested against state-of-the-art specialized engines, and outperforms them in several cases, for instance, on many cyclic queries. This suggests that tractable UCQ restrictions have the potential of providing new query optimization techniques with a great practical impact.

Finding tractable restrictions is even more relevant in practice for the case of UC2RPQs: Unlike the case of UCQs and relational databases, there are no robust optimization techniques to handle UC2RPQs over large graph datasets, and most popular systems, e.g., Neo4j [127] or Apache Jena [103] and Virtuoso [154] for RDF processing, do not scale well on large graphs when evaluating general navigational queries. This has motivated much research in recent years trying to optimize the evaluation of graph queries, especially RPQs [9, 110, 115, 145, 158, 159, 128]. Finding efficient restrictions for UC2RPQs may thus provide new tools for optimizing graph database systems.

The main result in this part is a positive answer to the first research question posed above. Specifically, we show that, for each fixed $k \geq 1$, there is a fixed-parameter tractable algorithm that evaluates UC2RPQs that are equivalent to some UC2RPQ with treewidth at most k. To show this result we develop new methods combining database and automata techniques.

1.1.2 Containment of regular queries

The search for a natural query language that achieves a good balance between expressiveness and complexity has been a major issue in graph database research. The class of UC2RPQs constitutes a fundamental graph query language: it has a reasonable complexity and can express many interesting navigational queries. However, there are several relevant queries that still cannot be expressed by UC2RPQs, which has motivated the proposal of a myriad of more expressive graph query languages [12, 13, 99, 98, 122, 111, 137, 138]. In this second part, we study one such an extension, namely, the class of *regular queries* (RQs), which we introduce below.

Besides expressive power, a key property for a query language is that of *algebraic closure*. A natural query language should be defined by means of basic queries and closure under certain operations. This allows users to easily define complex queries by reusing simpler ones. For instance, the relational algebra is defined as the closure of a set of relational operators [4]. Also, the class of CQs is closed under join and projection, while the class of UCQs is also closed under union. In a different context, the query language SPARQL – the W3C recommendation for querying RDF databases – is also defined in terms of basic operations [136, 133, 134].

Since navigational queries are central for graph databases, a fundamental operation is that of *transitive closure*. For instance, the class of RPQs is defined by closing atomic queries of the form a, for a label a in the database, under concatenation, union and transitive closure. The class of 2RPQs is also closed under the inverse operation. In contrast, UC2RPQs are closed under join, projection and union (just like UCQs), but not under transitive closure. Indeed, the transitive closure of a binary UC2RPQ cannot be expressed by a UC2RPQ in general. For instance, in the example of Figure 1.1, let us say that a person is a *friend* of another person if he knows and helps the other person. This can be expressed by the C2RPQ $\gamma(x, y) \leftarrow \texttt{knows}(x, y), \texttt{helps}(x, y)$. Now a natural query is to find all the indirect friends, that is, persons connected by a chain of friends. This query corresponds to the transitive closure of γ. It can be shown that this query cannot be expressed by any UC2RPQ (see e.g. [25, 123, 140]). Then a natural definition for a class of graph queries is the closure of atomic queries under join, projection, union and transitive closure. This constitutes precisely the class of RQs.

We formalize RQs as a fragment of *Datalog* [4], which is the extension of UCQs with recursion. An RQ is a *binary nonrecursive* Datalog program over graph databases extended with the transitive closure of binary predicates (either *extensional* or *intensional* predicates [4]). A program is binary if all its intensional predicates have arity 2. For example, in order to define all indirect friends in the example of Figure 1.1, we can use the following RQ (*Ans*

denotes the answer or goal predicate):

$$F(x,y) \leftarrow \texttt{knows}(x,y), \texttt{helps}(x,y)$$
$$Ans(x,y) \leftarrow F^+(x,y)$$

Now, let us say that a person p' is an *acquaintance* of p if p knows p' and they have an indirect friend in common. The pairs of person connected by a chain of acquaintances can be expressed by the following RQ:

$$F(x,y) \leftarrow \texttt{knows}(x,y), \texttt{helps}(x,y)$$
$$A(x,y) \leftarrow \texttt{knows}(x,y), F^+(x,z), F^+(y,z)$$
$$Ans(x,y) \leftarrow A^+(x,y)$$

Our main goal in this part is to understand the complexity of RQs. The complexity of a query language is typically measured in terms of query evaluation, but there are other fundamental problems that need to be considered. One prominent example is the *containment* problem, which asks whether a query q is *contained* in another query q', that is, the answer of q is contained in the answer of q', over *all* databases. Checking containment is one of the most fundamental static analysis tasks in databases and it is used in several contexts, such as query optimization [40, 32, 43, 120], view-based query answering [42, 117, 37], integrity constraint checking [90, 70], and knowledge representation systems based on description logics where it is at the heart of all relevant reasoning tasks [62, 119, 118]. For instance, the classes of UCQs and UC2RPQs have a decidable containment problem (NP-complete [40] and EXPSPACE-complete, respectively), while query containment for Datalog is undecidable [144]. This has motivated a large body of research identifying expressive fragments of Datalog with a decidable containment problem [23, 25, 26, 54, 38, 33, 44, 140].

It is not hard to see that the complexity of evaluation of RQs is the same as for UC2RPQs: NP-complete in general and NLOGSPACE-complete in *data complexity* [151], that is, when we consider the query to be fixed. This is a direct consequence of the fact that RQs are subsumed by binary *linear* Datalog [44, 52]. Nevertheless, the precise complexity of checking containment of RQs has been open so far. Decidability for this problem can be easily derived from Courcelle's theorem [55, 56], but this yields a *nonelementary* complexity upper bound, that is, a tower of exponentials of unbounded length. It is not known whether this bound can be improved to an elementary bound, as in the case of UC2RPQs. Given the importance of the containment problem, a nonelementary lower bound for containment of RQs would suggest that the class may be too powerful to be useful in practice. Thus we are interested in the following research questions:

What is the precise complexity of the containment problem for RQs? Is this problem still elementary, as in the case of UC2RPQs?

There are several other languages that are either more expressive or incomparable to RQs. One of the oldest is *GraphLog* [52], which is equivalent to *first-order logic with transitive closure* [121]. More recent languages include *extended* CRPQs [13], which extends CRPQs with *path variables*; the powerful *Walk Logic* [99], which has been proposed as an alternative

to extended CRPQs for expressing path properties; the conjunctive *context-free* path queries [98]; XPath for graph databases [122, 111]; and graph languages based on algebras over binary relations [72, 148, 123]. Although all these languages have interesting evaluation properties, the containment problem for all of them is undecidable.

Another body of research has focused on fragments of Datalog with decidable containment problems [23, 25, 26, 140]. In fact, RQs were investigated in [25, 26] (under the name of *nested positive* 2RPQs), but the precise complexity of checking containment was left open, with non-elementary tight bounds provided only for strict generalizations of RQs. Interestingly, the containment problem is non-elementary even for *positive* first-order logic with *unary* transitive closure [25, 26], which is a slight generalization of RQs.

Although query containment is a key problem in database theory, its practical usefulness is still very much an open question. As mentioned at the end of Section 1.1.1, the standard approach for optimizing the evaluation of queries, like UCQs, is cost-based optimization [4, 41], where we search for a low-cost plan amongst several alternative plans and their estimated costs. However, as we also discussed in Section 1.1.1, there are no standard optimization techniques for UC2RPQs and more expressive navigational queries, and therefore, algorithms for query containment may have a great impact in practice. In particular, an elementary upper bound for the containment of RQs would show that RQs form a well-behaved class of graph queries, and thus an interesting candidate for a general-purpose navigational query language (see [153] for a recent discussion about RQs and query containment).

The main result in this part is a positive answer to the research question posed above. Specifically, we show that containment of RQs is elementary. We also pinpoint out the precise complexity of this problem and show that it is 2Expspace-complete. Our proof is based on new methods that combine database and automata techniques.

1.2 Contributions

Fixed-parameter tractable evaluation of UC2RPQs

Regarding the first part of this thesis, our main contribution is to show that evaluation of UC2RPQs of bounded treewidth modulo equivalence is fixed-parameter tractable. As we mentioned, this was already known for UCQs, and actually in that case, polynomial time tractability can be achieved. The techniques used to prove this come from the area of *constraint satisfaction* [59, 86] and they are especially tailored to work with UCQs. Due to the recursive nature of UC2RPQs, it is not clear how to apply those techniques in this case. Thus, in order to prove our result, we need to develop new methods based on homomorphisms and automata techniques. A crucial ingredient in our proof is the following technical lemma that may be of independent interest (see Lemma 4.10 in Section 4.2):

Lemma 1.1 *Let $k \geq 1$ be a positive integer. There is an exponential space algorithm that given a UC2RPQ Γ computes a UC2RPQ Γ' such that (i) Γ' is of treewidth at most $2k+1$;*

(ii) if Γ *is equivalent to a UC2RPQ of treewidth at most* k*, then* Γ' *is equivalent to* Γ*; and (iii)* $|\Gamma'|$ *is at most exponential in* $|\Gamma|$*.*

Our main result follows easily from Lemma 1.1. Indeed, fix $k \geq 1$ and let $\mathcal{G}$ be a graph database, $\bar{n}$ be a tuple of nodes in $\mathcal{G}$, and Γ be a UC2RPQ that is equivalent to a UC2RPQ of treewidth at most k. In order to check whether $\bar{n}$ is in the answer of Γ over $\mathcal{G}$, we can compute Γ' from Lemma 1.1 and then check whether $\bar{n}$ is in the answer of Γ' over $\mathcal{G}$. Computing Γ' from Γ can be done using exponential space, and therefore, it takes at most double exponential time. Since the treewidth of Γ' is bounded by the constant $2k+1$, evaluating Γ' takes polynomial time, that is, $O(|\Gamma'|^c \cdot |\mathcal{G}|^c)$ time, where $c \geq 1$ is a constant. Since $|\Gamma'|$ is at most exponential in $|\Gamma|$, this is $O(2^{|\Gamma|^d} \cdot |\mathcal{G}|^c)$, for constants $c, d \geq 1$. Therefore, the overall running time of this algorithm is $O(g(|\Gamma|) + 2^{|\Gamma|^d} \cdot |\mathcal{G}|^c)$, where g is a double exponential function, which is fixed-parameter tractable.

We also carry out a deeper investigation for the case of UC2RPQs of bounded treewidth modulo equivalence when the bound on the treewidth equals 1, namely, *semantically acyclic* UC2RPQs (we use this name as treewidth 1 coincides with the traditional notion of acyclicity). In this case, we can refine Lemma 1.1 and show the following:

Lemma 1.2 *There is an exponential space algorithm that given a UC2RPQ* Γ *computes a UC2RPQ* Γ' *such that (i)* Γ' *is of treewidth* 1*; (ii) if* Γ *is equivalent to a UC2RPQ of treewidth* 1*, then* Γ' *is equivalent to* Γ*; and (iii)* $|\Gamma'|$ *is at most exponential in* $|\Gamma|$*.*

Observe that Lemma 1.2 is tight in the sense that the treewidth of Γ' and Γ in condition (ii) coincide (both have treewitdh 1), whereas in Lemma 1.1 there is a gap between k and $2k+1$. This allows us to prove further results regarding *acyclic approximations* of UC2RPQs and recognizability of semantically acyclic UC2RPQs. The results about acyclic approximations are motived by recent work on acyclic approximations of UCQs [14, 15], and we defer the technical details to Section 3.4. Regarding recognizability, in order to check whether a given UC2RPQ Γ is semantically acyclic, that is, it is equivalent to a UC2RPQ of treewidth 1, we can compute Γ' from Lemma 1.2 and then check whether Γ' is equivalent to Γ. As containment of UC2RPQs can be checked in Expspace [33], and $|\Gamma'|$ is at most exponential in $|\Gamma|$, it follows that recognizability of semantically acyclic UC2RPQs is in 2Expspace. By refining previous automata techniques [33, 137, 18], we improve this 2Expspace upper bound to Expspace, and also provide a matching lower bound. Surprisingly, once we move from treewidth 1 to treewidth 2, our techniques do not apply, specifically, we do not have a tight version of Lemma 1.1. Whether recognizability of treewidth k modulo equivalence UC2RPQs is decidable, when $k > 1$, is left as an interesting open question.

All the results regarding semantic acyclicity appear in the journal article *Semantic acyclicity on graph databases*, accepted for publication in the *SIAM Journal on Computing* (SICOMP) [19]. A previous version of this article, published in the *ACM Symposium on Principles of Database Systems* (PODS) in 2013 [17], includes most results except for the improved Expspace upper bound for recognizing semantically acyclic UC2RPQs. Both articles are coauthored with Pablo Barceló and Moshe Vardi. Our main result that bounded treewidth modulo equivalence UC2RPQs are fixed-parameter tractable is presented here for the first time.

Containment of regular queries

Our main contribution in the second part of this thesis is to show that the containment problem for RQs is 2Expspace-complete, thus in particular, it has elementary complexity. We attack this problem by considering an equivalent query language, called *nested* UC2RPQs (nUC2RPQs), which is of independent interest. An nUC2RPQ is basically an RQ where each extensional predicate could be a 2RPQ. We show that the containment problem for nUC2RPQs is in 2Expspace, and as a consequence we obtain a 2Expspace upper bound for containment of RQs. We also provide matching lower bounds.

Our proofs are based on automata-theoretic techniques. In particular, the 2Expspace upper bound is shown in two stages. First, we reduce the containment of two nUC2RPQs into containment of, essentially, an RPQ in an nUC2RPQ. The reduction is based on a *serialization* technique, where we show how to represent expansions of nUC2RPQs as strings. We then proceed to tackle the reduced containment problem. Here we exploit techniques used before, e.g., in [33, 38, 44] to show that containment of UC2RPQs is in Expspace. Nevertheless, our proof requires a deep understanding and a significant refinement of these techniques. The essence of the proof is a suitable representation of the *partial mappings* of nUC2RPQs into strings, that we call *cuts*, and that allows us to significantly extend the automata notions of previous work. This representation is robust against nesting and does not involve a non-elementary blow up in size.

We also investigate some interesting restrictions and extensions of RQs. First we consider RQs of bounded treewidth. Intuitively, for $k \geq 1$, an RQ has treewidth at most k, if each rule has treewidth at most k. As it turns out, RQs of bounded treewidth can be evaluated in polynomial time, as in the case of UC2RPQs [13]. Second, we consider RQs of bounded *depth* (of nesting) and we show that the containment problem for these is Expspace-complete, the same as for UC2RPQs. Finally, we focus on the arity of the predicates involved in RQs. By definition, all intensional predicates in an RQ have arity 2. We show that containment of RQs is still 2Expspace-complete if we relax this condition and allow unbounded arity of intensional predicates, but the evaluation problem now becomes Pspace-complete. Interestingly, we also show that 2Expspace-completeness remains even when the arity of the extensional predicates is unbounded, that is, when we deal with relational databases instead of graph databases.

All the results regarding RQs appears in the journal article *Regular queries on graph databases*, accepted for publication in the *ACM Transactions on Computer Systems* (TOCS) [138]. This is an extended version of a conference paper published in the *International Conference on Database Theory* (ICDT) in 2015 [137]. Both articles are coauthored with Juan Reutter and Moshe Vardi.

Chapter 2

Background

In this chapter we present the basic definitions and terminology used throughout the thesis. We also review known complexity results for some fundamental query languages. We assume familiarity with first order logic [121] and formal language theory [146] (regular languages, finite automata and regular expressions). Also, we assume familiarity with complexity theory [146, 11], specifically, standard complexity classes such as PTIME, NP, PSPACE, EXPTIME and others, and completeness under polynomial time (many to one) reductions.

In Section 2.1 we introduce the class of conjunctive queries (CQs), which is one of the most well-studied class of relational queries. We also present its extension with the union operator, namely, the class of unions of CQs (UCQs). Then, in Section 2.2, we present some fundamental classes of navigational queries over graph databases, namely, the class of regular path queries (RPQs) and all of its basic extensions: two-way regular path queries (2RPQs), conjunctive 2RPQs (C2RPQs) and unions of C2RPQs (UC2RPQs).

Finally, in Section 2.3, we define the most fundamental problems associated with any query language, namely, the evaluation and the containment problem. Most results in this thesis focus on the complexity of these two problems for particular graph query languages. We conclude in Section 2.3.1 and 2.3.2 by presenting known complexity results for (U)CQ and (U)C2RPQs.

2.1 Relational Databases and UCQs

A *relational schema* (or *schema* for short) is a finite set σ of predicates or relation symbols. Every predicate R in σ has a specific *arity* $r > 0$. A *relational database* (or *database* for short) $\mathcal{D}$ over schema σ consists of a *finite* domain D and a relation $R^{\mathcal{D}} \subseteq D^r$, for every predicate $R \in \sigma$, where r is the arity of R. If $\bar{t} \in R^{\mathcal{D}}$ for a predicate R, then we say that $R(\bar{t})$ is a *fact* of the database $\mathcal{D}$. We always assume that D is the *active* domain of $\mathcal{D}$, that is, it contains precisely all those elements that occur in some relation $R^{\mathcal{D}}$, with $R \in \sigma$. For simplicity, we usually denote the set of elements D by the name of the database $\mathcal{D}$.

A *conjunctive query* (CQ) [40] over schema σ is a logical formula in the $\exists, \wedge$-fragment of first-order logic, i.e., a formula of the form:

$$\theta(\bar{x}) \;=\; \exists \bar{y}\ (R_1(\bar{y}_1) \wedge \cdots \wedge R_m(\bar{y}_m)) \tag{2.1}$$

where each R_i is a predicate from σ and $\bar{y}_i$ is a tuple of variables among $\bar{x}, \bar{y}$ whose length is the arity of R_i. As usual, we assume that $\bar{x}$ are the *free* variables of θ, i.e. the variables mentioned in θ that are not existentially quantified. Each $R_i(\bar{y}_i)$ is an *atom* of θ. We adopt a rule-based notation as used, for instance, with the Datalog query language [4] and write a CQ θ of the form (2.1) as a rule of the form:

$$\theta(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m) \tag{2.2}$$

As customary, we define the semantics of CQs in terms of *homomorphisms* [4, 97]. Let θ be a CQ of the form (2.2) and $\mathcal{D}$ a database over schema σ. A homomorphism h from θ to $\mathcal{D}$ is a mapping from the variables of θ to the elements of $\mathcal{D}$ such that $R_i(h(\bar{y}_i))$ is a fact in $\mathcal{D}$, for each $1 \leq i \leq m$. The *answer* of θ over $\mathcal{D}$, denoted $\theta(\mathcal{D})$, is the set of all tuples of the form $h(\bar{x})$, for h a homomorphism from θ to $\mathcal{D}$.

Alternatively, we can define the semantics of CQs in terms of homomorphisms between databases. If $\mathcal{D}$ and $\mathcal{D}'$ are databases over schema σ, then a homomorphism h from $\mathcal{D}$ to $\mathcal{D}'$ is a mapping from the elements of $\mathcal{D}$ to the elements of $\mathcal{D}'$ such that, for each $R \in \sigma$ and tuple $\bar{t}$, $R(h(\bar{t}))$ is a fact in $\mathcal{D}'$ whenever $R(\bar{t})$ is a fact in $\mathcal{D}$. For a CQ θ over σ, we define its *canonical database* $\mathcal{D}^\theta$ as the database over σ whose elements are the variables of θ and where $R(\bar{z})$ is a fact iff $R(\bar{z})$ is an atom of θ. Then the answer $\theta(\mathcal{D})$ of $\theta(\bar{x})$ over $\mathcal{D}$ coincides with the set of tuples of the form $h(\bar{x})$, for h a homomorphism from $\mathcal{D}^\theta$ to $\mathcal{D}$.

A *union* of conjunctive queries (UCQ) $\Theta(\bar{x})$ over schema σ is a non-empty set $\{\theta_1(\bar{x}), \ldots, \theta_n(\bar{x})\}$ of CQs over σ with the same free variables. Each θ_j is a *disjunct* of Θ. The *answer* $\Theta(\mathcal{D})$ of Θ over a database $\mathcal{D}$ is $\bigcup_{1 \leq j \leq n} \theta_j(\mathcal{D})$.

For a *Boolean* CQ or UCQ (i.e. the tuple $\bar{x}$ of free variables is the empty tuple), the answer `true` is, as usual, modeled by the set containing the empty tuple, and the answer `false` by the empty set.

2.2 Graph databases and UC2RPQs

A *graph database* [8, 57, 12] is a finite edge-labeled graph. Formally, let Σ be a finite alphabet. A graph database $\mathcal{G}$ over Σ is a pair (N, E), where N is a finite set of nodes and $E \subseteq N \times \Sigma \times N$. Thus, each edge in $\mathcal{G}$ is a triple $(n, a, n') \in N \times \Sigma \times N$, whose interpretation is an a-labeled edge from n to n' in $\mathcal{G}$.

When querying graph databases, we are particularly interested in *navigational* queries. Next we review the core features of navigational graph query languages (for good surveys, see [157, 12]). The basic querying mechanism is provided by the class of *regular path queries* (RPQs) [57, 37]. An RPQ $\mathcal{A}$ over finite alphabet Σ is a nondeterministic finite automaton

(NFA) over Σ. Let $\mathcal{G} = (N, E)$ be a graph database over Σ. A *path* π from node n to n' in $\mathcal{G}$ is a sequence

$$\pi \;=\; n_0 a_1 n_1 a_2 n_2 \dots v_{k-1} n_k v_k,$$

where $k \geq 0$, $n_0 = n$, $n_k = n'$, and for each $1 \leq i \leq k$ it is the case that (n_{i-1}, a_i, n_i) is an edge in E. Notice that when $k = 0$ this path consists of the single node n_0. The *label* of π, denoted $\mathsf{label}(\pi)$, is the word $a_1 a_2 \dots a_k \in \Sigma^*$. Note that if $k = 0$, then $\mathsf{label}(\pi)$ is the empty word ε. The answer $\mathcal{A}(\mathcal{G})$ of the RPQ $\mathcal{A}$ over $\mathcal{G}$ is the set of all pairs $(n, n') \in N \times N$ such that there is a path π in $\mathcal{G}$ from n to n' for which it is the case that the word $\mathsf{label}(\pi)$ is accepted by $\mathcal{A}$.

The class of *two-way regular path queries* (2RPQs) [34, 37] extends RPQs with backward traversals of edges. A 2RPQ over Σ is a nondeterministic finite automaton (NFA) over the alphabet $\Sigma^{\pm}$ that extends Σ with the symbol a^- for each $a \in \Sigma$. Intuitively, a^- represents the inverse of a. For a graph database $\mathcal{G} = (N, E)$ over Σ, we define its *completion* $\mathcal{G}^{\pm}$ as the graph database over $\Sigma^{\pm}$ obtained from $\mathcal{G}$ by adding the edge (n', a^-, n), for each edge $(n, a, n') \in E$. The answer $\mathcal{A}(\mathcal{G})$ of the 2RPQ $\mathcal{A}$ over $\mathcal{G}$ is the set of all pairs $(n, n') \in N \times N$ such that there is a path π in $\mathcal{G}^{\pm}$ from n to n' for which it is the case that the word $\mathsf{label}(\pi)$ is accepted by $\mathcal{A}$. In other words, to evaluate $\mathcal{A}$ over $\mathcal{G}$, we simply consider $\mathcal{A}$ as an RPQ over $\Sigma^{\pm}$ and evaluate it over $\mathcal{G}^{\pm}$.

When the expressive power of 2RPQs is combined with the ability of CQs to express conjunctions and existential quantification, it yields a powerful class of queries, namely, the *conjunctive two-way regular path queries*, or C2RPQs [33, 73]. Formally, a C2RPQ over Σ is an expression of the form:

$$\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1, y_1'), \dots, \mathcal{A}_m(y_m, y_m'), \tag{2.3}$$

where each $\mathcal{A}_i$ is a 2RPQ over Σ, for $1 \leq i \leq m$. A CRPQ is a C2RPQ without inverses, i.e. a C2RPQ of the form (2.3) in which each $\mathcal{A}_i$ ($1 \leq i \leq m$) is an RPQ over Σ. As usual, we assume that $\bar{x}$ are the *free* variables of γ, i.e. the variables mentioned in γ that are not existentially quantified. Each $\mathcal{A}_i(y_i, y_i')$ is an *atom* of γ. The semantics of C2RPQs is defined by using a notion of homomorphism that maps atoms of γ into pairs that satisfy the corresponding 2RPQs. Given γ of the form (2.3) and a graph database $\mathcal{G} = (N, E)$, a homomorphism from γ to $\mathcal{G}$ is a mapping $h : \bigcup_{1 \leq i \leq m} \{y_i, y_i'\} \to N$ such that $(h(y_i), h(y_i')) \in \mathcal{A}_i(\mathcal{G})$ for every $1 \leq i \leq m$. We then define the answer $\gamma(\mathcal{G})$ of γ over $\mathcal{G}$ to be the set of all tuples $h(\bar{x})$ such that h is a homomorphism from γ to $\mathcal{G}$.

A union of C2RPQs (UC2RPQ) $\Gamma(\bar{x})$ over Σ is a non-empty set $\{\gamma_1(\bar{x}), \dots, \gamma_n(\bar{x})\}$ of C2RPQs over Σ with the same free variables. Each γ_i is a *disjunct* of Γ. The answer $\Gamma(\mathcal{G})$ of Γ over a graph database $\mathcal{G}$ is $\bigcup_{1 \leq j \leq n} \gamma_j(\mathcal{G})$. As before, for a Boolean (U)C2RPQ (i.e. $\bar{x}$ is the empty tuple), the answer is `false` iff it is the empty set.

We also consider (U)CQs over graph databases. These were already defined in Section 2.1 for relational databases, whereas graph databases are not formally relational databases. However, there is a natural way of viewing graph databases as relational databases: if Σ is a finite alphabet, we can define the schema $\sigma(\Sigma)$ that contains a binary predicate E_a for each $a \in \Sigma$. Then, the graph database $\mathcal{G}$ over Σ is represented by the relational database $\mathcal{D}(\mathcal{G})$, over the schema $\sigma(\Sigma)$ that consists of all facts of the form $E_a(n, n')$ such that (n, a, n') is an

edge in $\mathcal{G}$. Now a (U)CQ over Σ is simply a (U)CQ over $\sigma(\Sigma)$. The answer of a (U)CQ Θ over a graph database $\mathcal{G}$ is $\Theta(\mathcal{D}(\mathcal{G}))$. For simplicity, we identify the predicate E_a with the symbol a, for each $a \in \Sigma$.

Remark 2.1 While we use NFAs in order to specify regular languages in UC2RPQs, it is sometimes more convenient (especially in practice) to specify such languages using regular expressions. In fact, this is the approach often followed in the graph database literature (see, e.g., [12, 157]). Each regular expression can be easily translated into an NFA in polynomial time, and, therefore, all the complexity upper bounds obtained in this thesis continue to hold in the case when UC2RPQs are specified using regular expressions. The reason why we use NFA instead is that they allow to express regular languages more succinctly than the latter (which is crucial for some of our results). For the sake of readability though, all the examples we present in the paper are specified using regular expressions.

Example 2.2 Consider a graph database $\mathcal{G} = (N, E)$ of researchers, papers, conferences and journals over the alphabet $\Sigma = \{\texttt{creator}, \texttt{inJournal}, \texttt{inConf}\}$. The set of edges E consists of the following:

- All tuples $(r, \texttt{creator}, p)$ such that r is a researcher that (co)authors paper p.
- Each tuple $(p, \texttt{inJournal}, j)$ such that paper p appeared in journal j.
- All tuples $(p, \texttt{inConf}, c)$ such that paper p was published in conference c.

Consider the C2RPQ $\gamma(x, y)$ defined as:

$$\gamma(x, y) \leftarrow \texttt{creator}(x, z), \texttt{inConf}(z, w), \texttt{creator}^-(z, y)$$

Its evaluation $\gamma(\mathcal{G})$ over the graph database $\mathcal{G}$ consists of the set of pairs (n, n') such that researchers n and n' have a joint conference paper.

The evaluation over $\mathcal{G}$ of the C2RPQ $\gamma'(x, y)$ defined as:

$$\gamma'(x, y) \leftarrow (\texttt{creator} \cdot \texttt{creator}^-)^*(x, y), \texttt{creator}(y, z), \texttt{inJournal}(z, w)$$

consists of the set of pairs (n, n') of researchers that are linked by a coauthorhsip sequence where n' has at least one journal paper. □

2.3 Query evaluation and containment

The *evaluation* problem and the *containment* problem are two fundamental problems associated with any query language [4]. In this thesis, we study query languages with respect to the complexity of these two problems.

Query evaluation As it is customary in database theory [12, 131, 151], in order to analyze the complexity of query evaluation, we study how difficult is to recognize whether a given tuple belongs to the answer of a query over a database. In other words, we look at the problem

as a *decision* problem, instead of looking at it as the *computation* problem of enumerating all the tuples in the answer.

Let $\mathcal{L}$ be a graph query language (e.g., the class of UC2RPQs), then the evaluation problem for $\mathcal{L}$ is defined as follows:

PROBLEM :	*Evaluation for* $\mathcal{L}$
INPUT :	A query $q \in \mathcal{L}$, a graph database $\mathcal{G}$ and a tuple $\bar{n}$ of nodes of $\mathcal{G}$.
QUESTION :	Does $\bar{n}$ belong to $q(\mathcal{G})$?

If $\mathcal{L}$ is a relational query language (e.g., the class of UCQs), then the evaluation problem for $\mathcal{L}$ is defined similarly, but now we are given in the input a relational database $\mathcal{D}$ instead of a graph database.

Query containment Let q and q' be two queries over relational (resp. graph) databases. We say that q is *contained* in q', denoted by $q \subseteq q'$, if $q(\mathcal{D}) \subseteq q'(\mathcal{D})$, for *all* relational (resp. graph) databases $\mathcal{D}$. The containment problem asks whether q is contained in q', for given queries q and q'. More formally, let $\mathcal{L}$ be a query language. The containment problem for $\mathcal{L}$ is defined as follows:

PROBLEM :	*Containment for* $\mathcal{L}$
INPUT :	Queries $q, q' \in \mathcal{L}$.
QUESTION :	Is q contained in q'?

2.3.1 Basic complexity results for (U)CQs

In a seminal paper, Chandra and Merlin [40] showed that CQ evaluation and CQ containment are equivalent problems (recall that $\mathcal{D}^\theta$ denotes the canonical database of the CQ θ; see Section 2.1):

Proposition 2.3 [40] *Let $\theta(\bar{x})$ and $\theta'(\bar{x})$ be two CQs over the same schema. Then, θ is contained in θ' if and only if $\bar{x} \in \theta'(\mathcal{D}^\theta)$.*

As a consequence, they showed that CQ evaluation and CQ containment are NP-complete problems. To see this, let $\theta(\bar{x})$ be a CQ, $\mathcal{D}$ be a database, and $\bar{a}$ a tuple. An NP algorithm solving CQ evaluation guesses a mapping h from the variables of θ to the elements of $\mathcal{D}$ and then verifies in polynomial time whether h is a homomorphism from θ to $\mathcal{D}$ and $h(\bar{x}) = \bar{a}$. For the NP-hardness, we can reduce, for instance, from the 3-Colorability problem, which is a well-known NP-complete problem [76]. Indeed, let G be a graph. We define a Boolean CQ θ_G that naturally represents G: θ_G contains one variable x_v for each node v in G and there are atoms $E(x_v, x_{v'})$ and $E(x_{v'}, x_v)$ for each edge $\{v, v'\}$ in G. We also define the database K_3, whose domain is $\{1, 2, 3\}$ and where E^{K_3} is the disequality relation

$$E^{K_3} = \{(1,2), (1,3), (2,1), (2,3), (3,1), (3,2)\}$$

It can be verified that G is 3-colorable if and only if $\theta_G(K_3) = \texttt{true}$. Note that this

additionally shows that the problem is NP-hard even for Boolean CQs.

Theorem 2.4 [40] *The evaluation and containment problem for CQs are* NP*-complete. Both problems remain* NP*-hard even when the input is restricted to Boolean CQs.*

UCQ evaluation can be easily reduced to CQ evaluation, thus the former problem is still NP-complete. Regarding UCQ containment, Sagiv and Yannakakis [141] showed that containment between UCQs can be characterized in terms of containment of CQs:

Proposition 2.5 [141] *Let $\Theta = \{\theta_1, \ldots, \theta_n\}$ and $\Theta' = \{\theta'_1, \ldots, \theta'_\ell\}$ be UCQs. Then $\Theta \subseteq \Theta'$ if and only if for each $1 \leq i \leq n$, there exists $1 \leq j \leq \ell$ such that $\theta_i \subseteq \theta'_j$.*

It follows that UCQ containment reduces to CQ containment and thus the former problem is also NP-complete. In summary, we have the following:

Theorem 2.6 *The evaluation and containment problem for UCQs are* NP*-complete.*

2.3.2 Basic complexity results for (U)C2RPQs

We start by considering the evaluation problem. As it turns out, we can evaluate 2RPQs in polynomial time and actually in linear time in both the size of the query and the database.

Proposition 2.7 *The evaluation problem for 2RPQs can be solved in linear time, that is, given a 2RPQ $\mathcal{A}$, a graph database $\mathcal{G}$ and a pair of nodes (n, n'), checking whether $(n, n') \in \mathcal{A}(\mathcal{G})$ can be done in $O(|\mathcal{A}| \cdot |\mathcal{G}|)$ time.*

Next, we briefly sketch the algorithm from Proposition 2.7 (for more details, see e.g. [12]). Let $\mathcal{A}$ be a 2RPQ, $\mathcal{G}$ be a graph database both over an alphabet Σ, and (n, n') a pair of nodes of $\mathcal{G}$. Recall that $\mathcal{G}^{\pm}$ denotes the completion of $\mathcal{G}$, that is, the database over $\Sigma^{\pm}$ obtained from $\mathcal{G}$ by adding edges (n', a^-, n), for each edge (n, a, n') in $\mathcal{G}$. Let $\mathcal{A}_{\mathcal{G}^{\pm}}(n, n')$ be the NFA over $\Sigma^{\pm}$ whose transition graph is precisely $\mathcal{G}^{\pm}$ and whose initial and final state is n and n', respectively. Observe that $\mathcal{A}_{\mathcal{G}^{\pm}}(n, n')$ accepts precisely all the words w such that $w = \mathsf{label}(\pi)$, for some path π from n to n' in $\mathcal{G}^{\pm}$. It follows that $(n, n') \in \mathcal{A}(\mathcal{G})$ iff there is a word accepted by $\mathcal{A}$ and $\mathcal{A}_{\mathcal{G}^{\pm}}(n, n')$ simultaneously. The latter condition holds iff the language accepted by the *product* NFA $\mathcal{A} \times \mathcal{A}_{\mathcal{G}^{\pm}}(n, n')$ is nonempty [146]. This can be checked in time $O(|\mathcal{A} \times \mathcal{A}_{\mathcal{G}^{\pm}}(n, n')|) = O(|\mathcal{A}| \cdot |\mathcal{A}_{\mathcal{G}^{\pm}}(n, n')|) = O(|\mathcal{A}| \cdot |\mathcal{G}|)$, via a standard reachability test.

Using the same idea as above, we can show that computing $\mathcal{A}(\mathcal{G})$ for a 2RPQ $\mathcal{A}$ and a graph database $\mathcal{G}$ can be done in time $O(|\mathcal{A}| \cdot |\mathcal{G}|^2)$ (see e.g. [12]).

Proposition 2.8 *Given a 2RPQ $\mathcal{A}$ and a graph database $\mathcal{G}$, computing $\mathcal{A}(\mathcal{G})$ can be done in $O(|\mathcal{A}| \cdot |\mathcal{G}|^2)$ time.*

Combining the NP algorithm for UCQ evaluation and Proposition 2.7, we can obtain an

NP upper bound for UC2RPQ evaluation. This problem is already NP-hard for CQs.

Proposition 2.9 *The evaluation problem for UC2RPQs is* NP*-complete.*

The situation for containment is different. Let $\mathcal{A}$ and $\mathcal{A}'$ be two RPQs over the same alphabet Σ. It can be proved that $\mathcal{A}$ is contained in $\mathcal{A}'$ if and only if the language accepted by $\mathcal{A}'$ is contained in the language accepted by $\mathcal{A}'$. Containment of NFAs is a well-known PSPACE-complete problem [147]. By using *two-way* automata, it was shown in [37] that containment is still in PSPACE for 2RPQs.

Theorem 2.10 [37] *The containment problem for 2RPQs is* PSPACE*-complete. It remains* PSPACE*-hard even if restricted to RPQs.*

For UC2RPQs the complexity is higher:

Theorem 2.11 [33] *The containment problem for UC2RPQs is* EXPSPACE*-complete. It remains* EXPSPACE*-hard even if restricted to Boolean CRPQs.*

The EXPSPACE algorithm behind Theorem 2.11 is again based on automata techniques [33], more specifically, it reduces UC2RPQ containment to nonemptiness of a double exponential sized NFA.

Part I

Fixed-parameter tractable evaluation of UC2RPQs

Chapter 3

Semantically acyclic UC2RPQs

We start our investigation of UC2RPQs of bounded treewidth modulo equivalence by considering the most simple case, i.e., when the bound on the treewidth is 1. We refer to this class as semantically acyclic UC2RPQs. Our main result in this chapter is that evaluation of semantically acyclic UC2RPQs is fixed-parameter tractable.

We first consider semantic acyclicity in the context of relational databases and UCQs. We study the traditional notion of acyclicity introduced by Yannakakis [160] (also known as α-acyclicity [65]), whose definition is based on *acyclic hypergraphs*, or alternatively, on *join trees* [80]. When restricted to graph databases, this notion coincides with treewidth 1, but on arbitrary relational databases it generalizes treewidth 1, thus the results we present are stronger. Although these results follow from known techniques in the area of constraint satisfaction problems (CSP) [59, 48], we state them for the sake of completeness as they will help us developing the necessary intuitions for the more complicated case of graph databases and UC2RPQs.

Then we turn to the study of semantically acyclic UC2RPQs. To illustrate the usefulness of semantic acyclicity, consider an alphabet $\{p, f\}$ modeling social networks. Labels p and f stand for "person" and "friend", respectively. An object in a graph database is a person if it has an outgoing p-labeled edge (e.g., to its id). Two persons are friends if there is an f-labeled edge between them. Suppose we want to know whether there is a group of persons of size $n \geq 2$ in the graph database $\mathcal{G}$ that forms a *clique*. This means that there is an f-labeled edge between each pair of persons in the group. Evaluating this query basically corresponds to the well-known *clique problem*, which is NP-complete [76]. Typical algorithms for this problem take $|\mathcal{G}|^{O(n)}$ time.

On the other hand, in the context of graph databases it is sometimes more interesting knowing whether two persons are linked by a path labeled in the friendship relationship than if they are actually friends. This means that there is a sequence of mutual friends that connects these two persons, or, more formally, that there is a path between them labeled by the regular expression f^+. Our clique query can thus also be relaxed by using this condition. The left-hand side graph in Figure 3.1 depicts the result for $n = 4$ (dots represent variables and arrows represent labeled atoms). Notice that this query is non acyclic, and by applying

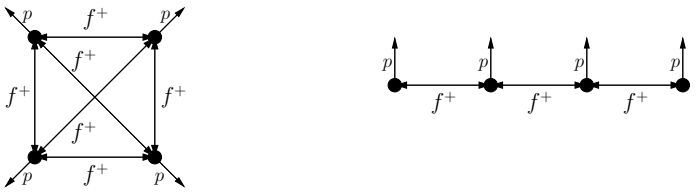

Figure 3.1: A semantically acyclic CRPQ and its equivalent acyclic query

standard algorithms we can still evaluate it in $|\mathcal{G}|^{O(n)}$ time. However, it is not hard to prove that the query is semantically acyclic, as it is equivalent to a *path* of n persons that are consecutively linked by paths of mutual friends (the case when $n = 4$ is depicted in the right-hand side of Figure 3.1). Indeed, note first that the clique query is trivially contained in the path query. Conversely, whenever there is a path in a graph database witnessing the path query, the non-consecutive nodes in this path are also connected by paths of mutual friends since we can concatenate the paths between consecutive nodes. Hence, the path is also a witness for the clique query. Observe that the existence of such a path can be checked in time $O(n \cdot |\mathcal{G}|^2)$. Therefore, evaluating a semantically acyclic query via its equivalent acyclic query can significantly improve the running time.

In order to address semantic acyclicity, we consider first the problem of UC2RPQ *approximations*, which is motivated by recent work on approximations of UCQs [14, 15]. Typical algorithms require $|\mathcal{D}|^{O(|\Theta|)}$ time to evaluate a UCQ Θ on a database $\mathcal{D}$, which might be prohibitively expensive for a large database $\mathcal{D}$ even if Θ is small. This led the idea of finding *approximations* of UCQs in tractable classes [15], in particular, in the class of acyclic UCQs. Intuitively, an acyclic UCQ Θ' is an approximation of a UCQ Θ if Θ' is contained in Θ and it is "as close as possible" to Θ in the class of acyclic UCQs. The latter condition means that Θ' is a maximal acyclic UCQ that is contained in Θ. It follows from techniques in [15] that UCQs have good properties in terms of acyclic approximations: Each UCQ Θ has a unique acyclic approximation (up to equivalence) and this approximation can be computed in single-exponential time. Since acyclic UCQs can be evaluated in linear time in both the query and the data, these good properties imply that computing and running the acyclic approximation of a UCQ Θ on a database $\mathcal{D}$ takes time $O(2^{|\Theta|^c} \cdot |\mathcal{D}|)$, for some constant $c \geq 1$. This is much faster than $|\mathcal{D}|^{O(|\Theta|)}$ on large databases. Thus, if the quality of the approximation is good, we may prefer to run this faster query instead of Θ.

We show that the good properties of UCQs in terms of acyclic approximations extend to UC2RPQs. In particular, we show that each UC2RPQ Γ has a unique acyclic approximation (up to equivalence) and that an approximation of exponential size can be computed in Expspace. The data complexity of evaluating this approximation is then quadratic in the size of the data, which is the same the data complexity of 2RPQs (recall that in data complexity, the query in considered to be fixed [151]). This shows that acyclic approximations might be useful when evaluating the original query is infeasible, though the cost of computing the approximation is quite high. We also show that UC2RPQs behave provably worse than UCQs in terms of approximations: Verifying whether an acyclic UCQ Θ' is the approximation of the UCQ Θ is in the second-level of the polynomial hierarchy, but it becomes Expspace-complete in the case of UC2RPQs. This is not surprising, as it is known

that checking containment of UC2RPQs is EXPSPACE-complete [33].

After this, we studying our main problem, namely, evaluation of semantically acyclic UC2RPQs. The results on acyclic approximations imply that, in order to evaluate a semantically acyclic UC2RPQ Γ, we can compute and evaluate its acyclic approximation Γ', as Γ and Γ' are equivalent in this case. We show that this gives us a fixed-parameter tractable algorithm for evaluating semantically acyclic UC2RPQs.

We conclude by proving further results about verification of semantically acyclic UC2RPQs. We show that verifying whether a UC2RPQ is semantically acyclic is decidable. As noted above, we can construct in EXPSPACE an exponential-sized acyclic approximation Γ' of a given UC2RPQ Γ. By construction, Γ' is contained in Γ. To check whether Γ is semantically acyclic we just have to check whether Γ is contained in Γ'. Because Γ' is exponentially large, we trivially get a 2EXPSPACE upper bound for the complexity of this step. Interestingly, we show that this can be improved to EXPSPACE. In order to prove this, we show that checking containment of a UC2RPQ Γ_1 in an acyclic UC2RPQ Γ_2 can be solved using exponential space but in such a way that the exponent only depends on the *width* of Γ_2, which is the maximum number of atoms in some disjunct of Γ_2 that mention the same pair of variables. As it turns out, this parameter is polynomial in $|\Gamma|$ for the acyclic approximation Γ'. Thus, applying this refined algorithm for containment we can check whether $\Gamma \subseteq \Gamma'$ in EXPSPACE. We also provide a matching lower bound for this problem.

Organization This chapter is organized as follows. In Section 3.1, we study semantic acyclicity for UCQs and show that the answers to the most basic questions follow from known CSP techniques. We introduce acyclic UC2RPQs in Section 3.2 and give some basic results about these queries. Section 3.3 is devoted to present the main techniques and results related to containment of UC2RPQs. In Section 3.4, we study acyclic approximations of UC2RPQs and show some of their good properties. Finally, in Section 3.5, we study semantic acyclicity of UC2RPQs. We show our main result, namely, that evaluation of semantically acyclic UC2RPQs is fixed-parameter tractable. We also prove that verification of semantically acyclic UC2RPQs is decidable and EXPSPACE-complete.

3.1 Interlude on UCQs

3.1.1 Acyclic UCQs

It is well-known that the evaluation of (U)CQs is NP-complete (see Theorems 2.4 and 2.6 in Section 2.3.1). On the other hand, tractability of (U)CQ evaluation can be obtained by restricting the syntactic shape of CQs. The oldest and most common such restriction is *acyclicity* [160] (also known as *α-acyclicity* [65]), which can be defined in terms of the existence of a well-behaved *tree decomposition* of the *hypergraph* of a CQ [81]. We review such notions below.

Recall that a hypergraph is a tuple $\mathcal{H} = (V, E)$, where V is its finite set of vertices and $E \subseteq 2^V$ is its set of *hyperedges*. With each CQ θ we associate its hypergraph $\mathcal{H}(\theta) = (V, E)$

such that V is the set of variables of θ and E consists of all sets of variables that appear in the same atom of θ. Consider for instance the CQ

$$\theta(x) \leftarrow R(x, y, z), T(y, u, u), S(y, v)$$

Then $\mathcal{H}(\theta) = (V, E)$, where $V = \{x, y, z, u, v\}$ and E consists of the hyperedges $\{x, y, z\}$, $\{y, u\}$ and $\{y, v\}$.

Definition 3.1 (Acyclicity) *A* tree decomposition *of a hypergraph $\mathcal{H} = (V, E)$ is a pair (T, λ), where T is a tree and λ is a mapping from the nodes of T to 2^V, that satisfies the following:*

1. *For each $v \in V$ the set $\{t \in T \mid v \in \lambda(t)\}$ is a connected subset of T.*
2. *Each hyperedge in E is contained in one of the sets $\lambda(t)$, for $t \in T$.*

Then $\mathcal{H}$ is acyclic *if there is a tree decomposition (T, λ) of it such that $\lambda(t)$ is a hyperedge in E, for each $t \in T$.*

A CQ θ is acyclic if its hypergraph $\mathcal{H}(\theta)$ is acyclic. A UCQ $\{\theta_1, \ldots, \theta_n\}$ is acyclic if each θ_i is acyclic ($1 \leq i \leq n$). For instance, the CQ

$$\theta(x) \leftarrow R(x, y, z), T(y, u, u), S(y, v)$$

presented above is acyclic, as witnessed by the following tree decomposition (T, λ) of $\mathcal{H}(\theta)$: T consists of vertices $\{1, 2, 3\}$ and edges $\{(1, 2), (1, 3)\}$, and $\lambda(1) = \{x, y, z\}$, $\lambda(2) = \{y, u\}$ and $\lambda(3) = \{y, v\}$.

It follows from the seminal work of Yannakakis that acyclic UCQs have good properties in terms of evaluation:

Proposition 3.2 [160] *The evaluation problem for acyclic UCQs can be solved in linear time, that is, given an acyclic UCQ Θ, a database $\mathcal{D}$, and a tuple $\bar{a}$, checking whether $\bar{a} \in \Theta(\mathcal{D})$ can be done in $O(|\mathcal{D}| \cdot |\Theta|)$ time.*

3.1.2 Semantically acyclic UCQs

Acyclicity is a syntactic property of UCQs. On the other hand, a non-acyclic UCQ can still be equivalent to an acyclic one. Formally, a UCQ Θ is *semantically acyclic* if there exists an acyclic UCQ Θ' such that $\Theta(\mathcal{D}) = \Theta'(\mathcal{D})$ for each database $\mathcal{D}$; in other words, Θ' is *equivalent* to Θ, denoted $\Theta' \equiv \Theta$.

Evaluating semantically acyclic UCQs As pointed out in [107], there is a close connection between CQ evaluation and constraint satisfaction: Both can be recast as the problem of determining whether there is a homomorphism from one relational structure into another one. This tight connection allows us to export tools from CSP [107, 48] and prove that semantically acyclic UCQs can be evaluated in polynomial time.

Theorem 3.3 *The evaluation problem for semantically acyclic UCQs can be solved in polynomial time.*

In order to prove Theorem 3.3, we first review some basic results about the so-called *existential k-cover game*, which was introduced in [48], to define tractable CSP restrictions. This game is a variant of the *existential k-pebble game* defined by Kolaitis and Vardi [106]. In the k-cover game, instead of imposing the Spoiler to use at most k pebbles – as in the existential k-pebble game – he is allowed to use any number of pebbles, but only as long as the set of elements where the pebbles are placed can be covered by at most k tuples in the database where the Spoiler is playing. We skip here the formal definition of the k-cover game but recall some important properties of the game that were proved in [48].

Proposition 3.4 [48] *The following hold:*

1. *There is a polynomial time algorithm that, given databases $\mathcal{D}$ and $\mathcal{D}'$ over the same schema, decides whether the Duplicator has a winning strategy in the 1-cover game on $\mathcal{D}$ and $\mathcal{D}'$.*
2. *Assume that the Duplicator has a winning strategy in the 1-cover game on $\mathcal{D}$ and $\mathcal{D}'$. Then for each acyclic Boolean CQ θ it is the case that $\theta(\mathcal{D}) = \texttt{true}$ implies $\theta(\mathcal{D}') = \texttt{true}$.*

Proof of Theorem 3.3: Let $\mathcal{D}$ be a database over schema σ and $\bar{a} = (a_1, \ldots, a_r)$ a tuple of elements over $\mathcal{D}$. We denote by $(\mathcal{D}, \bar{a})$ the database over schema $\sigma \cup \{P_1, \ldots, P_r\}$, where $P_1, \ldots, P_r$ are fresh predicates obtained from $\mathcal{D}$ by interpreting P_i as $\{a_i\}$, for each $1 \leq i \leq r$.

We now present an algorithm for evaluating semantically acyclic UCQs. Given a semantically acyclic UCQ $\Theta(\bar{x}) = \{\theta_1(\bar{x}), \ldots, \theta_n(\bar{x})\}$, a database $\mathcal{D}$, and a tuple $\bar{a}$ of elements in $\mathcal{D}$, the algorithm checks whether there is an index $i \in \{1, \ldots, n\}$ such that the Duplicator has a winning strategy in the 1-cover game on $(\mathcal{D}_{\theta_i}, \bar{x})$ and $(\mathcal{D}, \bar{a})$. Recall that $\mathcal{D}_{\theta_i}$ denotes the canonical database of θ_i (see Section 2.1). If this is the case, it accepts (i.e., it declares that $\bar{a} \in \Theta(\mathcal{D})$); otherwise it rejects. From the first part of Proposition 3.4 it follows that this algorithm runs in polynomial time. We show next that the algorithm is sound and complete.

Assume first that $\bar{a} \in \Theta(\mathcal{D})$. Then, $\bar{a} \in \theta_i(\mathcal{D})$, for some $1 \leq i \leq n$. By definition, there is a homomorphism h from θ_i (or, equivalently, from $\mathcal{D}_{\theta_i}$) to $\mathcal{D}$ such that $h(\bar{x}) = \bar{a}$. In particular, h is also an homomorphism from $(\mathcal{D}_{\theta_i}, \bar{x})$ to $(\mathcal{D}, \bar{a})$. This trivially implies the existence of a winning strategy for the Duplicator in the 1-cover game on $(\mathcal{D}_{\theta_i}, \bar{x})$ and $(\mathcal{D}, \bar{a})$. Therefore, the algorithm accepts. (Notice that in this direction the assumption that Θ is semantically acyclic is not used). Conversely, suppose that the algorithm accepts. Then there exists a winning strategy for the Duplicator in the 1-cover game on $(\mathcal{D}_{\theta_i}, \bar{x})$ and $(\mathcal{D}, \bar{a})$, for some $1 \leq i \leq n$. Let $\Theta'(\bar{x}) = \{\theta'_1(\bar{x}), \ldots, \theta'_\ell(\bar{x})\}$ be an acyclic UCQ which is equivalent to Θ. Recall that by Proposition 2.5 (see Section 2.3.1), there exists $1 \leq j \leq \ell$ such that $\theta_i \subseteq \theta'_j$, and thus $\bar{x} \in \theta'_j(\mathcal{D}_{\theta_i})$. Let θ''_j be the acyclic Boolean CQ obtained from $\theta'_j(\bar{x})$ by adding the atoms $P_1(x_1), \ldots, P_r(x_r)$, where $\bar{x} = (x_1, \ldots, x_r)$. Then, $\theta''_j((\mathcal{D}_{\theta_i}, \bar{x})) = \texttt{true}$. Using the second part of Proposition 3.4 with $\theta = \theta''_j$, it follows that $\theta''_j((\mathcal{D}, \bar{a})) = \texttt{true}$, and thus, $\bar{a} \in \theta'_j(\mathcal{D})$. This implies that $\bar{a} \in \Theta'(\mathcal{D})$. Since $\Theta' \equiv \Theta$, we conclude that $\bar{a} \in \Theta(\mathcal{D})$. $\square$

Verifying semantically acyclic UCQs Notice that the class of acyclic UCQs is remarkably well-behaved: Queries in the class are not only tractable, but also verifying whether a given UCQ belongs to the class can be done in polynomial (in fact, linear) time [149]. On the other hand, using techniques similar to those in [59], one can prove that this good behavior does not extend to the class of semantically acyclic UCQs, as the problem of verifying whether a query is semantically acyclic is computationally hard:

Proposition 3.5 *The problem of verifying whether a UCQ Θ is semantically acyclic is* NP*-complete. It remains* NP*-hard even when the input is restricted to Boolean CQs whose schema consists of a single binary relation (i.e. directed graphs).*

Before proving Proposition 3.5, we need some auxiliary results. Let $\theta(\bar{x})$ be a CQ. We say that a CQ $\theta'(\bar{x})$ is a *core* of θ [97, 40] if (a) $\theta \equiv \theta'$, and (b) no CQ with strictly fewer atoms than θ' is equivalent to θ. It is known (see, e.g., [97]) that each CQ $\theta(\bar{x})$ has a unique (up to isomorphism) core, and thus we can talk about *the* core of $\theta(\bar{x})$. As it turns out, acyclicity is preserved under taking cores:

Claim 3.6 *If θ is an acyclic CQ, then the core of θ is also acyclic.*

Proof: We show the claim in two steps. First, we show that the class of acyclic CQs is closed under taking *strong induced subqueries*. Then we prove that the core of a CQ θ is always a strong induced subquery.

For each atom P of θ we denote by V_P the set of variables that are mentioned in P. A CQ θ' is a strong induced subquery of a CQ θ if:

1. The set $V_{\theta'}$ of variables mentioned in θ' is contained in the set V_θ of variables mentioned in θ.
2. The atoms of θ' are exactly the atoms of θ induced by the variables in $V_{\theta'}$.
3. The free variables of θ' are exactly the free variables of θ.
4. If P is an atom in θ but not in θ', then there exists an atom P' in θ' that contains all the variables in $V_P \cap V_{\theta'}$.

Suppose θ is an acyclic CQ and let θ' be a strong induced subquery of θ. Let (T, λ) be a tree decomposition of $\mathcal{H}(\theta)$ witnessing the acyclicity of θ. Consider the tree decomposition (T, λ') such that, for each $t \in T$, $\lambda'(t) = \lambda(t) \cap V_{\theta'}$. Clearly, (T, λ') is a tree decomposition of $\mathcal{H}(\theta')$. We now show that we can transform the decomposition (T, λ') into a tree decomposition $(\tilde{T}, \tilde{\lambda})$ of $\mathcal{H}(\theta')$ such that $\tilde{\lambda}(t)$ is a hyperedge of $\mathcal{H}(\theta')$, for each $t \in \tilde{T}$. This would imply that θ' is acyclic, as required.

Let $t \in T$ be a node such that $\lambda'(t)$ is not a hyperedge of $\mathcal{H}(\theta')$. By definition, $\lambda(t)$ is a hyperedge of $\mathcal{H}(\theta)$. We can then pick an atom P in θ such that the set of variables V_P mentioned in P is precisely $\lambda(t)$. Then, by definition, $\lambda'(t) = V_P \cap V_{\theta'}$. By condition (4) in the definition of strong induced subquery, there is an atom P' in θ' such that $\lambda'(t) = V_P \cap V_{\theta'} \subseteq V_{P'}$. Moreover, by the definition of tree decomposition, there is a node $t^* \in T$ such that $V_{P'} \subseteq \lambda'(t^*)$. It follows that $\lambda'(t) \subseteq \lambda'(t^*)$. Applying standard techniques (see, e.g., [74]), we can remove the node t from T while preserving the tree decomposition properties. Iteratively

applying this modification, we end up with a tree decomposition $(\tilde{T}, \tilde{\lambda})$ that witnesses the acyclicity of θ'.

Finally, we show that the core θ' of a CQ θ is a strong induced subquery. Indeed, it follows from well-known core properties [97] that conditions (1), (2) and (3) in the definition of strong induced subquery hold. We prove next that condition (4) also holds. Again by well-known properties of cores [97], we can assume without loss of generality that there is a *retract* h from θ to θ', i.e., a homomorphism from $\mathcal{D}_\theta$ to $\mathcal{D}_{\theta'}$ that is the identity over the variables $V_{\theta'}$ of θ'. Let $P = R(u_1, \ldots, u_m)$ be an atom in θ but not in θ'. Then it is the case that $R(h(u_1), \ldots, h(u_m))$ is an atom of θ'. Since h is the identity in $V_P \cap V_{\theta'}$, we conclude that condition (4) holds. □

Claim 3.7 *Let θ be a CQ. Then θ is semantically acyclic if and only if its core is acyclic.*

Proof: The *if* direction is trivial since θ and its core are equivalent. For the *only if* direction, assume that θ is semantically acyclic. Then there is an acyclic UCQ $\Theta' = \{\theta'_1, \ldots, \theta'_n\}$ such that $\theta \equiv \Theta'$. It then follows from Proposition 2.5 that there is some $i \in \{1, \ldots, n\}$ for which it is the case that $\theta \equiv \theta'_i$. Since θ'_i is acyclic, we have from Claim 3.6 that its core is also acyclic. On the other hand, it is known (see, e.g., [97]) that equivalent CQs have the same core (up to renaming of variables). It follows that the core of θ coincides with the core of θ'_i (up to renaming of variables), and therefore, it is acyclic. □

Proof of Proposition 3.5: For membership in NP we show that if Θ is semantically acyclic then it is equivalent to an acyclic UCQ Θ' whose size is bounded by the size of Θ. Then, given a UCQ Θ, the NP algorithm guesses an acyclic UCQ Θ' with $|\Theta'| \leq |\Theta|$, and then checks whether $\Theta \equiv \Theta'$. The latter can be done in NP (see Theorem 2.6 in Section 2.3.1), and hence the whole procedure is in NP.

Now we turn to our initial claim: If Θ is a semantically acyclic UCQ, then it is equivalent to an acyclic UCQ Θ' with $|\Theta'| \leq |\Theta|$. Assume that Θ is semantically acyclic and let Θ' be an equivalent acyclic UCQ. Further, let Θ_{min} be a subset of the disjuncts in Θ such that (1) Θ_{min} is equivalent to Θ, and (2) no proper subset of Θ_{min} is equivalent to Θ. Analogously, we define Θ'_{min} to be "minimally" equivalent to Θ'. By minimality, there are no distinct disjuncts θ_1 and θ_2 in Θ_{min} such that $\theta_2 \subseteq \theta_1$. From Proposition 2.5 and the fact that $\Theta_{min} \equiv \Theta'_{min}$, it follows that, for each disjunct θ of Θ_{min}, there is an equivalent disjunct θ' in Θ'_{min}. Thus, each disjunct of Θ_{min} is semantically acyclic. Let Θ^* be the UCQ obtained from Θ_{min}, by replacing each disjunct by its core. From Claim 3.7, we have that Θ^* is acyclic. Moreover, $\Theta^* \equiv \Theta$ and $|\Theta^*| \leq |\Theta|$. This completes the proof of the NP upper bound.

The lower bound follows from [59], which shows that the problem of checking whether a Boolean CQ over a single binary relation is equivalent to an acyclic one is NP-hard. □

3.2 Acyclic UC2RPQs

Acyclicity of C2RPQs has been studied in several recent papers that define it in terms of the acyclicity of its *underlying conjunctive query* [13, 16]. Let $\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1, y_1'), \ldots, \mathcal{A}_m(y_m, y_m')$ be a C2RPQ. Its underlying CQ is the query over the schema of binary relation symbols $T_1, \ldots, T_m$ defined as: $\theta(\bar{x}) \leftarrow T_1(y_1, y_1'), \ldots, T_m(y_m, y_m')$. Intuitively, this underlying conjunctive query represents the structure of γ when the regular languages that label the atoms of γ are turned into relation symbols.

A C2RPQ is *acyclic* if its underlying CQ is acyclic. A UC2RPQ is acyclic if each one of its C2RPQs is acyclic. By combining techniques for UC2RPQ evaluation and polynomial time evaluation of acyclic CQs, it is possible to prove that the evaluation problem for acyclic UC2RPQs can be solved in polynomial time [160, 13]. We present the simple proof of this fact for the sake of completeness.

Theorem 3.8 *The evaluation problem for acyclic UC2RPQs can be solved in polynomial time. More precisely, given an acyclic UC2RPQ Γ, a graph database $\mathcal{G}$, and a tuple $\bar{n}$, checking whether $\bar{n} \in \Gamma(\mathcal{G})$ can be done in $O(|\Gamma|^2 \cdot |\mathcal{G}|^2)$ time.*

Proof: Let $\mathcal{G}$ be a graph database and Γ a UC2RPQ. Let us denote by Θ the UCQ that is obtained from Γ by replacing each C2RPQ γ in Γ by its underlying conjunctive query. Let $T_1(y_1, y_1'), \ldots, T_n(y_n, y_n')$ be an enumeration of all the relational atoms in Θ, and assume that for each i with $1 \leq i \leq n$ the atom $T_i(y_i, y_i')$ is introduced in Θ as a replacement for the 2RPQ $\mathcal{A}_i(y_i, y_i')$. Further, let $\mathcal{D}$ be a relational database over schema $\{T_1, \ldots, T_n\}$ such that the interpretation of the binary relation symbol T_i over $\mathcal{D}$, for each $1 \leq i \leq n$, is precisely $\mathcal{A}_i(\mathcal{G})$. Notice that $\mathcal{D}$ can be constructed in time $O(|\Gamma| \cdot |\mathcal{G}|^2)$ from $\mathcal{G}$ and Γ, using the fact that each $\mathcal{A}_i(\mathcal{G})$ can be computed in time $O(|\mathcal{A}_i| \cdot |\mathcal{G}|^2)$ (see Proposition 2.8 in Section 2.3.2) and $|\mathcal{A}_1| + \cdots + |\mathcal{A}_n|$ is $O(|\Gamma|)$.

It is clear then that checking whether a tuple $\bar{n}$ of nodes in $\mathcal{G}$ belongs to $\Gamma(\mathcal{G})$ reduces to checking whether $\bar{n} \in \Theta(\mathcal{D})$. Since Θ is acyclic, it follows from Proposition 3.2 that the latter can be done in time $O(|\Theta| \cdot |\mathcal{D}|)$, that is, in time $O(|\Gamma|^2 \cdot |\mathcal{G}|^2)$. □

Recall that a CQ θ is acyclic if its hypergraph $\mathcal{H}(\theta)$ admits a tree decomposition (T, λ) such that each set of the form $\lambda(t)$, for $t \in T$, is a hyperedge of $\mathcal{H}(\theta)$. The fact that acyclicity of C2RPQs is defined in terms of the acyclicity of its underlying CQ – and that the latter is specified in a schema of binary arity – allows us to provide a simple characterization of the class of acyclic C2RPQs that will be useful in our proofs. We explain this below.

The *simple undirected underlying graph* of C2RPQ $\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1, y_1'), \ldots, \mathcal{A}_m(y_m, y_m')$, which is denoted by $\mathcal{U}(\gamma)$, is the graph whose vertices are the variables of γ and its set of edges is $\{\{y_i, y_i'\} \mid 1 \leq i \leq m \text{ and } y_i \neq y_i'\}$. Notice that $\mathcal{U}(\gamma)$ is indeed simple (it contains neither loops nor multiedges) and undirected. The following self-evident proposition provides a simple reformulation of the notion of acyclicity of C2RPQs in terms of the acyclicity of their simple undirected underlying graph:

Proposition 3.9 *A C2RPQ γ is acyclic if and only if $\mathcal{U}(\gamma)$ is acyclic.*

Example 3.10 Let us consider again the C2RPQ

$$\gamma(x, y) \leftarrow \texttt{creator}(x, z), \texttt{inConf}(z, w), \texttt{creator}^{-}(z, y)$$

in Example 2.2. The graph $\mathcal{U}(\gamma)$ consists of nodes x, z, w, y and edges $\{x, z\}$, $\{z, w\}$, and $\{z, y\}$. Clearly, $\mathcal{U}(\gamma)$ is acyclic, and, therefore, $\gamma(x, y)$ is acyclic. Similarly, it can be proved that the C2RPQ $\gamma'(x, y)$ in Example 2.2 is acyclic. □

Notice that this definition of acyclicity allows for the existence of loops and multiedges in the structure of a C2RPQ, that is, in its underlying CQ, as shown in the following example.

Example 3.11 Let $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ be arbitrary NFAs over Σ. The CRPQs $\gamma \leftarrow \mathcal{A}_1(x, x)$ and $\gamma' \leftarrow \mathcal{A}_1(x, y), \mathcal{A}_2, (y, x)$ are acyclic. Notice that the underlying CQ of γ contains a loop, while the underlying CQ of γ' contains edges from x to y and from y to x. On the other hand, the CRPQ $\gamma'' \leftarrow \mathcal{A}_1(x, y), \mathcal{A}_2(y, z), \mathcal{A}_3(z, x)$ is not acyclic. □

Our goal is to study the notion of semantic acyclicity for UC2RPQs. To attack the problem of evaluation for UC2RPQs that are semantically acyclic we make a necessary detour in the next section to study the containment problem for UC2RPQs, and then in Section 3.4 to study approximations of UC2RPQs.

3.3 Containment of UC2RPQs

To prove our results, it is essential to have a good understanding of the machinery used to study the containment problem for UC2RPQs. We develop such machinery in this section.

Calvanese et al. proved that the containment problem for C2RPQs is in Expspace [33]. More specifically, it follows from [33] that checking whether $\gamma \subseteq \gamma'$, when γ and γ' are C2RPQs, can be solved using exponential space in such a way that the exponent only depends on $|\gamma'|$ and the number of variables in γ. For a UC2RPQ Γ, we define $\mathsf{maxvar}(\Gamma)$ to be the maximum number of variables over all disjuncts of Γ. A straightforward extension of the techniques in [33] shows that containment of Γ in Γ', when Γ and Γ' are UC2RPQs, can also be solved in exponential space, but now the exponent depends only on $|\Gamma'|$ and $\mathsf{maxvar}(\Gamma)$. It is also shown in [33] that the containment problem for C2RPQs is Expspace-hard even when both γ and γ' are acyclic CRPQs [33]. In summary:

Proposition 3.12 *The following hold:*

1. *Checking whether $\Gamma \subseteq \Gamma'$, for UC2RPQs Γ and Γ', can be solved using exponential space where the exponent only depends on $|\Gamma'|$ and $\mathsf{maxvar}(\Gamma)$.*
2. *The problem of checking whether $\gamma \subseteq \gamma'$, for C2RPQs γ and γ', is* Expspace*-hard. It remains hard even if both γ and γ' are acyclic CRPQs.*

When Γ' is an acyclic UC2RPQ it is possible to show that the containment of Γ in Γ' can

be checked in exponential space, but where the exponent only depends on Γ', specifically, on the *width* of Γ' (defined below). This result will be crucial to later prove in Section 3.5 that the problem of checking whether a UC2RPQ is semantically acyclic (i.e., equivalent to an acyclic UC2RPQ) is in EXPSPACE.

Let γ be a C2RPQ over Σ. For two distinct variables x, y in γ we define $\texttt{Atoms}(x, y)$ to be the set of atoms in γ that are of the form $\mathcal{A}(x, y)$ or $\mathcal{A}(y, x)$, where $\mathcal{A}$ is an NFA over $\Sigma^{\pm}$. The *width* $w(\gamma)$ of γ is defined as follows:

$$w(\gamma) \;=\; \max\{\, |\texttt{Atoms}(x, y)| \;\mid\; x, y \text{ are distinct variables appearing in } \gamma\}$$

If $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ is a UC2RPQ, then the *width* of Γ is $w(\Gamma) = \max\{w(\gamma_i) \mid 1 \leq i \leq m\}$. We devote the rest of this section to prove the following theorem.

Theorem 3.13 *Containment of a UC2RPQ Γ in an acyclic UC2RPQ Γ' can be solved in deterministic $O((|\Gamma| + |\Gamma'|)^{C \cdot w(\Gamma')})$ space, for some constant $C \geq 1$.*

The proof exploit automata techniques as in [33, 36, 137]. It follows from [137] (see also Section 5.3) that checking containment of two UC2RPQs can be reduced to checking containment of a *single-atom* C2RPQ (i.e., a Boolean C2RPQ of the form $\gamma \leftarrow \mathcal{A}(x, y)$, for a 2RPQ $\mathcal{A}$) in a Boolean UC2RPQ (single-atoms C2RPQs are called *Boolean* 2RPQs in [137]). In particular, it is shown that there are integers $c, c' \geq 1$ and a polynomial time algorithm that, given UC2RPQs Γ and Γ', constructs a single-atom C2RPQ $\tilde{\gamma}$ and a Boolean UC2RPQ $\tilde{\Gamma}$ such that the following hold:

1. $\Gamma \subseteq \Gamma'$ if and only if $\tilde{\gamma} \subseteq \tilde{\Gamma}$,
2. $|\tilde{\gamma}| = O(|\Gamma|^c)$ and $|\tilde{\Gamma}| = O((|\Gamma| + |\Gamma'|)^{c'})$,
3. $w(\Gamma') = w(\tilde{\Gamma})$, and
4. if Γ' is acyclic then $\tilde{\Gamma}$ also is.

Our proof is based on the following lemma.

Lemma 3.14 *There are integers $d, d' \geq 1$ such that the problem of checking containment of a single-atom C2RPQ γ in an acyclic UC2RPQ Γ' can be solved in deterministic $O(|\Gamma'|^{d \cdot w(\Gamma')} \cdot |\gamma|^{d'})$ space.*

Lemma 3.14 directly implies Theorem 3.13. Indeed, in order to check containment of a UC2RPQ Γ in an acyclic UC2RPQ Γ', we first construct the single-atom C2RPQ $\tilde{\gamma}$ and the acyclic UC2RPQ $\tilde{\Gamma}$, and then apply the algorithm from Lemma 3.14. The space used is

$$O(|\tilde{\Gamma}|^{d \cdot w(\tilde{\Gamma})} |\tilde{\gamma}|^{d'}) \;=\; O((|\Gamma| + |\Gamma'|)^{c' d \cdot w(\Gamma')} |\Gamma|^{cd'}),$$

which is $O((|\Gamma| + |\Gamma'|)^{Cw(\Gamma')})$, for $C \geq c'd + cd'$.

Before proving Lemma 3.14, we make a necessary detour through the notions of *canonical* graph database and *foldings*, which are an important component of several containment algorithms for UC2RPQs and its extensions [73, 33, 137].

3.3.1 Canonical databases and foldings

A *semipath* is a graph database whose underlying graph is a path. Formally, the graph database $\mathcal{G} = (N, E)$ over Σ is a semipath if $V = \{n_0, \ldots, n_k\}$, $E = \{e_1, \ldots, e_k\}$, and for each $1 \leq i \leq k$ it is the case that e_i is of the form (n_{i-1}, a_i, n_i) or (n_i, a_i, n_{i-1}) (for $a_i \in \Sigma$). The *internal nodes* of $\mathcal{G}$ are $\{n_1, \ldots, n_{k-1}\}$. If $k = 0$ we call $\mathcal{G}$ the *empty semipath*.

Notice that the completion $\mathcal{G}^{\pm}$ of the semipath $\mathcal{G}$ contains a single path π from n_0 to n_k. The label of π is $b_1 b_2 \ldots b_k$, where for each $1 \leq i \leq k$ it is the case that $b_i = a_i$, if e_i is of the form (n_{i-1}, a_i, n_i), and $b_i = a_i^-$, otherwise. We slightly abuse notation and write $\mathsf{label}(\mathcal{G})$ in order to denote $\mathsf{label}(\pi)$ whenever $\mathcal{G}$ is a semipath. Notice that if $\mathcal{G}$ is an empty semipath then $\mathsf{label}(\mathcal{G}) = \varepsilon$.

Containment of UC2RPQs can be recast in terms of the notion of *canonical* databases [73, 33], which we describe below.

Definition 3.15 (Canonical databases for UC2RPQs) *Let γ be a C2RPQ of the form $\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1, y_1'), \ldots \mathcal{A}_m(y_m, y_m')$. A graph database $\mathcal{G}$ is* canonical *for γ, if there exists a mapping ν from the variables of γ to the nodes of $\mathcal{G}$ such that:*

1. *$\mathcal{G}$ consists of m semipaths, one for each conjunct of γ. Formally, for each $1 \leq i \leq m$ there is a semipath κ_i in $\mathcal{G}$ from $\nu(y_i)$ to $\nu(y_i')$. These semipaths are node and edge disjoint, save for the start and end nodes that can be shared between different κ_i's. E.g., if $y_i = y_j'$, for $1 \leq i, j \leq m$, then $\nu(y_i) = \nu(y_j')$ and, thus, the start node of κ_i coincides with the end node of κ_j.*
2. *For each $1 \leq i \leq m$ it is the case that $\mathsf{label}(\kappa_i)$ is accepted by $\mathcal{A}_i$.*
3. *For every two distinct variables z and z' in γ it is the case that $\nu(z) = \nu(z')$ if and only if there is a sequence $z_0, z_1, \ldots, z_\ell$ of variables in γ such that $z_0 = z$, $z_\ell = z'$ and for each $1 \leq j \leq \ell$ there is an $1 \leq i \leq m$ such that (i) the atom $\mathcal{A}_i(y_i, y_i')$ in γ corresponds to either $\mathcal{A}_i(z_{j-1}, z_j)$ or $\mathcal{A}_i(z_j, z_{j-1})$, and (ii) κ_i is an empty semipath.*

We say that mapping ν is associated *with the canonical database $\mathcal{G}$. A* canonical database *for a UC2RPQ Γ is a canonical database for some disjunct of Γ.*

The intuition is that a canonical database $\mathcal{G}$ is obtained from a C2RPQ γ as follows. We view each variable in γ as a node in $\mathcal{G}$. For each atom $\mathcal{A}_i(y_i, y_i')$ in γ, we add to $\mathcal{G}$ a semipath κ_i from y_i to y_i' with fresh internal nodes that is accepted by $\mathcal{A}_i$. We then identify nodes y_i and y_i' each time κ_i is chosen as an empty semipath. Condition (3) in the definition of canonical database ensures that the identification of nodes only occurs due to those empty paths.

A slight extension of the techniques used in [33] yields the following proposition, which states that if $\Gamma \not\subseteq \Gamma'$, for UC2RPQs Γ and Γ', then a counterexample to $\Gamma \subseteq \Gamma'$ can be found in the set of canonical databases for Γ.

Proposition 3.16 *Let $\Gamma(\bar{x})$ and $\Gamma'(\bar{x})$ be two UC2RPQs. Then $\Gamma \subseteq \Gamma'$ if and only if for*

each canonical database $\mathcal{G}$ for Γ with associated mapping ν, it is the case that $\nu(\bar{x}) \in \Gamma'(\mathcal{G})$.[1]

We now define *foldings*. Let Σ be a finite alphabet. Recall that we denote by $\Sigma^{\pm}$ the alphabet $\Sigma \cup \{a^- \mid a \in \Sigma\}$. If $p \in \Sigma^{\pm}$ and $p = a$ for some $a \in \Sigma$, then p^- denotes a^-. On the other hand, if $p = a^-$ for $a \in \Sigma$, then p^- denotes a. Let $s = s_1 \dots s_k$ and $t = t_1 \dots t_\ell$ be words over $\Sigma^{\pm}$. We say that t *folds* into s [35] from j_1 to j_2, for $j_1, j_2 \in \{0, \dots, k\}$, if there is a sequence $i_0, \dots, i_\ell$ of positions in the set $\{0, \dots, k\}$ such that:

- $i_0 = j_1$ and $i_\ell = j_2$.
- For each $1 \leq j \leq \ell$, it is the case that $i_j = i_{j-1} + 1$ and $t_j = s_{i_j}$, or $i_j = i_{j-1} - 1$ and $t_j = s^-_{i_{j-1}}$.

Intuitively, t folds into s if t can be read in s by a *two-way automaton* that outputs symbol p each time p is read from left-to-right, and symbol p^- each time p is read from right-to-left. For instance, the word $abb^-a^-abb^-c$ folds into abb^-c from 0 to 5.

The notion of folding is also important in Part II. In this part, foldings are defined explicitly as a sequence of positions and some more notation is needed (see Definition 5.19 in Section 5.3.2).

3.3.2 Proof of Lemma 3.14

We need to introduce some concepts from automata theory. Recall that *two-way alternating automata* generalize NFAs with the ability to move on the input word in both directions, and with the possibility to perform universal or existential moves (actually a combination of both). We define them following [36]. Given a set X, let $\mathcal{B}(X)$ be the set of positive Boolean formulae over X, built inductively by applying $\wedge$ and $\vee$ starting from `true`, `false` and elements of X. For a set $Y \subseteq X$ and a formula $\varphi \in \mathcal{B}(X)$, we say that Y *satisfies* φ if and only if assigning `true` to the elements in Y and `false` to those in $X \setminus Y$ makes φ true. A two-way alternating finite automaton (2AFA) is a tuple $\mathcal{A} = (\Sigma, Q, \delta, Q_0, F)$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to \mathcal{B}(\{-1, 0, 1\} \times Q)$ is the transition function. Intuitively, a transition $\delta(q, a)$ spawns several copies of $\mathcal{A}$, each one starting in a certain state and with the head on the symbol to the left of a (-1), to the right of a (1), or on a itself (0), and specifies by means of a positive Boolean formula how to combine acceptance or non-acceptance of the spawned copies.

A *run* of $\mathcal{A}$ on a word $w = a_0 \dots a_{\ell-1}$, for $\ell \geq 0$, is a pair (T, λ), where T is a finite rooted tree and λ is a labeling of the nodes of T by elements in $Q \times \{0, \dots, \ell\}$ such that:

1. $\lambda(r) = (q_0, 0)$, for some $q_0 \in Q_0$, where r is the root of T.
2. For each node u in T with $\lambda(u) = (q, i)$ and $\delta(q, a_i) = \varphi$, where $q \in Q$, $0 \leq i \leq \ell - 1$,

[1] The original definition of canonical database in [33] only considers conditions (1) and (2). Condition (3) is innocuous in terms of the characterization of containment we provide here (i.e., Proposition 3.16 continues to hold if we remove such condition). As we shall see, on the other hand, condition (3) is important for our proof.

and $\varphi \in \mathcal{B}(\{-1,0,1\} \times Q)$, there is a (possible empty) set $X = \{(d_1, q_1), \ldots, (d_n, q_n)\} \subseteq \{-1,0,1\} \times Q$ such that (i) X satisfies φ, and (ii) for all $j \in \{1, \ldots, n\}$ there is a child of u in T labeled $(q_i, i + d_j)$.

The run is *accepting* if for each leaf u of T it is the case that $\lambda(u)$ is of the form (q, ℓ), for $q \in F$. The 2AFA $\mathcal{A}$ accepts w if it has an accepting run on it. It is known that 2AFAs define regular languages [114]. The following fact was shown in [36].

Lemma 3.17 [36] *Given a 2AFA $\mathcal{A}$ with n states, one can construct an NFA with $2^{O(n)}$ states that accepts precisely those words which are not accepted by $\mathcal{A}$.*

We now start with the proof of the lemma. Given a single-atom C2RPQ γ and an acyclic UC2RPQ Γ' over Σ, we define an NFA $\mathcal{A}_{\gamma,\Gamma'}$ (over an extended alphabet) such that $\gamma \not\subseteq \Gamma'$ if and only if the language accepted by $\mathcal{A}_{\gamma,\Gamma'}$ is not empty. Assume that γ is of the form $\gamma \leftarrow \mathcal{A}_\gamma(x, y)$, where $\mathcal{A}_\gamma$ is a 2RPQ over Σ. The definition of $\mathcal{A}_{\gamma,\Gamma'}$ is based on NFAs $\mathcal{A}_{cd}$ and $\mathcal{A}_{loops}$, and on a 2AFA $\mathcal{A}_{\Gamma'}$, all of which are introduced below.

All these automata are defined over a suitable alphabet Δ. Let $\texttt{Loops}$ be $2^{\mathcal{T}_{\Gamma'}}$, where $\mathcal{T}_{\Gamma'}$ is the set:

$$\{(\mathcal{A}, q, q') \mid \mathcal{A} \text{ is an 2RPQ mentioned in some atom of } \Gamma' \text{ and } q, q' \text{ are states of } \mathcal{A}\}$$

Then $\Delta := \Sigma^{\pm} \cup \texttt{Loops}$. Note that the size of Δ is bounded by $O(2^{|\Gamma'|^2})$.

Let us start by defining the NFA $\mathcal{A}_{cd}$ as the product of the following two NFAs:

1. $\mathcal{A}_{\text{format}}$, which accepts all words over Δ satisfying the regular expression $\texttt{Loops} \cdot (\Sigma^{\pm} \cdot \texttt{Loops})^*$. (Here we are slightly abusing notation by writing $\texttt{Loops}$ and $\Sigma^{\pm}$ for the regular expression that represents the union of all symbols in $\texttt{Loops}$ and $\Sigma^{\pm}$, respectively).
2. $\mathcal{A}_{\text{acc}}$, which accepts all words $a_1 \ldots a_m$ over Δ such that $a_2 a_4 \ldots a_{2\lfloor m/2 \rfloor}$ is a word in $\Sigma^{\pm}$ accepted by $\mathcal{A}_\gamma$.

It is easy to see that the number of states of $\mathcal{A}_{cd}$ can be bounded by $O(|\gamma|)$.

Since γ is a single-atom C2RPQ it is the case that if $\mathcal{G}$ is a canonical database of γ with associated mapping ν, then $\mathcal{G}$ is simply a semipath from $\nu(x)$ to $\nu(y)$ such that $\mathsf{label}(\mathcal{G})$ is accepted by $\mathcal{A}_\gamma$. Thus we can naturally associate the canonical databases of γ with the words accepted by $\mathcal{A}_\gamma$. The NFA $\mathcal{A}_{cd}$ then accepts all the (labels of) canonical databases of γ "adorned" with some additional information from $\texttt{Loops}$. More formally, $\mathcal{A}_{cd}$ accepts words of the form $L_0 a_1 L_1 \ldots L_{m-1} a_m L_m$, where $L_i \in \texttt{Loops}$ and $a_i \in \Sigma^{\pm}$ for each $0 \leq i \leq m$, such that (i) $a_1 a_2 \ldots a_m$ is the label of some canonical database of γ, and (ii) $L_0 L_1 \ldots L_m$ represents an "adornment" of $a_1 a_2 \ldots a_m$ over $\texttt{Loops}$.

Let us now define the NFA $\mathcal{A}_{loops}$ over alphabet Δ. This NFA will help us restricting the possible adornments of the words $L_0 a_1 L_1 \ldots L_{m-1} a_m L_m$ accepted by $\mathcal{A}_{cd}$ in such a way that the following holds for each $0 \leq i \leq m$:

(†) $(\mathcal{A}, q, q')$ belongs to L_i if and only if there is a word w over $\Sigma^{\pm}$ and a run of $\mathcal{A}$ over w from state q to q' such that w folds into $a_1 \ldots a_m$ from i to i.

If the word $L_0 a_1 L_1 \dots L_{m-1} a_m L_m$ satisfies (†) for each $0 \leq i \leq m$ then we say that it is *correct*. The NFA $\mathcal{A}_{loops}$ then satisfies that the intersection of $\mathcal{A}_{cd}$ and $\mathcal{A}_{loops}$ accepts precisely those words accepted by $\mathcal{A}_{cd}$ which are correct. We explain next how $\mathcal{A}_{loops}$ is defined.

We start by defining a function $\xi : \texttt{Loops} \times \texttt{Loops} \to \texttt{Loops}$ as follows. Given $(L_1, L_2) \in \texttt{Loops} \times \texttt{Loops}$, the set $\xi(L_1, L_2)$ contains exactly all those triples $(\mathcal{A}, q, q') \in \mathcal{T}_{\Gamma'}$ such that either (i) $q = q'$, or (ii) there is a sequence

$$(\mathcal{A}, q_0, q_1), (\mathcal{A}, q_1, q_2), \dots, (\mathcal{A}, q_{k-1}, q_k)$$

for which (a) $q_0 = q$, $q_k = q'$ and the states $q_0, \dots, q_k$ are all distinct, and (b) $(\mathcal{A}, q_j, q_{j+1}) \in L_1 \cup L_2$, for each $0 \leq j \leq k-1$.

We also define two functions ξ^ℓ and ξ^r from $\texttt{Loops} \times \Sigma^\pm$ to $\texttt{Loops}$. Given $L \in \texttt{Loops}$ and $a \in \Sigma^\pm$, the set $\xi^\ell(L, a)$ contains exactly all those triples $(\mathcal{A}, q, q') \in \mathcal{T}_{\Gamma'}$ such that either (i) $q = q'$, or (ii) there is a sequence

$$(\mathcal{A}, q_0, q_1), (\mathcal{A}, q_2, q_3), \dots, (\mathcal{A}, q_{2k}, q_{2k+1}),$$

where all the q_i's are distinct and the following holds:

1. There is an a^--labeled transition in $\mathcal{A}$ from q to q_0,
2. there is an a-labeled transition from q_{2k+1} to q' in $\mathcal{A}$, and
3. for each $0 \leq j \leq k$ the tuple $(\mathcal{A}, q_{2j}, q_{2j+1})$ belongs to L, and for each $0 \leq j \leq k-1$ there is a run of $\mathcal{A}$ over aa^- from q_{2j+1} to q_{2j+2}.

We define $\xi^r(L, a)$ analogously, but switching a for a^-, and viceversa.

Consider a word $a_1 \dots a_m$, where $a_i \in \Sigma^\pm$ for each $1 \leq i \leq m$. For each $0 \leq i \leq m$ we define sets $T_i^\ell, T_i^r \in \texttt{Loops}$ as follows:

- The set T_i^ℓ contains precisely those triples $(\mathcal{A}, q, q')$ such that there is a word w over $\Sigma^\pm$ and a run of $\mathcal{A}$ over w from state q to q' such that w folds into $a_1 \dots a_i$ from i to i. If $i = 0$ then T_i^ℓ is just the set of all triples of the form $(\mathcal{A}, q, q)$.
- The set T_i^r contains precisely those triples $(\mathcal{A}, q, q')$ such that there is a word w over $\Sigma^\pm$ and a run of $\mathcal{A}$ over w from state q to q' such that w folds into $a_{i+1} \dots a_m$ from 0 to 0. If $i = m$ then T_i^r is just the set of all triples of the form $(\mathcal{A}, q, q)$.

As it turns out, the fact that L_i satisfies (†) can be stated in terms of T_i^ℓ and T_i^r, for each $0 \leq i \leq m$. Indeed, it is not hard to see that L_i satisfies (†) if and only if $L_i = \xi(T_i^\ell, T_i^r)$. Furthermore, consecutive T_i^ℓ's (resp., T_i^r's) can be expressed in terms of the function ξ^ℓ (resp., ξ^r). In fact, it is easy to see that for each $0 \leq i \leq m$ it is the case that $T_{i+1}^\ell = \xi^\ell(T_i^\ell, a_{i+1})$ (resp., $T_i^r = \xi^r(T_{i+1}^r, a_{i+1})$).

Now we construct the NFA $\mathcal{A}_{loops}$. The set of states is $\texttt{Loops} \times \texttt{Loops}$. The initial states are the pairs of the form (L_1, L_2), where L_1 is the set of triples in $\mathcal{T}_{\Gamma'}$ of the form $(\mathcal{A}, q, q)$. Similarly, the final states are the pairs of the form (L_1, L_2), where now L_2 is the set of all triples in $\mathcal{T}_{\Gamma'}$ of the form $(\mathcal{A}, q, q)$. The transition function δ is defined as follows:

1. Let L_1, L_2, L be elements in `Loops`. Then $(L_1, L_2) \in \delta((L_1, L_2), L)$ whenever $L = \xi(L_1, L_2)$.
2. Assume that L_1, L_2, L_1', L_2' are elements in `Loops` and a is a symbol in $\Sigma^{\pm}$. Then $(L_1', L_2') \in \delta((L_1, L_2), a)$ whenever $L_1' = \xi^{\ell}(L_1, a)$ and $L_2 = \xi^{r}(L_2', a)$.

Notice that the size of $\mathcal{A}_{loops}$ is bounded by $2^{O(|\Gamma'|^2)}$.

Therefore, an accepting run of $\mathcal{A}_{loops}$ over $L_0 a_1 L_1 \dots L_{m-1} a_m L_m$ is a sequence of the form:

$$(L_1^0, L_2^0), (L_1^0, L_2^0), (L_1^1, L_2^1), (L_1^1, L_2^1), \dots, (L_1^m, L_2^m), (L_1^m, L_2^m),$$

where the following holds by the definition of the transition function δ:

1. $(L_1^{i+1}, L_2^i) = (\xi^{\ell}(L_1^i, a_{i+1}), \xi^{r}(L_2^{i+1}, a_{i+1}))$, for each $0 \leq i \leq m-1$, and
2. $L_i = \xi(L_1^i, L_2^i)$, for each $0 \leq i \leq m$.

Further, (L_1^0, L_2^0) and (L_1^m, L_2^m) are an initial and final state of $\mathcal{A}_{loops}$, respectively.

We prove now that $\mathcal{A}_{loops}$ has an accepting run over $L_0 a_1 L_1 \dots L_{m-1} a_m L_m$ if and only if this word is correct. From our previous remarks, it is sufficient to prove that $L_i = \xi(T_i^{\ell}, T_i^{r})$ for each $0 \leq i \leq m$. But since we know that $L_i = \xi(L_1^i, L_2^i)$, we only need to prove that $L_1^i = T_i^{\ell}$ and $L_2^i = T_i^{r}$ for each $0 \leq i \leq m$. We prove the former first by induction on $0 \leq i \leq m$. For the base case $i = 0$, notice that L_1^0 is the set of all triples in $\mathcal{T}_{\Gamma'}$ of the form $(\mathcal{A}, q, q)$ (because (L_1^0, L_2^0) is an initial state of $\mathcal{A}_{loops}$). But this set is precisely T_0^{ℓ} by definition. Consider now the inductive case $i+1$, for $0 \leq i < m$. We know from our previous remarks that $T_{i+1}^{\ell} = \xi^{\ell}(T_i^{\ell}, a_{i+1})$. By inductive hypothesis, $T_i^{\ell} = L_1^i$, and, therefore, $T_{i+1}^{\ell} = \xi^{\ell}(L_1^i, a_{i+1})$. But the latter is precisely L_1^{i+1}. Proving that $L_2^i = T_i^{r}$ for each $0 \leq i \leq m$ is completely analogous, but this time we do it by induction on i from m to 0 (more formally, we prove by induction on i that $L_2^{m-i} = T_{m-i}^{r}$ for each $0 \leq i \leq m$).

Given a canonical database $\mathcal{G}$ of γ, we denote by $w_{\mathcal{G}}$ the unique word $L_0 a_1 L_1 \dots \ a_m L_m$ over Δ such that $\mathsf{label}(\mathcal{G}) = a_1 \dots a_m$ and the L_i's satisfy (†). In particular, $L_0 a_1 L_1 \dots a_m L_m$ is correct. We now define the 2AFA $\mathcal{A}_{\Gamma'}$. This 2AFA satisfies the following for each canonical database $\mathcal{G}$ of γ: The word $w_{\mathcal{G}}$ is accepted by $\mathcal{A}_{\Gamma'}$ if and only if Γ' evaluates to `true` over $\mathcal{G}$. In order to do this, we define for each disjunct γ' of Γ' a 2AFA $\mathcal{A}_{\gamma'}$ such that the following holds for each canonical database $\mathcal{G}$ of γ: The word $w_{\mathcal{G}}$ is accepted by $\mathcal{A}_{\gamma'}$ iff $\gamma'(\mathcal{G}) = \texttt{true}$. Then $\mathcal{A}_{\Gamma'}$ is obtained by simply taking the union of all the $\mathcal{A}_{\gamma'}$'s.

Let γ' be a disjunct of Γ'. A *connected component* of γ' is a maximal subquery γ'' of γ' such that its underlying undirected graph $\mathcal{U}(\gamma'')$ is connected. Since γ' is acyclic, we can naturally interpret (the underlying undirected graph of) each connected component of γ' as a rooted tree. This allows us to talk about the *parent* or a *child* of a variable in γ', with the obvious meaning. Given a variable x in γ', we define $\texttt{Loops}(x)$ to be the set of atoms of γ' of the form $\mathcal{A}(x, x)$, for $\mathcal{A}$ an NFA over $\Sigma^{\pm}$. Recall that if x and y are variables in γ' then $\texttt{Atoms}(x, y)$ denotes the set of atoms in γ' that mention both x and y. Without loss of generality, we assume that each atom in $\texttt{Atoms}(x, y)$ is of the form $\mathcal{A}(x, y)$ (otherwise, we simply "reverse" $\mathcal{A}$).

Let x and y be variables in γ' such that x is the parent of y in γ'. For the rest of the

proof, we fix an enumeration

$$\mathcal{A}_1(x,y),\ldots,\mathcal{A}_p(x,y)$$

of the elements in $\texttt{Atoms}(x,y)$. We then define $\texttt{Cuts}(x,y)$ to be the set of all tuples of the form $(q_1,\ldots,q_p)$, where q_i is a state of $\mathcal{A}_i$ for each $1 \leq i \leq p$. A tuple $(q_1,\ldots,q_p) \in \texttt{Cuts}(x,y)$ is an *initial cut* (respectively, a *final cut*) if q_i is an initial state (respectively, a final state) of $\mathcal{A}_i$ for each $1 \leq i \leq p$. Given cuts $C = (q_1,\ldots,q_p)$ and $C' = (q'_1,\ldots,q'_p)$ in $\texttt{Cuts}(x,y)$, and a symbol $a \in \Sigma^{\pm}$, we say that C' is *a-reachable* from C, if there is an a-transition in $\mathcal{A}_i$ from q_i to q'_i for each $1 \leq i \leq p$. For a symbol $L \in \texttt{Loops}$ we say that C' is *L-reachable* from C, if $(\mathcal{A}_i, q_i, q'_i) \in L$ for each $1 \leq i \leq p$.

Intuitively, while reading word $w_{\mathcal{G}} = L_0 a_1 L_1 \ldots a_m L_m$, for $\mathcal{G}$ a canonical database of γ, the 2AFA $\mathcal{A}_{\gamma'}$ looks for a homomorphism from γ' to $\mathcal{G}$. The positions of the L_i's represent the nodes of $\mathcal{G}$ and the a_i's represent the edges. At any particular moment, $\mathcal{A}_{\gamma'}$ is trying to map a particular variable x in γ'. Once $\mathcal{A}_{\gamma'}$ maps x, a universal transition takes place to ensure that *all* the children of x can be mapped too. Suppose x is mapped to the node i, for $0 \leq i \leq m$. Then, while reading the symbol L_i, each copy of the 2AFA $\mathcal{A}_{\gamma'}$ guesses a mapping position $0 \leq j \leq m$ for a particular child y of x. This copy then moves from position i to j while it verifies that all the atoms in $\texttt{Atoms}(x,y)$ can be correctly mapped to $w_{\mathcal{G}}$.

Consider an atom of the form $\mathcal{A}(x,y)$ in $\texttt{Atoms}(x,y)$. The 2AFA $\mathcal{A}_{\gamma'}$ then has to verify that there is a a word w over $\Sigma^{\pm}$ that can be read in $\mathcal{A}$ from an initial state q_I to a final state q_F, such that w folds into $a_1 \ldots a_m$ from i to j. Let us assume without loss of generality that $i < j$ (any other case is analogous). Observe that the word w can always be decomposed in the form:

$$w \;=\; w_i a_{i+1} w_{i+1} \ldots a_j w_j,$$

where w_h is a word over $\Sigma^{\pm}$ that can be folded into $a_1 \ldots a_m$ from h to h for each $i \leq h \leq j$. Then $\mathcal{A}_{\gamma'}$ guesses the run on $\mathcal{A}$ over w from q_I to q_F, while it reads $L_i a_{i+1} L_{i+1} \ldots a_j L_j$ from left-to-right. If $\mathcal{A}_{\gamma'}$ reads a_h, with $h \in \{i+1,\ldots,j\}$, and the current guessed state is q then it guesses q' such that there is an a_h-transition in $\mathcal{A}$ from q to q'. If $\mathcal{A}_{\gamma'}$ reads L_h, with $h \in \{i,\ldots,j\}$, and the current guessed state is q, then it guesses q' such that $(\mathcal{A}, q, q') \in L_h$. However, there is a slight complication as the automaton has to do this simultaneously for all atoms in $\texttt{Atoms}(x,y)$. Thus $\mathcal{A}_{\gamma'}$ actually guesses a sequence of cuts $C_0,\ldots,C_{2(j-i)+1}$ from $\texttt{Cuts}(x,y)$, where C_0 is an initial cut, $C_{2(j-i)+1}$ is a final cut, and C_h is α_h-reachable from C_{h-1}, for each $h \in \{1,\ldots,2(j-1)+1\}$, where α_h is the h-th symbol in $L_i a_{i+1} L_{i+1} \ldots a_j L_j$. The 2AFA $\mathcal{A}_{\gamma'}$ is formally defined below.

Let us assume that γ' has K connected components with roots $r_1,\ldots,r_K$, respectively. Then the set of states of $\mathcal{A}_{\gamma'}$ is

$$Q \;=\; \texttt{Cuts} \times \{-1,1\} \cup \{r_1,\ldots,r_K,\mathit{start},\mathit{accept}\},$$

where $\texttt{Cuts}$ is the set $\bigcup\{\texttt{Cuts}(x,y) \mid x \text{ is the parent of } y \text{ in } \gamma'\}$. The initial state is *start* and the final state is *accept*. Next we define the transition function δ' of $\mathcal{A}_{\gamma'}$. For readability, whenever $\delta'(q,a) = \varphi_1 \vee \cdots \vee \varphi_n$, we write $\varphi_i \in \delta'(q,a)$ for each $i \in \{1,\ldots,n\}$. We also assume that whenever $\mathcal{A}_{\gamma'}$ reaches the final state *accept*, it stays in the same state and moves its head all the way to the right. The function δ' is defined as follows:

1. There is an initial transition $(0, r_1) \wedge \cdots \wedge (0, r_K) \in \delta'(start, L)$, for each $L \in \texttt{Loops}$.
2. There is a transition $(1, r_i) \in \delta'(r_i, \alpha)$, for each $i \in \{1, \ldots, K\}$ and $\alpha \in \Sigma^{\pm} \cup \texttt{Loops}$. These transitions look for the position where the root r_i is mapped.
3. For each $i \in \{1, \ldots, K\}$ and $L \in \texttt{Loops}$, we have a "root mapping" transition:

   $$[(0, (I_1, -1)) \vee (0, (I_1, 1))] \wedge \cdots \wedge [(0, (I_t, -1)) \vee (0, (I_t, 1))] \in \delta'(r_i, L),$$

   where (a) $u_1, \ldots, u_t$ are the children of r_i in γ', (b) for each $1 \leq j \leq t$ it is the case that I_j is an initial cut from $\texttt{Cuts}(r_i, u_j)$, and (c) $\texttt{Loops}(r_i)$ is *compatible* with L, i.e., for each atom $\mathcal{A}(r_i, r_i) \in \texttt{Loops}(r_i)$ there are initial and final states q_I and q_F of $\mathcal{A}$ such that $(\mathcal{A}, q_I, q_F) \in L$.

   Intuitively, a "root mapping" transition maps r_i to the current position h, and for each child u_j, it guesses an initial cut and the fact whether u_j will be mapped to the right of h (indicated by the symbol 1) or to the left of h (indicated by the symbol -1).
4. For each $C \in \texttt{Cuts}(x, y)$, with x the parent of y, each $D \in \{-1, 1\}$ and each $a \in \Sigma^{\pm}$, we have an "atom mapping" transition:

   $$(D, (C', D)) \in \delta'((C, D), a)$$

   where $C' \in \texttt{Cuts}(x, y)$ is a-reachable from C if $D = 1$ or a^--reachable from C if $D = -1$.
5. For each $C \in \texttt{Cuts}(x, y)$, with x the parent of y, each $D \in \{-1, 1\}$ and each $L \in \texttt{Loops}$, we have a "loop guessing" transition:

   $$(D, (C', D)) \in \delta'((C, D), L),$$

   where $C' \in \texttt{Cuts}(x, y)$ is L-reachable from C.

   We also have a "variable mapping" transition:

   $$[(0, (I_1, -1)) \vee (0, (I_1, 1))] \wedge \cdots \wedge [(0, (I_t, -1)) \vee (0, (I_t, 1))] \in \delta'((C, D), L),$$

   where (a) $u_1, \ldots, u_t$ are the children of y in γ', (b) for each $1 \leq j \leq t$ we have that I_j is an initial cut from $\texttt{Cuts}(y, u_j)$, (c) $\texttt{Loops}(y)$ is compatible with L, and (d) there is a final cut $C' \in \texttt{Cuts}(x, y)$ which is L-reachable from C.

   Finally, we have a "leaf mapping" transition:

   $$(0, accept) \in \delta'((C, D), L),$$

   whenever (a) y is a leaf in γ', (b) $\texttt{Loops}(y)$ is compatible with L, and (c) there is a final cut $C' \in \texttt{Cuts}(x, y)$ which is L-reachable from C.

It is not hard to see that $\mathcal{A}_{\gamma'}$ is sound and complete, i.e., that for each canonical database $\mathcal{G}$ of γ the 2AFA $\mathcal{A}_{\gamma'}$ accepts $w_{\mathcal{G}}$ iff $\gamma'(\mathcal{G}) = \texttt{true}$.

Recall that $\mathcal{A}_{\Gamma'}$ is the union of the $\mathcal{A}_{\gamma'}$s, for γ' a disjunct of Γ'. We now analyze the size of $\mathcal{A}_{\Gamma'}$. Let γ' be a disjunct of Γ'. Recall that the width $w(\gamma')$ of γ' is the maximum value of $|\texttt{Atoms}(x, y)|$, for x and y distinct variables in γ'. The number of states of $\mathcal{A}_{\gamma'}$ is $2|\texttt{Cuts}| + K + 2$ (where K is the number of connected components of γ'). Recall that $\texttt{Cuts}$ is the set $\bigcup\{\texttt{Cuts}(x, y) \mid x \text{ is the parent of } y \text{ in } \gamma'\}$. Notice that for each pair of distinct

variables x and y in γ', it is the case that $|\texttt{Cuts}(x,y)|$ is bounded by $O(|\gamma'|^{w(\gamma')})$, and therefore, $|\texttt{Cuts}|$ is bounded by $O(|\gamma'| \cdot |\gamma'|^{w(\gamma')}) = O(|\gamma'|^{w(\gamma')+1})$. Furthermore, clearly K is bounded by $|\gamma'|$. Thus, the number of states of $\mathcal{A}_{\gamma'}$ is $O(|\gamma'|^{w(\gamma')+1})$. Since the width $w(\Gamma')$ of Γ' is defined as the maximum value of $w(\gamma')$, for γ' a disjunct of Γ', it follows that the number of states of $\mathcal{A}_{\Gamma'}$ is at most $O(|\Gamma'| \cdot |\Gamma'|^{w(\Gamma')+1}) = O(|\Gamma'|^{w(\Gamma')+2})$.

Now we are ready to define the NFA $\mathcal{A}_{\gamma,\Gamma'}$. This NFA is the product of $\mathcal{A}_{cd}$, $\mathcal{A}_{loops}$ and $\overline{\mathcal{A}_{\Gamma'}}$, where $\overline{\mathcal{A}_{\Gamma'}}$ is the NFA from Lemma 3.17 that accepts the complement of the language accepted by $\mathcal{A}_{\Gamma'}$. From our previous remarks, $\gamma \not\sqsubseteq \Gamma'$ if and only if the language accepted by $\mathcal{A}_{\gamma,\Gamma'}$ is not empty. Furthermore, notice that the number of states of $\mathcal{A}_{\gamma,\Gamma'}$ is bounded by $2^{O(\log|\gamma|+|\Gamma'|^{w(\Gamma')+2})}$.

It is well-known that nonemptiness of an NFA with n states can be checked in nondeterministic $O(\log n)$ space. Following this approach, our algorithm simply checks that the language accepted by $\mathcal{A}_{\gamma,\Gamma'}$ is nonemtpy using nondeterministic $O(\log|\gamma|+|\Gamma'|^{w(\Gamma')+2})$ space. Of course we cannot explicitly construct $\mathcal{A}_{\gamma,\Gamma'}$, as it might be of double exponential size. Instead, we use a standard "on the fly" implementation: We generate the states of $\mathcal{A}_{\gamma,\Gamma'}$ on the fly and whenever is needed, we check if there is a transition from one state to another. Given two states, we can easily decide whether there is a transition in $\mathcal{A}_{\gamma,\Gamma'}$ from one state to the other using only polynomial space in $|\gamma|$ and $|\Gamma'|$. Thus the on the fly implementation actually uses only $O(\log|\gamma|+|\Gamma'|^{w(\Gamma')+2})$ space. By Savitch's theorem, we conclude that containment of γ in Γ' can be solved in deterministic $O((\log|\gamma|+|\Gamma'|^{w(\Gamma')+2})^2) = O(|\Gamma'|^{2w(\Gamma')+4} \cdot \log^2|\gamma|)$ space, which is $O(|\Gamma'|^{d\cdot w(\Gamma')} \cdot |\gamma|^{d'})$ space, for suitable constants $d, d' \geq 1$. This concludes the proof of the lemma.

3.4 Approximations of UC2RPQs

Acyclic UC2RPQs form a good class in terms of complexity of evaluation: They are tractable as opposed to arbitrary C2RPQs (and even CQs) for which the evaluation problem is NP-complete and even hard in parameterized complexity [131]. This motivates our study of approximations of UC2RPQs in the class of acyclic UC2RPQs, which is inspired by recent research on approximations of UCQs. We explain this below.

Evaluating an arbitrary CQ on a big database might be prohibitively expensive. This has led to the recent study of (U)CQ *approximations* in tractable classes [14, 15], in particular, in the class of acyclic (U)CQs. Intuitively, an acyclic UCQ Θ' is an approximation of a UCQ Θ if the following holds: (1) Θ' is contained in Θ (i.e. Θ' returns no false positives with respect to Θ) and (2) Θ' is "as close as possible" to Θ among all acyclic UCQs.

It follows from results and techniques in [15] that approximations of UCQs have good properties.

1. First of all, they always exist, that is, each UCQ has at least one acyclic approximation, and, in addition, such approximation is unique (up to equivalence) and of at most exponential size.

2. Second, for each UCQ Θ, its acyclic approximation Θ' can be computed in single-exponential time.
3. Third, verifying whether an acyclic UCQ Θ' is an approximation of a UCQ Θ is decidable in the second-level of the polynomial hierarchy.

These good properties imply that computing and running the acyclic approximation of a UCQ Θ on a database $\mathcal{D}$ takes time $O(2^{|\Theta|^c} + 2^{|\Theta|^c} \cdot |\mathcal{D}|)$, for a constant $c \geq 1$, which is $O(2^{|\Theta|^c} \cdot |\mathcal{D}|)$. On large databases, this is much better than the general $|\mathcal{D}|^{O(|\Theta|)}$ cost of evaluating Θ over $\mathcal{D}$. Thus, if evaluation of Θ is infeasible or too slow and the quality of its acyclic approximation is good, we may prefer to run this faster approximation instead of Θ.

Here we study acyclic approximations for UC2RPQs, and show that several of the good properties mentioned above for acyclic approximations of UCQs extend to the class of UC2RPQs.

3.4.1 Approximations: Existence and computation

Suppose we want to approximate a UC2RPQ Γ in the class of acyclic UC2RPQs. As explained earlier, we are interested in approximations that are guaranteed to return correct results only. Thus, we are looking for an acyclic UC2RPQ that is *maximally contained* in Γ:

Definition 3.18 (Acyclic approximations) *Let Γ and Γ' be UC2RPQs such that Γ' is acyclic and $\Gamma' \subseteq \Gamma$. Then Γ' is an* acyclic approximation *of Γ if for every acyclic UC2RPQ Γ'' with $\Gamma'' \subseteq \Gamma$ we have that $\Gamma'' \subseteq \Gamma'$.*

It is worth noticing that the definition of approximations in [15] is different, but equivalent to this one.

An important property of UCQs is that each query in the class has an acyclic approximation, and that such an approximation is unique. We can prove that this is also true for the class of UC2RPQs.

Theorem 3.19 *Each UC2RPQ has a unique acyclic approximation (up to equivalence).*

As a corollary to the proof of Theorem 3.19 we get the following important result about the computation and size of approximations:

Corollary 3.20 *There exists an* Expspace *algorithm that takes as input a UC2RPQ Γ and computes the approximation Γ' of Γ. This approximation is of at most exponential size.*

It follows from Corollary 3.20 that approximations of UC2RPQs are meaningful. In fact, computing and running the acyclic approximation of a UC2RPQ Γ on a graph database $\mathcal{G}$ takes time

$$O\left(2^{2^{|\Gamma|^c}} + 2^{|\Gamma|^{c'}} \cdot |\mathcal{G}|^2\right),$$

for some constant $c, c' \geq 1$, which is $O\big(2^{2^{|\Gamma|^d}} \cdot |\mathcal{G}|^2\big)$, for some constant $d \geq 1$. In terms of data complexity this is only $O(|\mathcal{G}|^2)$, like the data complexity of 2RPQs. This is much faster than $|\mathcal{G}|^{O(|\Gamma|)}$ – the order of the evaluation problem for Γ on $\mathcal{G}$ – on large datasets.

We finish by proving that there is an important aspect of approximations that is harder for UC2RPQs than for UCQs: the identification problem, i.e. verifying if a query is an approximation of another. We mentioned above that checking whether an acyclic UCQ Θ' is an approximation of a UCQ Θ can be solved in the second-level of the polynomial hierarchy [15]; more precisely, it is complete for the class DP, that consists of all those languages that are the intersection of an NP and a coNP problem [130]. This problem is considerably harder for UC2RPQs:

Proposition 3.21 *Let Γ and Γ' be UC2RPQs such that Γ' is acyclic. The problem of verifying whether Γ' is an acyclic approximation of Γ is* Expspace*-complete.*

We prove Theorem 3.19, Corollary 3.20 and Proposition 3.21 in the following section.

3.4.2 Proofs of results

All results in Section 3.4.1 follow from an important lemma that states that there exists an Expspace algorithm that, on input a UC2RPQ Γ, computes an acyclic UC2RPQ Γ_{app} – of at most exponential size – which is a maximum for the class of acyclic UC2RPQs that are contained in Γ.

Lemma 3.22 *There exists an* Expspace *algorithm that given a UC2RPQ Γ computes an acyclic UC2RPQ Γ_{app} such that:*

1. $\Gamma_{\text{app}} \subseteq \Gamma$.
2. *For every acyclic UC2RPQ Γ' such that $\Gamma' \subseteq \Gamma$ it is the case that $\Gamma' \subseteq \Gamma_{\text{app}}$.*
3. *The number of atoms and variables in each disjunct of Γ_{app} is at most polynomial in $|\Gamma|$.*
4. *The number of disjuncts and the size of each 2RPQ appearing in Γ_{app} is at most exponential in $|\Gamma|$.*

In particular, the size of Γ_{app} is at most exponential in $|\Gamma|$.

Before proving Lemma 3.22 we show how the results in Section 3.4.1 follow from it.

Proofs of Theorem 3.19, Corollary 3.20 and Proposition 3.21: The algorithm in Lemma 3.22 computes for each UC2RPQ Γ a query Γ_{app} which is the maximum for the class of acyclic UC2RPQs that are contained in Γ. In other words, Γ_{app} is an acyclic approximation of Γ. This approximation must be unique (up to equivalence) by definition.

In order to prove Corollary 3.20, we use the algorithm in Lemma 3.22 to compute the approximation Γ_{app} of a UC2RPQ Γ. The algorithm runs in Expspace and its output Γ_{app}

is of at most exponential size in $|\Gamma|$.

Finally, we prove Proposition 3.21. Checking whether the acyclic UC2RPQ Γ' is an approximation of the UC2RPQ Γ is equivalent to checking whether (†) $\Gamma' \subseteq \Gamma$ and (††) $\Gamma_{\text{app}} \subseteq \Gamma'$. Indeed, if this is the case then $\Gamma' \equiv \Gamma_{\text{app}}$, and, therefore, Γ' is the approximation of Γ. We prove that (†) and (††) can be checked in Expspace.

It directly follows from the first part of Proposition 3.12 that (†) can be verified in Expspace. Furthermore, checking (††) requires computing Γ_{app} and then checking whether $\Gamma_{\text{app}} \subseteq \Gamma'$. The first step can be done in Expspace from Lemma 3.22, while the second one can be carried out in exponential space, where the exponent depends only on $|\Gamma'|$ and $\mathsf{maxvar}(\Gamma_{\text{app}})$ (again from the first part of Proposition 3.12). Part (3) of Lemma 3.22 implies that $\mathsf{maxvar}(\Gamma_{\text{app}})$ is at most polynomial in the size of $|\Gamma|$. Therefore, the second step can also be done in Expspace. We conclude that checking whether Γ' is an acyclic approximation of Γ can be carried out in Expspace.

For the lower bound, observe that $\Gamma' \subseteq \Gamma$ if and only if Γ' is an acyclic approximation of $\Gamma \wedge \Gamma'$. The result now follows since Proposition 3.12 states that containment of CRPQs is Expspace-hard even when Γ' is acyclic. □

We devote the rest of this section to proving Lemma 3.22. We start by presenting some definitions and a technical lemma which is crucial for our proof. We conclude by explaining the construction of the acyclic approximation Γ_{app} for a UC2RPQ Γ.

A technical lemma

The lemma requires some terminology which we define next.

Pseudo acyclic graph databases. A graph database $\mathcal{G}$ is *connected* if for each pair of nodes n, n' in $\mathcal{G}$, there is a path from n to n' in $\mathcal{G}^{\pm}$ (that is, if the underlying undirected graph of $\mathcal{G}$ is connected). A *subgraph database* of $\mathcal{G} = (N, E)$ is a graph database $\mathcal{G}' = (N', E')$ such that $N' \subseteq N$ and $E' \subseteq E$. A *connected component* of $\mathcal{G}$ is a maximal connected subgraph database of $\mathcal{G}$. We call $\mathcal{G}$ *pseudo-acyclic* if each connected component of $\mathcal{G}$ can be obtained from a tree T as follows:

- Each edge $\{n, n'\}$ in T is replaced by a finite number of semipaths from n to n' whose internal nodes are "fresh", i.e., they are disjoint from any other nodes in $\mathcal{G}$.
- For some (maybe none) nodes n in T we add a finite number of semipaths from n to n, again with fresh internal nodes.

The fresh internal nodes of the paths in the previous definition are the *internal* nodes of the pseudo-acyclic database $\mathcal{G}$. The rest of the nodes (that is, the nodes that are inherited from T) are called *external* nodes.

We will slightly abuse notation and talk about the *parent* (respectively, *ancestor*) of an external node n in the pseudo-acyclic graph database $\mathcal{G}$. By this we refer to the external node

in $\mathcal{G}$ that corresponds to the parent (respectively, an ancestor) of n in the tree T. Similarly, we talk about the *least common ancestor* of two external nodes in $\mathcal{G}$, which refers to the external node in $\mathcal{G}$ that corresponds to their least common ancestor in T.

The following proposition is straightforward.

Proposition 3.23 *Let Γ be an acyclic UC2RPQ. Then each canonical database of Γ is pseudo-acyclic.*

We also define pseudo-acyclicity for graphs in the obvious way: instead of adding semi-paths to T, we add undirected paths. For pseudo-acyclic graphs we define the concepts of external/internal node and parent of a node as before.

Types for UC2RPQs. Let Γ be UC2RPQ over Σ. Recall that $\mathcal{T}_\Gamma$ denotes the set of all triples of the form $(\mathcal{A}, q, q')$, where $\mathcal{A}$ is a 2RPQ mentioned in some atom of Γ and q and q' are states of $\mathcal{A}$. A Γ-*type* is a 4-tuple $\tau = (\tau_1, \tau_2, \tau_3, \tau_4)$, where τ_i is a subset of $\mathcal{T}_\Gamma$, for each $1 \leq i \leq 4$. Let w be a word over $\Sigma^\pm$ of length $\ell \geq 0$. The Γ-*type of* w is the Γ-type $\tau_w = (\tau_{it}, \tau_{ti}, \tau_{ii}, \tau_{tt})$ such that $(\mathcal{A}, q, q')$ belongs to τ^* if and only if there is a word u over $\Sigma^\pm$ and a run of $\mathcal{A}$ over u from state q to q', and u can be folded into w from (i) 0 to ℓ, if $* = it$, (ii) from ℓ to 0, when $* = ti$, (iii) from 0 to 0, if $* = ii$, and (iv) from ℓ to ℓ, when $* = tt$ (i stands for "initial" and t for "terminal").

Given a Γ-type $\tau = (\tau_1, \tau_2, \tau_3, \tau_4)$, we define $\mathcal{L}^{\subseteq}(\tau)$ to be the language of words w over $\Sigma^\pm$ such that τ is coordinate-wise contained in the Γ-type τ_w of w. As it turns out, $\mathcal{L}^{\subseteq}(\tau)$ defines a regular language:

Lemma 3.24 *Let Γ be a UC2RPQ and τ a Γ-type. Then the language $\mathcal{L}^{\subseteq}(\tau)$ is regular and can be defined by an NFA $\mathcal{A}_\tau$ over $\Sigma^\pm$ of at most exponential size in $|\Gamma|$.*

Proof: We start introducing some terminology. A *two-way nondeterminisitic finite automaton* (2NFA) [100, 152] is a tuple $\mathcal{B} = (\Sigma, Q, Q_0, \delta, F)$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq S$ is the set of initial states, $\delta : Q \times \Sigma \to 2^{Q \times \{-1,0,1\}}$ is the transition function, and $F \subseteq Q$ is the set of final states. Intuitively, a transition indicates both the new state of the automaton and whether the head should move left (-1), right (1), or stay in place (0). A *configuration* of $\mathcal{B}$ is a pair (q, j) consisting of a state $q \in Q$ and a position represented as an integer $j \geq 0$. The sequence $(q_0, j_0), \ldots, (q_m, j_m)$ of configurations of $\mathcal{B}$ is an *accepting run* of $\mathcal{B}$ on a word $w = a_0 \ldots a_{\ell-1} \in \Sigma^*$, for $\ell \geq 0$, if (i) $q_0 \in Q_0$, (ii) $j_0 = 0$, (iii) $q_m \in F$, (iv) $j_m = \ell$, and (v) for each $i \in \{0, \ldots, m-1\}$ we have that $0 \leq j_i < \ell$ and there is some $(q', d) \in \delta(q_i, a_{j_i})$ such that $q_{i+1} = q'$ and $j_{i+1} = j_i + d$. The 2NFA $\mathcal{B}$ *accepts* w if it has an accepting run on w. The following fact was proved in [152]:

Proposition 3.25 [152] *Given a 2NFA with n states, one can construct an equivalent NFA with $O(2^{n \log n})$ states.*

Assume that $\mathcal{A}_1, \ldots, \mathcal{A}_m$ is an enumeration of all the 2RPQs over Σ mentioned in Γ. For the Γ-type τ and $1 \leq i \leq 4$, we denote by $\tau(i)$ the i-th coordinate of τ. Let $1 \leq j \leq m$ and

assume that $\mathcal{A}_j = (\Sigma^{\pm}, Q, Q_0, \delta, F)$, where δ is a transition function of the form $Q \times \Sigma \to 2^Q$, and that q and q' are states of $\mathcal{A}_j$. We then define a 2NFA $\mathcal{B}_{it}(j, q, q')$ that accepts the following language:

$$\{z \in (\Sigma^{\pm})^* \mid (\mathcal{A}_j, q, q') \in \tau_z(1), \text{ where } \tau_z \text{ is the } \Gamma\text{-type of } z\}.$$

Intuitively, $\mathcal{B}_{it}(j, q, q')$ guesses a folding from the first to the last position of z and verifies that such folding can be read from state q to q' in $\mathcal{A}_j$ (as this implies, by definition, that $(\mathcal{A}_j, q, q')$ belongs to $\tau_z(1)$). It is not hard to see that $\mathcal{B}_{it}(j, q, q')$ can be constructed in such a way that its size is bounded by $O(|\mathcal{A}_j|)$, and thus by $O(|\Gamma|)$.

Similarly, we define a 2NFA $\mathcal{B}_{ti}(j, q, q')$ that accepts the language:

$$\{z \in (\Sigma^{\pm})^* \mid (\mathcal{A}_j, q, q') \in \tau_z(2), \text{ where } \tau_z \text{ is the } \Gamma\text{-type of } z\}.$$

This NFA also guesses a folding from the first to the last position of z – which represents the "inverse" of a folding from the last position of z to the first one – and then verifies that such a folding can be read "backwards" from state q' to q in $\mathcal{A}_j$ (as this implies, by definition, that $(\mathcal{A}_j, q, q')$ belongs to $\tau_2(z)$). As before, the 2NFA $\mathcal{B}_{ti}(j, q, q')$ can be easily constructed in such a way that its size is bounded by $O(|\mathcal{A}_j|)$, and thus by $O(|\Gamma|)$.

Let $\mathcal{A}_{it}(j, q, q')$ and $\mathcal{A}_{ti}(j, q, q')$ be the NFAs over $\Sigma^{\pm}$ which are equivalent to the 2NFAs $\mathcal{B}_{it}(j, q, q')$ and $\mathcal{B}_{ti}(j, q, q')$, respectively, according to Proposition 3.25. Thus, the number of states in $\mathcal{A}_{it}(j, q, q')$ and $\mathcal{A}_{ti}(j, q, q')$ is bounded by $O(2^{|\Gamma|^c})$, for some constant $c \geq 1$.

We now explain how to construct an NFA $\mathcal{A}_{ii}(j, q, q')$ which accepts the language:

$$\{z \in (\Sigma^{\pm})^* \mid (\mathcal{A}_j, q, q') \in \tau_z(3), \text{ where } \tau_z \text{ is the } \Gamma\text{-type of } z\}.$$

Notice that if $z = a_0 \dots a_{\ell-1}$, where each a_k is a symbol in $\Sigma^{\pm}$, then the triple $(\mathcal{A}_j, q, q')$ belongs to $\tau_z(3)$ if and only if there is a state p of $\mathcal{A}_j$ and a position $0 \leq k \leq \ell - 1$ such that $(\mathcal{A}_j, q, p) \in \tau_{z_k}(1)$ and $(\mathcal{A}_j, p, q') \in \tau_{z_k}(2)$, where for each $0 \leq k \leq \ell - 1$ we have that $z_k = a_0 \dots a_k$. Then for each state p of $\mathcal{A}_j$ we define an NFA $\mathcal{A}_{ii}(j, q, p, q')$ which accepts the language:

$$\{z \in (\Sigma^{\pm})^* \mid (\mathcal{A}_j, q, p) \in \tau_{z'}(1) \text{ and } (\mathcal{A}_j, p, q') \in \tau_{z'}(2), \text{ for some prefix } z' \text{ of } z\}.$$

It is easy to define the NFA $\mathcal{A}_{ii}(j, q, p, q')$ by (i) taking the cross product between $\mathcal{A}_{it}(j, q, p)$ and $\mathcal{A}_{ti}(j, p, q')$, (ii) adding an ε-transition from each final state of this cross product to a fresh state f, (iii) adding a transition a from f to f, for each $a \in \Sigma^{\pm}$, and (iv) letting f be the unique final state of the resulting NFA. The NFA $\mathcal{A}_{ii}(j, q, q')$ is then defined as the union of all the NFAs in the set:

$$\{\mathcal{A}_{ii}(j, q, p, q') \mid p \text{ is a state in } \mathcal{A}_j\}.$$

The number of states of $\mathcal{A}_{ii}(j, q, q')$ is $O(2^{|\Gamma|^{c'}})$, for some constant $c' \geq 1$.

Using the previous idea it is then straightforward to construct an NFA $\mathcal{A}_{tt}(j, q, q')$ which accepts the language:

$$\{z \in (\Sigma^{\pm})^* \mid (\mathcal{A}_j, q, q') \in \tau_z(4), \text{ where } \tau_z \text{ is the } \Gamma\text{-type of } z\}.$$

Again, the number of states of $\mathcal{A}_{tt}(i,q,q')$ is $O(2^{|\Gamma|^{c'}})$.

In order to construct the NFA $\mathcal{A}_\tau$ which accepts the language $\mathcal{L}^{\subseteq}(\tau)$ we proceed as follows. We first define NFAs $\mathcal{A}_{it}, \mathcal{A}_{ti}, \mathcal{A}_{ii}$ and $\mathcal{A}_{tt}$ as the product of the NFAs in the set:

$$\{\mathcal{A}_{it}(j,q,q') \mid (\mathcal{A}_j,q,q') \in \tau(1),\ 1 \leq j \leq m\}$$
$$\{\mathcal{A}_{ti}(i,q,q') \mid (\mathcal{A}_j,q,q') \in \tau(2),\ 1 \leq j \leq m\}$$
$$\{\mathcal{A}_{ii}(j,q,q') \mid (\mathcal{A}_j,q,q') \in \tau(3),\ 1 \leq j \leq m\}$$
$$\{\mathcal{A}_{tt}(j,q,q') \mid (\mathcal{A}_j,q,q') \in \tau(4),\ 1 \leq j \leq m\},$$

respectively. Then we define $\mathcal{A}_\tau$ as the product of $\mathcal{A}_{it}, \mathcal{A}_{ti}, \mathcal{A}_{ii}$ and $\mathcal{A}_{tt}$. It is straightforward to check that the language accepted by $\mathcal{A}_\tau$ is precisely $\mathcal{L}^{\subseteq}(\tau)$. Furthermore, the number of states of $\mathcal{A}_\tau$ is $O(2^{|\Gamma|^d})$, for some constant $d \geq 1$. Thus the size of $\mathcal{A}_\tau$ is at most exponential in $|\Gamma|$, as required. □

The technical lemma. We are now ready to state the main lemma of this section.

Lemma 3.26 *Let $\mathcal{G}$ be a pseudo-acyclic graph database and $\bar{n}$ a tuple of external nodes in $\mathcal{G}$. Let $\Gamma(\bar{x})$ be a UC2RPQ with $|\bar{n}| = |\bar{x}|$ such that $\bar{n} \in \Gamma(\mathcal{G})$. There exists an acyclic C2RPQ $\alpha(\bar{x})$ and a finite set $\mathcal{R}_\Gamma$ of 2RPQs over Σ such that:*

1. *$\mathcal{R}_\Gamma$ can be constructed from Γ in single exponential time.*
2. *$\bar{n} \in \alpha(\mathcal{G})$ and $\alpha \subseteq \Gamma$.*
3. *The number of atoms in α is at most polynomial in $|\Gamma|$.*
4. *Each 2RPQ mentioned in α belongs to $\mathcal{R}_\Gamma$.*

Proof: Since $\bar{n} \in \Gamma(\mathcal{G})$, there is a disjunct γ of Γ such that $\bar{n} \in \gamma(\mathcal{G})$. Assume that γ is of the form $\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1,y_1'),\ldots,\mathcal{A}_m(y_m,y_m')$. Since $\bar{n} \in \gamma(\mathcal{G})$, there is a mapping h from the variables of γ to the nodes of $\mathcal{G}$ that satisfies $h(\bar{x}) = \bar{n}$ and a path π_i from $h(y_i)$ to $h(y_i')$ in $\mathcal{G}^{\pm}$ such that $\mathsf{label}(\pi_i)$ is accepted by $\mathcal{A}_i$, for each $1 \leq i \leq m$. We assume without loss of generality that for each $1 \leq i \leq m$ it is the case that π_i is of minimal length among the set of paths from $h(y_i)$ to $h(y_i')$ in $\mathcal{G}^{\pm}$ whose label is accepted by $\mathcal{A}_i$.

We define a subset $\mathcal{V}$ of the nodes of $\mathcal{G}$ that will help us define the variable set of the C2RPQ $\alpha(\bar{x})$. Let $\mathcal{I}$ be the set $\{h(z) \mid z \text{ is a variable in } \gamma\}$. We define $\mathcal{I}'$ to be the set of nodes obtained from $\mathcal{I}$ by adding, for each node u in $\mathcal{I}$ that is an internal node of $\mathcal{G}$, its associated external nodes in $\mathcal{G}$ (that is, the external nodes that are at the beginning and at the end of the semipath where u belongs). We then define $\mathcal{V}$ as the set of nodes which is obtained from $\mathcal{I}'$ by adding the least common ancestor in $\mathcal{G}$ of each pair n, n' of external nodes from $\mathcal{I}'$ in the same connected component of $\mathcal{G}$. Observe that each node in $\bar{n}$ belongs to $\mathcal{I}$ (since $h(\bar{x}) = \bar{n}$), and thus to $\mathcal{V}$. Also notice that $|\mathcal{V}|$ is $O(|\gamma|^2)$. Indeed, $|\mathcal{I}| \leq var$, where var is the number of variables in γ. Moreover, we have that $|\mathcal{I}'| \leq 3|\mathcal{I}|$ and $|\mathcal{V}| \leq |\mathcal{I}'| + |\mathcal{I}'|^2$, as the number of least common ancestors we add to $\mathcal{V}$ is bounded by the number of pairs in $|\mathcal{I}'|$.

For each $1 \leq i \leq m$, we decompose the path π_i according to $\mathcal{V}$ in the natural way. That is,

the path π_i is decomposed as the concatenation $\pi_i^1 \cdots \pi_i^{k_i}$ of paths $\pi_i^1, \ldots, \pi_i^{k_i}$ such that each path π_i^j starts and ends at a node in $\mathcal{V}$ and each internal node of π_i^j is not in $\mathcal{V}$. As π_i itself starts and ends at $h(y_i)$ and $h(y_i')$, respectively, which are nodes in $\mathcal{V}$, this decomposition is always possible. The decomposition is also unique (assuming that no π_i^j is the empty path, except in the case when π_i itself is empty).

Let $\pi_i^1 \cdots \pi_i^{k_i}$ be the decomposition of the path π_i, for $1 \leq i \leq m$. By the minimality of π_i, it follows that $k_i \leq |Q_i| \cdot |\mathcal{V}| - 1$, where Q_i is the set of states of the NFA $\mathcal{A}_i$. Assume for the sake of contradiction that $k_i \geq |Q_i| \cdot |\mathcal{V}|$. We fix an accepting run $q_0 \cdots q_\ell$ of $\mathcal{A}_i$ over $\mathsf{label}(\pi_i)$ (i.e., each q_j is a state in Q_i, q_0 and q_ℓ are an initial and final state of $\mathcal{A}_i$, respectively, and q_{j+1} is obtained from q_j according to the transition function of $\mathcal{A}_i$). For each $1 \leq j \leq k_i$, let n^j be the last node of π_i^j (which belongs to $\mathcal{V}$) and q^j be the state of the run $q_0 \cdots q_\ell$ which is associated with the prefix $\mathsf{label}(\pi_i^1 \ldots \pi_i^j)$ of $\mathsf{label}(\pi_i^1 \ldots \pi_i^{k_i})$. In particular, $n^{k_i} = h(y_i')$ and $q^{k_i} = q_\ell$. We also define $n^0 = h(y_i)$ and $q^0 = q_0$. Since $k_i + 1 > |\mathcal{V}| \cdot |Q_i|$, there are two positions $0 \leq \ell_1 < \ell_2 \leq k_i$ such that $n^{\ell_1} = n^{\ell_2}$ and $q^{\ell_1} = q^{\ell_2}$. It is clear then that we can ignore the subpath $\pi_i^{\ell_1+1} \cdots \pi_i^{\ell_2}$. This contradicts the minimality of π_i. We conclude that the number of paths involved in all the decompositions of the π_i's is at most $(|Q_1|+\cdots+|Q_m|)\cdot|\mathcal{V}|$. Clearly, $|Q_1| + \cdots + |Q_m|$ is polynomially bounded by $|\gamma|$. Furthermore, we know that $|\mathcal{V}|$ is $O(|\gamma|^2)$ and $|\gamma| \leq |\Gamma|$. We then conclude that the number of paths involved in all the decompositions of the π_i's is polynomial in $|\Gamma|$.

Let π_i^j be a path in the decomposition of π_i, for $1 \leq i \leq m$ and $1 \leq j \leq k_i$. We denote by $\ell(i,j)$ the length of the word $\mathsf{label}(\pi_i^j)$, and define (i) $c(j) := \sum_{t=1}^{j-1} \ell(i,t)$, and (ii) $d(j) := \sum_{t=1}^{j} \ell(i,t)$ (we set $c(1)$ to be 0). Let $q_0 \cdots q_\ell$ be an accepting run of $\mathcal{A}_i$ over $\mathsf{label}(\pi_i)$. Then E_i^j is an NFA obtained from $\mathcal{A}_i$ by setting $q_{c(j)}$ and $q_{d(j)}$ as the unique initial and final state, respectively. Notice that $\mathsf{label}(\pi_i^j)$ is accepted by E_i^j. Furthermore, E_i^j is of at most polynomial size in $|\mathcal{A}_i|$, and therefore, in $|\Gamma|$.

We now define a C2RPQ $\alpha'(\bar{x})$, which will be the basis for constructing the C2RPQ $\alpha(\bar{x})$ from the statement of the lemma. The variable set of α' is $\{z_n \mid n \in \mathcal{V}\}$. The free variables are $\bar{x} = (z_{p_1}, \ldots, z_{p_r})$, assuming that $\bar{n} = (p_1, \ldots, p_r)$. Finally, for each $1 \leq i \leq m$ and $1 \leq j \leq k_i$, the atom $E_i^j(z_n, z_{n'})$ is in α', where n and n' are the initial and terminal nodes of π_i^j, respectively. The C2RPQ $\alpha'(\bar{x})$ is not necessarily acyclic, but it can be turned into an acyclic C2RPQ $\alpha(\bar{x})$ satisfying conditions (2) and (3) in the statement of the lemma as we shall see later. But before explaining how to turn α' into α, it is important to show that $\alpha'(\bar{x})$ satisfies conditions (2) and (3) in the statement of the lemma (recall that (2) states that $\bar{n} \in \alpha'(\mathcal{G})$ and $\alpha' \subseteq \Gamma$, while (3) states that the number of atoms in α' is at most polynomial in $|\Gamma|$).

First, α' satisfies condition (3) of the lemma. In fact, the number of atoms in α' is at most the number of paths π_i^j's involved in the decompositions of the π_i's, which is at most polynomial in $|\Gamma|$. Now we prove that α' satisfies condition (2). First, consider the mapping g from the variables of α' to $\mathcal{G}$ that maps each z_n to n. Since $\mathsf{label}(\pi_i^j)$ satisfies E_i^j, this mapping is actually a homomorphism. It follows that $\bar{n} \in \alpha'(\mathcal{G})$. In order to show that $\alpha' \subseteq \Gamma$, we use Proposition 3.16. Let $\mathcal{G}'$ be a canonical database for α' with associated mapping ν. Consider the mapping f from the variables of γ to $\mathcal{G}'$ that maps each variable y to $\nu(z_{h(y)})$. The mapping f is well-defined since $h(y) \in \mathcal{I} \subseteq \mathcal{V}$, and thus $z_{h(y)}$ is a variable of α'. Now we

show that for each $1 \leq i \leq m$, there is a path from $\nu(z_{h(y_i)})$ to $\nu(z_{h(y'_i)})$ in $(\mathcal{G}')^{\pm}$ whose label is accepted by $\mathcal{A}_i$. We can simulate the path $\pi_i = \pi_i^1 \cdots \pi_i^{k_i}$ in $\mathcal{G}^{\pm}$ by the path $\chi_i^1 \cdots \chi_i^{k_i}$ in $(\mathcal{G}')^{\pm}$, where χ_i^j is the semipath in the canonical database $\mathcal{G}'$ that is associated with the atom of α' which is labeled E_i^j. By construction, there is a run of $\mathcal{A}_i$ over χ_i^j from $q_{c(j)}$ to $q_{d(j)}$, and thus $\mathsf{label}(\chi_i^1 \dots \chi_i^{k_i})$ is accepted by $\mathcal{A}_i$ since it can be read in $\mathcal{A}_i$ from q_0 to q_ℓ. Therefore, f is a homomorphism. Moreover, $f(\bar{x}) = \nu(\bar{x})$. It follows that $\nu(\bar{x}) \in \gamma(\mathcal{G}') \subseteq \Gamma(\mathcal{G}')$. By Proposition 3.16, we conclude that $\alpha' \subseteq \Gamma$.

As mentioned before, the problem with α' is that it is not necessarily acyclic. In fact, since $\mathcal{G}$ is pseudo-acyclic, the underlying graph $\mathcal{U}(\alpha')$ of α' could be pseudo-acyclic as opposed to acyclic. (Here we assume that the rooted tree used to construct the pseudo-acyclic graph $\mathcal{U}(\alpha')$ is the natural one, that is, the sets of external and internal nodes of $\mathcal{U}(\alpha')$ are precisely $\{z_t \mid t \text{ is an external node in } \mathcal{V}\}$ and $\{z_t \mid t \text{ is an internal node in } \mathcal{V}\}$, respectively). For this to happen, the homomorphism h from γ to $\mathcal{G}$ must map two distinct variables in γ to internal nodes u and u' in two different semipaths π and π' in $\mathcal{G}$, such that π and π' connect exactly the same pair (n, n') of external nodes of $\mathcal{G}$. In such a case, $\mathcal{U}(\alpha')$ might contain a cycle $n \to u \to n' \to u' \to n$.

Therefore, in order for $\mathcal{U}(\alpha')$ to be pseudo-acyclic but not acyclic, it must contain *bad* paths, which are simple paths of the form $uu_1 \dots u_k u'$, where u and u' are distinct external nodes of $\mathcal{U}(\alpha')$ and $u_1, \dots, u_k$ are internal ones (notice that $k \geq 1$, i.e., at least one internal node belongs to a bad path). In other words, if $\mathcal{U}(\alpha')$ does not contain bad paths then it is actually acyclic. Thus, to obtain our desired acyclic query α we can modify α' in such a way that we eliminate all the bad paths from $\mathcal{U}(\alpha')$. This is what we do next.

Let us consider a bad path b in $\mathcal{U}(\alpha')$ of the form $z_{n_0} \cdots z_{n_{p+1}}$, where (i) $p \geq 1$, (ii) $z_{n_1}, \dots, z_{n_p}$ are internal nodes of $\mathcal{U}(\alpha')$, and (iii) z_{n_0} and $z_{n_{p+1}}$ are distinct external nodes of $\mathcal{U}(\alpha')$. We assume without loss of generality that z_{n_0} is the parent of $z_{n_{p+1}}$ in $\mathcal{U}(\alpha')$. Therefore, we have that n_0 and n_{p+1} are external nodes in $\mathcal{G}$ such that n_0 is the parent of n_{p+1}, and there is a semipath ζ from n_0 to n_{p+1} such that $n_1, \cdots, n_p$ are precisely the internal nodes of ζ that belong to $\mathcal{V}$ (since ζ is a semipath it does not repeat nodes by definition). For $k \in \{0, \dots, p\}$, let ζ_k be the subpath of ζ that starts at n_k and ends at n_{k+1}. Furthermore, for $k, k' \in \{0, \dots, p+1\}$, let $\mathsf{Paths}_{k,k'}$ be the set of paths in some decomposition of some π_i that starts at n_k and end at $n_{k'}$. Notice that each path in $\mathsf{Paths}_{k,k}$, with $k \in \{1, \dots, p\}$, satisfies that all its internal nodes are contained either in ζ_{k-1} or in ζ_k. For each $k \in \{1, \dots, p\}$, we define $\mathsf{Paths}^{\uparrow}_{k,k}$ to be the set of paths in $\mathsf{Paths}_{k,k}$ whose internal nodes are in ζ_{k-1}. Analogously, $\mathsf{Paths}^{\downarrow}_{k,k} = \mathsf{Paths}_{k,k} \setminus \mathsf{Paths}^{\uparrow}_{k,k}$, i.e., $\mathsf{Paths}^{\downarrow}_{k,k}$ contains all the paths in $\mathsf{Paths}_{k,k}$ with internal nodes in ζ_k. For convenience, we define $\mathsf{Paths}^{\downarrow}_{0,0} = \mathsf{Paths}^{\uparrow}_{p+1,p+1} = \emptyset$.

In addition, for each $k \in \{0, \dots, p\}$ we define a Γ-type $\tau_k = (\tau_{it}, \tau_{ti}, \tau_{ii}, \tau_{tt})$ as follows. The set τ_{it} contains the triple $(\mathcal{A}, q, q')$ if and only if there is a path $\pi_i^j \in \mathsf{Paths}_{k,k+1}$ such that $\mathcal{A} = \mathcal{A}_i$ and there is a run of $\mathcal{A}_i$ over $\mathsf{label}(\pi_i^j)$ from state q to q'. We define τ_{ti}, τ_{ii} and τ_{tt} analogously, but this time replacing $\mathsf{Paths}_{k,k+1}$ with $\mathsf{Paths}_{k+1,k}$, $\mathsf{Paths}^{\downarrow}_{k,k}$ and $\mathsf{Paths}^{\uparrow}_{k+1,k+1}$, respectively. Observe that since $\mathsf{Paths}^{\downarrow}_{0,0} = \mathsf{Paths}^{\uparrow}_{p+1p+1} = \emptyset$, we have that $\tau_{ii} = \emptyset$ for τ_0 and $\tau_{tt} = \emptyset$ for τ_p.

We then define $\mathcal{A}_{\tau_k}$ to be the NFA from Lemma 3.24, i.e., $\mathcal{A}_{\tau_k}$ defines the language $\mathcal{L}^{\subseteq}(\tau_k)$.

Notice that $\mathsf{label}(\zeta_k)$ is accepted by $\mathcal{A}_{\tau_k}$. Furthermore, τ_k, and therefore $\mathcal{A}_{\tau_k}$, is completely determined by ζ_k. For the sake of presentation we thus denote $\mathcal{A}_{\tau_k}$ by $\mathcal{A}_{\zeta_k}$ from now on. Finally, we define $\mathcal{I}_b$ to be the set of atoms

$$\{\mathcal{A}_{\zeta_0}(z_{n_0}, z_{n_1}), \mathcal{A}_{\zeta_1}(z_{n_1}, z_{n_2}), \cdots, \mathcal{A}_{\zeta_p}(z_{n_p}, z_{n_{p+1}})\}$$

and $\mathcal{O}_b$ to be the set of atoms in α' of the form $E_i^j(z_t, z_{t'})$, where (t, t') is a pair in $\{n_0, \ldots, n_{p+1}\} \times \{n_0, \ldots, n_{p+1}\} \setminus \{(n_0, n_0), (n_{p+1}, n_{p+1})\}$.

We define an auxiliary query α'' as the C2RPQ which is obtained from α' by simultaneously replacing all atoms in $\mathcal{O}_b$ with those in $\mathcal{I}_b$, for each bad path b in $\mathcal{U}(\alpha')$. Since $\mathcal{O}_b$ and $\mathcal{O}_{b'}$ are disjoint for different bad paths b and b', the query α'' is well-defined. We claim that α'' satisfies conditions (2) and (3) of the lemma. For condition (3), note that for each bad path $b = z_{n_0} \cdots z_{n_{p+1}}$ in $\mathcal{U}(\alpha')$ the number of internal nodes p is bounded by the number of variables of γ, and in particular, by $|\Gamma|$. It follows that $|\mathcal{I}_b| \leq |\Gamma| + 1$. Since the number of bad paths is also polynomial in $|\Gamma|$, we conclude that the number of atoms in α'' is still polynomial in $|\Gamma|$.

Now we show that α'' satisfies condition (2). As noticed before, for an atom of the form $\mathcal{A}_{\zeta_i}(\cdot, \cdot)$ in α'', we have that $\mathsf{label}(\zeta_i)$ is accepted by $\mathcal{A}_{\zeta_i}$. Putting this together with (i) the fact that the function that maps the variable z_n in α' to the node n in $\mathcal{G}$ is a homomorphism, and (ii) the way in which α'' is constructed from α', we conclude that $\bar{n} \in \alpha''(\mathcal{G})$. We prove that $\alpha'' \subseteq \Gamma$ using Proposition 3.16. Let $\mathcal{G}'$ be a canonical database for α'' with associated mapping ν, and assume that f is the mapping that sends each variable y in γ to $\nu(z_{h(y)})$ (as before, f is well-defined). Clearly $f(\bar{x}) = \nu(\bar{x})$. We prove next that $(f(y_i), f(y_i'))$ belongs to the evaluation $\mathcal{A}_i(\mathcal{G}')$ of $\mathcal{A}_i$ over $\mathcal{G}'$, for each $1 \leq i \leq m$. This implies that $\nu(\bar{x}) \in \gamma(\mathcal{G}')$ and therefore that $\alpha'' \subseteq \gamma \subseteq \Gamma$.

As before, one would like to simulate the path $\pi_i = \pi_i^1 \cdots \pi_i^{k_i}$ in $\mathcal{G}^{\pm}$ by the path $\chi_i^1 \cdots \chi_i^{k_i}$ in $(\mathcal{G}')^{\pm}$, where χ_i^j is the semipath in the canonical database $\mathcal{G}'$ that is associated with the atom of α'' which is labeled E_i^j. The only problem is that some such atoms may have disappeared from α' and been replaced by atoms labeled with an NFA $\mathcal{A}_{\zeta_l}$ which accepts the semipath ζ_l in $\mathcal{G}$. Nevertheless, this could only have happened if one of the endpoints of a path of the form π_i^j is an internal node of $\mathcal{V}$. Let n and n' be external nodes in $\mathcal{V}$ such that n is the parent of n', and let ζ be a semipath in $\mathcal{G}$ from n to n'. Let t and t' be internal nodes of ζ (t is closer to n than t') and suppose that π_i^j goes from t to t'. (All other cases are analogous, i.e., when π_i^j goes from t' to t, when $t = t'$, and when t or t' is an external node). Furthermore, let η be the subpath of ζ that goes from t to t'. We now explain how to simulate π_i^j by a path from $\nu(z_t)$ to $\nu(z_{t'})$ in $(\mathcal{G}')^{\pm}$.

By construction, we have that α'' contains the atom $\mathcal{A}_\eta(z_t, z_{t'})$. Let $\tau = (\tau_{it}, \tau_{ti}, \tau_{ii}, \tau_{tt})$ be the Γ-type such that $\mathcal{A}_\eta = \mathcal{A}_\tau$. Then we have that $(\mathcal{A}_i, q_{c(j)}, q_{d(j)})$ is in τ_{it}. Let κ be the semipath from $\nu(z_t)$ to $\nu(z_{t'})$ in $\mathcal{G}'$ that is naturally associated with the atom $\mathcal{A}_\eta(z_t, z_{t'})$. By definition, the first coordinate of the Γ-type of $\mathsf{label}(\kappa)$ contains $(\mathcal{A}_i, q_{c(j)}, q_{d(j)})$. Therefore, there is a word u such that $\mathcal{A}_i$ has a run over u from $q_{c(j)}$ to $q_{d(j)}$ and u can be folded into $\mathsf{label}(\kappa)$ from the initial to the final position. It follows that there is a path ϱ_i^j in $(\mathcal{G}')^{\pm}$ from $\nu(z_t)$ to $\nu(z_{t'})$ that can be read in $\mathcal{A}_i$ from $q_{c(j)}$ to $q_{d(j)}$. This is the path that simulates π_i^j. From this reasoning it is easy to conclude that there must be a path in $(\mathcal{G}')^{\pm}$ from $\nu(z_{h(y_i)})$ to

$\nu(z_{h(y_i')})$ whose label is accepted by $\mathcal{A}_i$, for each $1 \leq i \leq m$. Therefore, $(f(y_i), f(y_i')) \in \mathcal{A}_i(\mathcal{G}')$ for each $1 \leq i \leq m$, which finishes the proof that α'' satisfies condition (2) in the statement of the lemma.

Now we define α. The query α is obtained from α'' by simultaneously replacing all atoms in $\mathcal{I}_b$ with the atom $\mathcal{A}_{\zeta_0} \cdot \mathcal{A}_{\zeta_1} \cdots \mathcal{A}_{\zeta_p}(z_{n_0}, z_{n_{p+1}})$, for all bad paths $b = z_{n_0} \cdots z_{n_{p+1}}$ in $\mathcal{U}(\alpha'')$, where $\mathcal{A}_{\zeta_0} \cdot \mathcal{A}_{\zeta_1} \cdots \mathcal{A}_{\zeta_p}$ accepts the concatenation of the languages accepted by the NFAs $\mathcal{A}_{\zeta_0}, \mathcal{A}_{\zeta_1}, \ldots, \mathcal{A}_{\zeta_p}$. Notice that this does not remove any free variables from α'', as they are external nodes of $\mathcal{U}(\alpha'')$. By construction, $\mathcal{U}(\alpha)$ does not contain bad paths, and thus it is acyclic. Furthermore, α contains fewer atoms than α'', which implies that condition (3) still holds. Clearly α is equivalent to α'', and therefore α also satisfies condition (2).

We are left to define the set $\mathcal{R}_\Gamma$. For an NFA $\mathcal{A}$ and states q, q' in $\mathcal{A}$, let $\mathcal{A}(q, q')$ be the NFA obtained from $\mathcal{A}$ by setting q as the initial state and q' as the only final state. Let $\mathcal{B}_\Gamma = \{\mathcal{A}(q, q') \mid \mathcal{A} \text{ appears in } \Gamma \text{ and } q, q' \text{ are states of } \mathcal{A}\}$. Furthermore, let $\mathcal{C}_\Gamma$ be the set of all NFAs of the form $\mathcal{A}_\tau$, where τ is a Γ-type (according to Lemma 3.24), and $\mathcal{D}_\Gamma$ be the set of all NFAs that accept the concatenation of the languages accepted by at most p NFAs in $\mathcal{C}_\Gamma$, where $p \leq |\Gamma| + 1$. Then we define $\mathcal{R}_\Gamma = \mathcal{B}_\Gamma \cup \mathcal{D}_\Gamma$. It is easy to see that $\mathcal{R}_\Gamma$ can be constructed from Γ in single exponential time. Moreover, it is clear that all 2RPQs mentioned in α belong to $\mathcal{R}_\Gamma$. Therefore conditions (1) and (4) are satisfied, which concludes the proof of the lemma. □

Construction of Γ_{app}

We now present the proof of Lemma 3.22. That is, we provide an EXPSPACE algorithm that, given a UC2RPQ Γ, constructs an acyclic UC2RPQ Γ_{app} such that:

1. $\Gamma_{\text{app}} \subseteq \Gamma$.
2. For every acyclic UC2RPQ Γ' such that $\Gamma' \subseteq \Gamma$ it is the case that $\Gamma' \subseteq \Gamma_{\text{app}}$.
3. The number of atoms and variables in each disjunct of Γ_{app} is at most polynomial in $|\Gamma|$.
4. The number of disjuncts and the size of each 2RPQ appearing in Γ_{app} is at most exponential in $|\Gamma|$.

Recall that Lemma 3.26 tells us that if $\mathcal{G}$ is a pseudo-acyclic graph database, $\bar{n}$ is a tuple of external nodes in $\mathcal{G}$ and $\Gamma(\bar{x})$ is a UC2RPQ with $|\bar{n}| = |\bar{x}|$ such that $\bar{n} \in \Gamma(\mathcal{G})$, then there is an acyclic C2RPQ $\alpha(\bar{x})$ and a finite set $\mathcal{R}_\Gamma$ of 2RPQs over Σ such that:

1. $\mathcal{R}_\Gamma$ can be constructed from Γ in time $2^{|\Gamma|^d}$, for a constant $d \geq 1$.
2. $\bar{n} \in \alpha(\mathcal{G})$ and $\alpha \subseteq \Gamma$.
3. The number of atoms in α is at most $|\Gamma|^c$, for a constant $c \geq 1$.
4. Each 2RPQ mentioned in α belongs to $\mathcal{R}_\Gamma$.

Given a UC2RPQ $\Gamma(\bar{x})$ over Σ our algorithm proceed as follows:

a. We construct the finite set $\mathcal{R}_\Gamma$ of 2RPQs over Σ from Lemma 3.26.
b. We iterate through every acyclic C2RPQ $\alpha'(\bar{x})$ with at most $|\Gamma|^c$ atoms, all of them labeled with 2RPQs in $\mathcal{R}_\Gamma$. If the C2RPQ α' is contained in Γ, then we add it as a disjunct to the output Γ_{app}.

First, observe that our algorithm is actually in ExpSpace. In fact, step (a) can be carried out in exponential time. Furthermore, the size of each 2RPQ in $\mathcal{R}_\Gamma$ is at most $2^{|\Gamma|^d}$. This implies that the size of every acyclic C2RPQ α' that we need to consider in the loop of step (b) is bounded by $|\Gamma|^c \cdot 2^{|\Gamma|^d}$, i.e., it is exponentially bounded by $|\Gamma|$. We can thus iterate over all such queries using no more than exponential space. Furthermore, from item (1) of Proposition 3.12, it follows that checking whether $\alpha' \subseteq \Gamma$ can be done using space which is at most exponential in $|\Gamma|$ and $\mathsf{maxvar}(\alpha')$. But $\mathsf{maxvar}(\alpha')$ is at most $2|\Gamma|^c$, and thus checking $\alpha' \subseteq \Gamma$ can be carried out in exponential space in $|\Gamma|$. Therefore, the whole procedure can be performed in ExpSpace.

Notice that Γ_{app} is nonempty. Indeed, assume that $x_1, \ldots, x_n$ are the free variables of Γ, i.e., those in $\bar{x}$. We assume without loss of generality that $\mathcal{R}_\Gamma$ contains the 2RPQs that define the empty word ε and the symbol a, for each $a \in \Sigma$. (If not, we simply extend $\mathcal{R}_\Gamma$ with those 2RPQs. The resulting set continues to satisfy all the desired conditions for $\mathcal{R}_\Gamma$). We also assume that the codification of Γ has size at least $n + |\Sigma|$. In particular, we have that $n + |\Sigma| \leq |\Gamma|^c$. Consider now the C2RPQ α^* defined by

$$\alpha^*(x_1, \ldots, x_n) \leftarrow \varepsilon(x_1, x_2), \varepsilon(x_2, x_3), \ldots, \varepsilon(x_{n-1}, x_n), a_1(x_1, x_1), \ldots, a_p(x_1, x_1),$$

where $\Sigma = \{a_1, \ldots, a_p\}$. Clearly, α^* is acyclic and its number of atoms is at most $n + |\Sigma| \leq |\Gamma|^c$. By the previous observations, it follows that α^* is one of the C2RPQs visited in step (b) of the procedure. Moreover, it is easy to verify that $\alpha^* \subseteq \Gamma$. Indeed, the only canonical database $\mathcal{G}$ of α^* consists of a single node u (which represents all the free variables $x_1, \ldots, x_n$) and a self-loop on u labeled a, for each $a \in \Sigma$. It is easy to see that $\bar{x} \in \Gamma(\mathcal{G})$. This is because if $\gamma(\bar{x})$ is an arbitrary C2RPQ in Γ, then the mapping h that sends every variable y of γ to u is a homomorphism which satisfies $h(\bar{x}) = \bar{x}$. We conclude that α^* is a disjunct of Γ_{app} and therefore, that Γ_{app} is not empty.

We now prove that Γ_{app} satisfies conditions (1), (2), (3) and (4) in the statement of Lemma 3.22. Conditions (3) and (4) are trivially satisfied by construction. For condition (1), we have by definition that each disjunct α' of Γ_{app} is contained in Γ, from which we conclude that $\Gamma_{\text{app}} \subseteq \Gamma$.

For condition (2), let $\Gamma'(\bar{x})$ be an acyclic UC2RPQ such that $\Gamma' \subseteq \Gamma$. We need to show that $\Gamma' \subseteq \Gamma_{\text{app}}$. Let $\mathcal{G}$ be a canonical database of Γ' with associated mapping ν. By Propositon 3.16, it suffices to show that $\nu(\bar{x}) \in \Gamma_{\text{app}}(\mathcal{G})$. Since Γ' is acyclic, we have from Proposition 3.23 that $\mathcal{G}$ is pseudo-acyclic. We also have that the tuple $\nu(\bar{x})$ only contains external nodes of $\mathcal{G}$. Moreover, since $\Gamma' \subseteq \Gamma$ it is the case that $\nu(\bar{x}) \in \Gamma(\mathcal{G})$. Therefore we can apply Lemma 3.26 to UC2RPQ Γ, graph database $\mathcal{G}$ and tuple of nodes $\nu(\bar{x})$. This ensures the existence of an acyclic C2RPQ $\alpha(\bar{x})$ satisfying conditions (1)-(4) in the statement of Lemma 3.26. Conditions (3) and (4) ensure that α is visited by our algorithm in step (b). On the other hand, the second statement in condition (2) ensures that $\alpha \subseteq \Gamma$, from which we have that α is a disjunct of Γ_{app}. Finally, the first statement of condition (2) tells us that $\nu(\bar{x}) \in \alpha(\mathcal{G})$, and therefore, $\nu(\bar{x}) \in \Gamma_{\text{app}}(\mathcal{G})$. This completes the proof of Lemma 3.22.

3.5 Semantically acyclic UC2RPQs: Evaluation and verification

We finish this chapter by studying the notion of semantic acyclicity in the context of graph databases and UC2RPQs. We are interested in understanding the complexity of evaluating semantically acyclic UC2RPQs. In particular, we want to know whether this problem is fixed-parameter tractable. Our main result in this chapter gives a positive answer to this question. Our proof is based on the approximation techniques developed in Section 3.4.

We conclude by studying the problem of verifying whether a UC2RPQ is semantically acyclic. Combining our techniques from Section 3.3 and 3.4 regarding containment algorithms for acyclic UC2RPQs and acyclic approximations, respectively, we obtain an EXPSPACE algorithm for this problem. We also provide a matching lower bound. Before presenting our results, we start by defining the terminology and providing some basic insights about the nature of semantic acyclicity for UC2RPQs.

3.5.1 Basic terminology and insights

We start by formally defining the notion of semantically acyclic UC2RPQs:

Definition 3.27 (Semantic acyclicity for UC2RPQs) *A UC2RPQ* Γ *is* semantically acyclic *if there exists an acyclic UC2RPQ* Γ' equivalent *to* Γ*, which is denoted by* $\Gamma \equiv \Gamma'$*. The latter means that* $\Gamma(\mathcal{G}) = \Gamma'(\mathcal{G})$*, for each graph database* $\mathcal{G}$*.*

Since acyclicity of C2RPQs is defined in terms of the acyclicity of its underlying CQ, one may be tempted to think that the two notions coincide. Clearly, if the underlying CQ of a C2RPQ γ is semantically acyclic then γ is also semantically acyclic. The following example shows that the opposite does not hold:

Example 3.28 Consider again the non-acyclic CRPQ $\gamma'' \leftarrow \mathcal{A}_1(x,y), \mathcal{A}_2(y,z), \mathcal{A}_3(z,x)$ in Example 3.11. It is not hard to prove that γ'' is equivalent to the acyclic CRPQ $\alpha \leftarrow \mathcal{A}_1 \cdot \mathcal{A}_2 \cdot \mathcal{A}_3(x,x)$, and thus, it is semantically acyclic. On the other hand, the underlying CQ of γ'' is $\theta \leftarrow T_1(x,y), T_2(y,z), T_3(z,x)$, which is not semantically acyclic.

Intuitively, the query γ'' is semantically acyclic because it can be "simplified" by concatenating the regular languages that label its atoms. A more interesting example is given by the Boolean CRPQ γ_{sa} over alphabet $\Sigma = \{a, \$_1, \$_2, \$_3\}$ shown in Figure 3.2 (Dots represent variables and arrows represent labeled atoms).

It is easy to see that the underlying CQ of γ_{sa} is not semantically acyclic. On the other hand, it can be proved that γ_{sa} is equivalent to the acyclic CRPQ shown in Figure 3.3. In this case, semantic acyclicity is obtained by the way in which the regular languages that label the atoms of γ_{sa} interact with each other. □

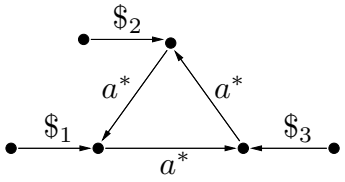

Figure 3.2: The CRPQ γ_{sa}

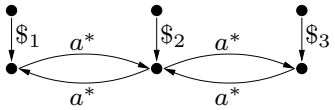

Figure 3.3: The acyclic CRPQ that is equivalent to γ_{sa}

The previous example shows that the notion of semantic acyclicity of C2RPQs is richer than the notion of semantic acyclicity of its underlying CQs, as many queries fall in the former category but not in the latter. Not only that, the first notion is also theoretically more challenging: While the same techniques used in Section 3.1 can be applied to prove that the evaluation problem is tractable for UC2RPQs whose underlying UCQ is semantically acyclic, it is by no means clear whether the same is true for the class of semantically acyclic UC2RPQs. We delve into this issue below.

As mentioned before, the CSP techniques used in Section 3.1 to prove that the evaluation of semantically acyclic UCQs is tractable do not yield answers to our questions about semantically acyclic UC2RPQs. The results in Section 3.4 about approximations help us to prove, on the other hand, that the problem is fixed-parameter tractable (Theorem 3.30), which was not known to date.

Before finishing the section we explore the limits of the notion of semantic acyclicity. The next example shows a simple CRPQ over graph databases that is not equivalent to any acyclic UC2RPQ.

Example 3.29 Let $\Sigma = \{a, \$_1, \$_2, \$_3\}$ be a finite alphabet and consider the Boolean CRPQ γ_{na} over Σ that is graphically depicted in Figure 3.4.

Notice that the underlying CQ of γ_{na} coincides with that of the semantically acyclic CRPQ γ_{sa} from Figure 3.2. However, a simple case-by-case analysis shows that γ_{na} is not

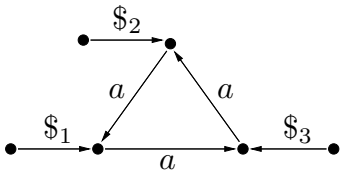

Figure 3.4: The CRPQ γ_{na} from Example 3.29

semantically acyclic. The reason is that γ_{na} forbids the interaction between the different RPQs that label its atoms by replacing each RPQ of the form a^* in γ_{sa} with a. □

3.5.2 Evaluation of semantically acyclic UC2RPQs

With the help of Corollary 3.20 we can provide an answer to our main question regarding semantically acyclic UC2RPQs: Its evaluation is fixed-parameter tractable.

Theorem 3.30 *The problem of checking whether $\bar{n} \in \Gamma(\mathcal{G})$, for a given graph database $\mathcal{G}$, semantically acyclic UC2RPQ Γ, and tuple $\bar{n}$ of nodes in $\mathcal{G}$, is fixed-parameter tractable.*

Proof: Using the algorithm in Corollary 3.20 it is possible to compute in EXPSPACE (i.e., in double-exponential time) an acyclic approximation of Γ, that is, an acyclic UC2RPQ Γ' such that $\Gamma' \subseteq \Gamma$, and $\Gamma'' \subseteq \Gamma'$, whenever Γ'' is an acyclic UC2RPQ contained in Γ. Corollary 3.20 also tell us that $|\Gamma'|$ is at most exponential in $|\Gamma|$. Observe that since Γ is semantically acyclic then Γ' must be equivalent to Γ. Thus, we have that $\bar{n} \in \Gamma(\mathcal{G})$ if and only if $\bar{n} \in \Gamma'(\mathcal{G})$. From Theorem 3.8 the latter can be checked in time $O(|\Gamma'|^2 \cdot |\mathcal{G}|^2)$, and hence in time $O(2^{|\Gamma|^c} \cdot |\mathcal{G}|^2)$, for a constant $c \geq 1$. Thus, semantically acyclic UC2RPQs can be evaluated in time $O(2^{2^{|\Gamma|^d}} + 2^{|\Gamma|^c} \cdot |\mathcal{G}|^2)$, for constant $c, d \geq 1$. □

3.5.3 Verification of semantic acyclicity

We conclude by considering the following question: Is the notion of semantic acyclicity for UC2RPQs decidable? In this section we show that this is indeed the case and provide matching upper and lower bounds for its computational cost.

We start by proving that the notion of semantic acyclicity for UC2RPQs is decidable, and provide an EXPSPACE upper bound for the problem. The algorithm also yields an equivalent acyclic UC2RPQ Γ' of exponential size for a semantically acyclic UC2RPQ Γ.

Theorem 3.31 *There exists an* EXPSPACE *algorithm that on input a UC2RPQ Γ does the following:*

1. *It checks whether Γ is semantically acyclic.*
2. *If the latter holds, it outputs an acyclic UC2RPQ Γ' of single-exponential size such that $\Gamma \equiv \Gamma'$.*

Proof: The algorithm in Lemma 3.22 computes on input Γ an acyclic UC2RPQ Γ_{app} such that Γ_{app} is the maximum among all acyclic UC2RPQs that are contained in Γ. It follows that Γ is semantically acyclic if and only if $\Gamma \subseteq \Gamma_{\text{app}}$. Thus, in order to check semantic acyclicity of Γ we can compute Γ_{app}, which can be done in EXPSPACE, and then check whether $\Gamma \subseteq \Gamma_{\text{app}}$. It follows from item (1) of Proposition 3.12 that the latter can be done

using space exponential in $\mathsf{maxvar}(\Gamma)$ and $|\Gamma_{\text{app}}|$, and hence double-exponential in $|\Gamma|$ (since Γ_{app} can be of exponential size in $|\Gamma|$). This provides us with an easy 2Expspace procedure for checking semantic acyclicity of UC2RPQs.

To obtain an Expspace procedure we exploit Theorem 3.13 from Section 3.3. Observe first that the width $w(\Gamma_{\text{app}})$ of Γ_{app} is at most the maximum number of atoms over its disjuncts, which by Lemma 3.22 item (3) is at most polynomial in $|\Gamma|$. Since Γ_{app} is acyclic, Theorem 3.13 tells us that the problem of checking $\Gamma \subseteq \Gamma_{\text{app}}$ can be decided using $O((|\Gamma|+|\Gamma_{\text{app}}|)^{C\cdot w(\Gamma_{\text{app}})})$ space, for some constant $C \geq 1$. Hence, from the fact that $w(\Gamma_{\text{app}})$ is polynomially bounded by $|\Gamma|$ we obtain that checking $\Gamma \subseteq \Gamma_{\text{app}}$ can be solved in Expspace. This gives an Expspace algorithm for checking whether Γ is semantically acyclic. For the second part of the theorem, we output the query Γ_{app} if $\Gamma \subseteq \Gamma_{\text{app}}$; otherwise, we reject the input. □

We now provide a lower bound for the problem that shows that checking semantic acyclicity of (U)C(2)RPQs is considerably harder than for UCQs:

Proposition 3.32 *It is* Expspace*-hard to check whether a UC2RPQ Γ is semantically acyclic. The problem remains* Expspace*-hard even if the input is restricted to Boolean CRPQs.*

Proof: First we claim that checking containment of γ_1 in γ_2, where γ_1 and γ_2 are UC2RPQs, is Expspace-hard even when γ_1 is a Boolean acyclic CRPQ, γ_2 is a Boolean CRPQ with $\mathcal{U}(\gamma_2)$ connected and γ_2 is not semantically acyclic.

Indeed, it follows from [33] (also stated in item (2) of Proposition 3.12) that containment is Expspace-hard, even for γ_1 and γ_2 acyclic CRPQs. Moreover, the reduction in [33] yields CRPQs γ_1 and γ_2 of the following form:

$$\gamma_1(x_1, x_2) \leftarrow E(x_1, x_2)$$
$$\gamma_2(x_1, x_2) \leftarrow E_1(x_1, y_1), F_0(y_1, y_2), \ldots, F_n(y_1, y_2), E_1(y_2, x_2),$$

where $E, E_1, F_0, \ldots, F_n$ are RPQs over an alphabet Δ. Let $\Sigma = \{a, \$_1, \$_2, \$_3\}$ be an alphabet disjoint from Δ and $\#_1, \#_2$ fresh symbols not in $\Delta \cup \Sigma$. We then construct Boolean CRPQs γ_1' and γ_2' from γ_1, γ_2 as follows:

$$\gamma_1' \leftarrow E(x_1, x_2), \#_1(x_1, x_1), \#_2(x_2, x_2), a(x_2, x_2), \$_1(x_2, x_2), \$_2(x_2, x_2), \$_3(x_2, x_2)$$
$$\gamma_2' \leftarrow \#_1(x_1, x_1), \#_2(x_2, x_2), E_1(x_1, y_1), F_0(y_1, y_2), \ldots, F_n(y_1, y_2), E_1(y_2, x_2),$$
$$a(x_2, z_2), a(z_2, z_3), a(z_3, x_2), \$_1(w_1, x_2), \$_2(w_2, z_2), \$_3(w_3, z_3)$$

Intuitively, γ_1' is the Boolean version of γ_1, where we have marked the free variables x_1 and x_2 of γ_1 with special symbols $\#_1$ and $\#_2$, respectively, and appended a loop labeled b to x_2, for each $b \in \Sigma$. Similarly, γ_2' is the Boolean version of γ_2, where we have again marked the free variables x_1 and x_2 of γ_2 with special symbols $\#_1$ and $\#_2$, respectively, and appended to x_2 a copy of the (non-semantically acyclic) query γ_{na} from Example 3.29.

It is straightforward to show that $\gamma_1 \subseteq \gamma_2$ if and only if $\gamma_1' \subseteq \gamma_2'$. Note also that γ_1' is a Boolean acyclic CRPQ and γ_2' is a Boolean CRPQ such that $\mathcal{U}(\gamma_2')$ is connected. Moreover, since γ_{na} is not semantically acyclic, it is easy to show that γ_2' is not semantically acyclic either. Thus containment is EXPSPACE-hard even for these kind of queries. Next we reduce this EXPSPACE-hard restriction of the containment problem to our problem of checking whether a query is semantically acyclic.

Let γ_1 and γ_2 be Boolean CRPQs such that γ_1 is acyclic, $\mathcal{U}(\gamma_2)$ is connected and γ_2 is not semantically acyclic. We assume without loss of generality that the variable set of γ_1 and γ_2 are disjoint, and denote by $\gamma_1 \wedge \gamma_2$, the Boolean CRPQ whose set of atoms is the union of the atoms of γ_1 and γ_2. We claim that γ_1 is contained in γ_2 if and only if $\gamma_1 \wedge \gamma_2$ is semantically acyclic. Assume first that γ_1 is contained in γ_2. Then $\gamma_1 \wedge \gamma_2$ is equivalent to γ_1, which is acyclic, and thus $\gamma_1 \wedge \gamma_2$ is semantically acyclic.

On the other hand, assume that $\gamma_1 \wedge \gamma_2 \equiv \alpha$, where $\alpha = \{\alpha_1, \ldots, \alpha_m\}$ and each α_i is an acyclic C2RPQ. Since $\gamma_1 \wedge \gamma_2 \subseteq \gamma_2$, it follows that $\alpha \subseteq \gamma_2$. Also, since $\alpha_i \subseteq \alpha$ it follows that $\alpha_i \subseteq \gamma_2$, for each $1 \leq i \leq m$. Let $\alpha_i^1, \ldots, \alpha_i^{k_i}$ be the C2RPQs associated with each connected component of $\mathcal{U}(\alpha_i)$. We show that for each $1 \leq i \leq m$, there exists $1 \leq j_i \leq k_i$ such that $\alpha_i^{j_i} \subseteq \gamma_2$. Assume to the contrary. Then, by Proposition 3.16, there exists for each $1 \leq j \leq k_i$, a canonical graph database $\mathcal{G}_j$ for α_i^j such that $\gamma_2(\mathcal{G}_j) = \texttt{false}$. Note that since $\mathcal{U}(\alpha_i^j)$ is connected, then $\mathcal{G}_j$ also is. Consider the disjoint union $\mathcal{G}$ of $\mathcal{G}_1, \ldots \mathcal{G}_{k_i}$. Clearly this is a canonical database for α_i. Since $\alpha_i \subseteq \gamma_2$, it follows that $\gamma_2(\mathcal{G}) = \texttt{true}$. But $\mathcal{U}(\gamma_2)$ is connected, which implies that $\gamma_2(\mathcal{G}_j) = \texttt{true}$, for some $1 \leq j \leq k_i$. This is a contradiction.

Consider the acyclic UC2RPQ $\alpha' = \{\alpha_1^{j_1}, \ldots, \alpha_m^{j_m}\}$. Notice that $\alpha \subseteq \alpha'$. By definition, we have that $\alpha' \subseteq \gamma_2$. Note also that since γ_2 is not semantically acyclic, it must be the case that $\gamma_2 \nsubseteq \alpha'$. Hence there is a canonical database $\mathcal{G}^*$ for γ_2 such that $\alpha'(\mathcal{G}^*) = \texttt{false}$. We now prove that $\gamma_1 \subseteq \alpha'$, which implies $\gamma_1 \subseteq \gamma_2$. Let $\mathcal{G}$ be a canonical database for γ_1. Consider the disjoint union $\mathcal{G}'$ of $\mathcal{G}$ and $\mathcal{G}^*$. Clearly $\mathcal{G}'$ is a canonical database for $\gamma_1 \wedge \gamma_2$. Since $\gamma_1 \wedge \gamma_2 \equiv \alpha \subseteq \alpha'$, it follows that $\alpha'(\mathcal{G}') = \texttt{true}$. Then $\alpha_i^{j_i}(\mathcal{G}') = \texttt{true}$, for some $1 \leq i \leq m$. Since $\mathcal{U}(\alpha_i^{j_i})$ is connected, it follows that either $\alpha_i^{j_i}(\mathcal{G}) = \texttt{true}$ or $\alpha_i^{j_i}(\mathcal{G}^*) = \texttt{true}$. But $\alpha'(\mathcal{G}^*) = \texttt{false}$, and thus $\alpha_i^{j_i}(\mathcal{G}^*) = \texttt{false}$. We conclude that $\alpha_i^{j_i}(\mathcal{G}) = \texttt{true}$, and thus $\alpha'(\mathcal{G}) = \texttt{true}$. By Proposition 3.16, we conclude that $\gamma_1 \subseteq \alpha'$. This proves the proposition. □

Chapter 4

UC2RPQs of bounded treewidth modulo equivalence

In this chapter, we study the UC2RPQs of treewidth at most k *modulo equivalence*, that is, the UC2RPQs that are equivalent to a UC2RPQ of treewidth at most k. While Chapter 3 only focused on the case $k = 1$, here we consider the general case when $k \geq 1$. Our main result in this chapter (and in this first part of the thesis) states that the evaluation problem for UC2RPQs of treewidth at most k modulo equivalence is fixed-parameter tractable, for each fixed $k \geq 1$.

For the sake of completeness, we start by reviewing known results about UCQs. We say that a class $\mathcal{C}$ of UCQs has *bounded treewidth modulo equivalence* if there is a constant $k \geq 1$ such that each UCQ in $\mathcal{C}$ has treewidth at most k modulo equivalence. The results we review will help us to understand the intimate connection between bounded treewidth modulo equivalence and tractability of UCQs, which has motivated our study of this notion for UC2RPQs. This connection states that the following are equivalent (under some widely believed assumptions) for a class $\mathcal{C}$ of UCQs over graph databases: (i) Evaluation for $\mathcal{C}$ is tractable, (ii) Evaluation for $\mathcal{C}$ is fixed-parameter tractable, and (iii) $\mathcal{C}$ has bounded treewidth modulo equivalence.

Then we turn to the case of UC2RPQs. The techniques used to show that bounded treewidth modulo equivalence characterizes tractable and fixed-parameter tractable evaluation of UCQs come again from the area of CSPs, and thus cannot be applied directly to UC2RPQs. However, we show that our results developed in Chapter 3 for semantically acyclic UC2RPQs, which are based on automata and homomorphisms techniques, can be extended to the case of UC2RPQs of treewidth at most k modulo equivalence, for $k \geq 1$. In particular, we show that there is an Expspace algorithm that given a UC2RPQ Γ of treewidth at most k modulo equivalence, where $k \geq 1$ is fixed, computes an equivalent UC2RPQ Γ' of exponential size and treewidth at most $2k+1$. This result is a nontrivial extension of Lemma 3.22 from Section 3.4 and Theorem 3.31 from Section 3.5.

Now we can prove the main result of this chapter, namely, that evaluation of UC2RPQs of treewidth at most k modulo equivalence is fixed-parameter tractable, for each fixed $k \geq 1$.

Indeed, in order to evaluate such a UC2RPQ Γ, we can compute and evaluate the equivalent UC2RPQ Γ'. Since Γ' still has bounded treewidth, specifically, treewidth at most $2k+1$, it can be evaluated in polynomial time, and thus the overall running time is fixed-parameter tractable. It is important to note that, unlike Lemma 3.22 and Theorem 3.31 from Chapter 3, there is a gap between the treewidth of Γ, which is at most k modulo equivalence, and the treewidth of Γ', which is at most $2k+1$. As a consequence, the results in this chapter do not imply results about approximations and verification of UC2RPQs of treewidth at most k modulo equivalence, when $k > 1$.

Organization We review in Section 4.1 known connections between bounded treewidth and UCQ evaluation. Then in Section 4.2, we introduce the notion of treewidth for UC2RPQs and show our main result that evaluating UC2RPQs of bounded treewidth modulo equivalence is fixed-parameter tractable.

4.1 Known results for UCQs

Besides acyclic UCQs, introduced in Section 3.1, there is another prominent family of tractable restrictions of UCQs, namely, the classes of UCQs of bounded treewidth. These classes were first studied by Chekuri and Rajaraman [45], who showed that evaluation for these can be done in polynomial time. Since then, a large body of research has focused on classes of database queries of bounded treewidth and extensions thereof [45, 59, 80, 81, 85, 89, 86, 107].

The concept of treewidth comes from the area of graph theory, where it has played an important role in many important results such as the famous and deep graph minor theorem [61]. Intuitively, the treewidth measures how close is a graph from being acyclic. In particular, a graph has treewidth 1 if and only if it is acyclic.

Definition 4.1 (Treewidth) *A* tree decomposition *of a graph $G = (V, E)$ is a pair (T, λ), where T is a tree and λ is a mapping from the nodes of T to 2^V, that satisfies the following:*

1. *For each $v \in V$ the set $\{t \in T \mid v \in \lambda(t)\}$ is a connected subset of T.*
2. *Each edge in E is contained in one of the sets $\lambda(t)$, for $t \in T$.*

The width *of the tree decomposition (T, λ) is defined as $\max\{|\lambda(t)| \mid t \in T\} - 1$. The* treewidth *of a graph G is the minimum width over all its possible tree decompositions.*

Figure 4.1 shows a tree decomposition of width 2 of the graph C_5, that is, the cycle of length 5. In fact, it can be easily shown that the treewidth of C_n is 2, for each $n \geq 3$.

On the other hand, two examples of graphs with high treewidth are (i) the clique K_n, with $n \geq 2$, whose treewidth is $n-1$ and (ii) the $n \times n$ *grid* whose vertex set is $\{1, \ldots, n\} \times \{1, \ldots, n\}$ and its edge set is $\{\{(i, i'), (j, j')\} \mid |i - i'| + |j - j'| = 1\}$. The treewidth of the $n \times n$ grid is precisely n. For instance, Figure 4.2 depicts the 3×3 grid with a tree decomposition of width 3.

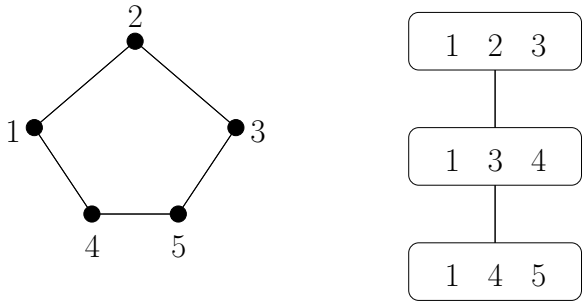

Figure 4.1: The cycle C_5 and a tree decomposition of width 2

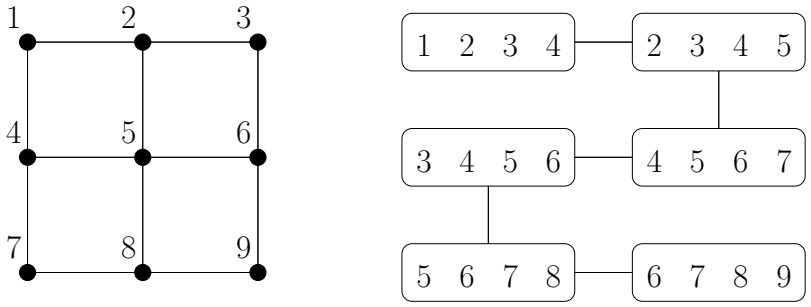

Figure 4.2: The grid 3×3 and a tree decomposition of width 3

The *Gaifman graph* $\mathcal{G}(\theta)$ of a CQ θ is the graph whose vertices are the variables of θ and there is an edge $\{x, y\}$ in $\mathcal{G}(\theta)$ whenever x and y are distinct variables and they appear together in some atom of θ. The treewidth of a CQ θ is the treewidth of $\mathcal{G}(\theta)$. We define the treewidth of a UCQ Θ to be the maximum treewidth over all disjuncts of Θ. For a positive integer $k \geq 1$, we define $\mathsf{TW}_{\mathtt{ucq}}(k)$ to be the class of UCQs with treewidth at most k. It follows from the work of Chekuri and Rajaraman [45] (see also [59]) that UCQs of bounded treewidth are tractable in terms of evaluation:

Proposition 4.2 **[45]** *Let $k \geq 1$ be a positive integer. The evaluation problem for UCQs in $\mathsf{TW}_{\mathtt{ucq}}(k)$ can be solved in polynomial time.*

Observe that Proposition 4.2 does not follow directly from results about acyclic UCQs in Section 3.1, as the notion of acyclicity and bounded treewidth are incomparable. Indeed, for $n \geq 3$, let θ_n be a Boolean CQ over directed graphs, that is, over the schema that contains a single binary predicate, such that the Gaifman graph of θ_n is the cycle C_n of length n. Then as we mentioned before, the class $\{\theta_n \mid n \geq 3\}$ is of bounded treewidth, since each θ_n is of treewidth at most 2. However, none of the CQs in this class is acyclic.

On the other hand, consider a Boolean CQ θ'_n, for $n \geq 3$, whose set of atoms is $\mathcal{K}_n \cup \{R_n(x_1, \ldots, x_n)\}$, where the Gaifman graph of $\mathcal{K}_n$ is the clique on $\{x_1, \ldots, x_n\}$. Then the class $\{\theta'_n \mid n \geq 3\}$ is of unbounded treewidth as the treewidth of θ'_n is $n - 1$, but each CQ in this class is acyclic. Nevertheless, when each predicate in the schema is binary, as in the case of graph databases, a UCQ is acyclic if and only if it has treewidth 1. Therefore, over

graph databases, the notion of bounded treewidth actually generalizes acyclicity as defined in Section 3.1.

Interestingly, Proposition 4.2 can be extended from bounded treewidth to bounded treewidth modulo equivalence [59]. Formally, let $k \geq 1$ be a positive integer. We say that a UCQ has treewidth at most k *modulo equivalence* if it is equivalent to a UCQ of treewidth at most k. We denote by $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$ the class of UCQs of treewidth at most k modulo equivalence. The following result follows from [59]:

Proposition 4.3 [59] *Let $k \geq 1$ be a positive integer. The evaluation problem for UCQs in $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$ can be solved in polynomial time.*

The algorithm behind Proposition 4.3 exploits a well-known connection between the existential $(k+1)$-pebble game [106] and CQs of treewidth at most k. More precisely, suppose that Θ is a UCQ in $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$, $\mathcal{D}$ is a database and $\bar{a}$ is a tuple. It follows from [59], that $\bar{a} \in \Theta(\mathcal{D})$ if and only if duplicator has a winning strategy in the existential $(k+1)$-pebble game on $(\mathcal{D}^\theta, \bar{x})$ and $(\mathcal{D}, \bar{a})$, for some disjunct $\theta \in \Theta$. Recall that $\mathcal{D}^\theta$ is the canonical database of θ (see Section 2.1) and $(\mathcal{D}, \bar{a})$ denotes the database $\mathcal{D}$ extended with constants $\bar{a}$ (similarly for $(\mathcal{D}^\theta, \bar{x})$; see beginning of the proof of Theorem 3.3 in Section 3.1.2). Since checking whether the duplicator can win the existential $(k+1)$-pebble game can be checked in polynomial time, for each fixed $k \geq 1$ [59], Proposition 4.3 follows.

Observe that the class $\mathsf{TW}_{\mathtt{ucq}}(k)$ is strictly contained in $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$. Indeed, consider the Boolean CQ $\theta_{n,n}$, for $n \geq 3$, that contains one variable x_u for each node u in the $n \times n$ grid and atoms $E(x_u, x_v)$ and $E(x_v, x_u)$, whenever $\{u, v\}$ is an edge in the $n \times n$ grid. By construction, the Gaifman graph of $\theta_{n,n}$ is the $n \times n$ grid and thus the treewidth of $\theta_{n,n}$ is n. However, it can be checked that $\theta_{n,n}$ is equivalent to the Boolean CQ $\theta^1 \leftarrow E(x,y), E(y,x)$ (this follows from the fact that the $n \times n$ grid is bipartite). Since the treewidth of θ^1 is 1, it follows that the family $\{\theta_{n,n} \mid n \geq 3\}$ is contained in $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(1))$, while it is not contained in $\mathsf{TW}_{\mathtt{ucq}}(1)$ (and actually it is not contained in $\mathsf{TW}_{\mathtt{ucq}}(k)$ for any $k \geq 1$).

Can we extend Proposition 4.3 even further? Remarkably, Grohe [86] showed that this is impossible (under some reasonable assumptions). Formally, we say that a class $\mathcal{C}$ of UCQs has *bounded treewidth modulo equivalence* if there is a positive integer $k \geq 1$ such that $\mathcal{C} \subseteq \mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$. Also, we define the *arity* of a UCQ Θ to be the maximum arity of a predicate appearing in Θ. Then we say that a class $\mathcal{C}$ of UCQs has *bounded arity* if there is a constant $r \geq 1$ such that the arity of any UCQ in $\mathcal{C}$ is at most r. The following result follows from [86]:

Theorem 4.4 [86] *Assume* $\mathrm{W}[1] \neq \mathrm{FPT}$. *Let $\mathcal{C}$ be a recursively enumerable class of UCQs of bounded arity. Then the following are equivalent:*

1. *The evaluation problem for UCQs in $\mathcal{C}$ is in polynomial time*
2. *The evaluation problem for UCQs in $\mathcal{C}$ is fixed-parameter tractable*
3. *$\mathcal{C}$ has bounded treewidth modulo equivalence*

The assumption $\mathrm{W}[1] \neq \mathrm{FPT}$ is a widely believed assumption from parameterized com-

plexity theory [75], analogous to the well-known assumption P $\neq$ NP from classical complexity theory. The hypothesis that $\mathcal{C}$ is recursively enumerable is a mild condition and it can actually be removed by strengthening the W[1] $\neq$ FPT assumption [86]. On the other hand, the condition that $\mathcal{C}$ is of bounded arity is necessary in the theorem. Indeed, there are several classes with unbounded arity that are not of bounded treewidth modulo equivalence but are still tractable (e.g., the acyclic UCQs and the UCQs of bounded hypertree width [160, 81]). For example, the class $\{\theta'_n \mid n \geq 1\}$ defined previously has unbounded treewidth modulo equivalence, but each θ'_n is acyclic and thus the class can be evaluated in polynomial time.

Observe that, in the context of graph databases, any UCQ has arity 2 by definition. Thus over graph databases, Theorem 4.4 give us a clean characterization of the classes of UCQs that are tractable and fixed-parameter tractable: they are precisely the classes of bounded treewidth modulo equivalence. This is stated in the following corollary, which also justifies our study of UC2RPQs of bounded treewidth modulo equivalence:

Corollary 4.5 *Assume* W[1] $\neq$ FPT. *Let $\mathcal{C}$ be a recursively enumerable class of UCQs over graph databases. Then the following are equivalent:*

1. *The evaluation problem for UCQs in $\mathcal{C}$ is in polynomial time*
2. *The evaluation problem for UCQs in $\mathcal{C}$ is fixed-parameter tractable*
3. *$\mathcal{C}$ has bounded treewidth modulo equivalence*

We conclude with a result that establishes an important difference between the classes $\mathsf{TW}_{\mathtt{ucq}}(k)$ and $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$: verifying membership into the former class it is known to be in polynomial time (when $k \geq 1$ is fixed) [10, 24], while it is intractable for the latter class:

Proposition 4.6 **[59]** *Let $k \geq 1$ be a positive integer. Then checking whether a UCQ belongs to $\mathcal{H}(\mathsf{TW}_{\mathtt{ucq}}(k))$ is* NP*-complete.*

4.2 UC2RPQs of bounded treewidth modulo equivalence: Evaluation is fixed-parameter tractable

The treewidth of a C2RPQ is the treewidth of its underlying CQ, as defined in Section 3.2. Equivalently, we can define the treewidth of a C2RPQ γ as the treewidth of its underlying graph $\mathcal{U}(\gamma)$ (see Section 3.2). The treewidth of a UC2RPQ is the maximum treewidth over all its disjuncts. For a positive integer $k \geq 1$, let $\mathsf{TW}_{\mathtt{uc2rpq}}(k)$ (or $\mathsf{TW}(k)$ for short) be the class of UC2RPQs of treewidth at most k. As expected, UC2RPQs of bounded treewidth can be evaluated efficiently:

Proposition 4.7 **[13]** *Let $k \geq 1$ be a positive integer. The evaluation problem for UC2RPQs in $\mathsf{TW}(k)$ can be solved in polynomial time.*

Below we introduce the notion of treewidth modulo equivalence for UC2RPQs:

Definition 4.8 (Treewidth modulo equivalence for UC2RPQs) *Let $k \geq 1$ be a positive integer. A UC2RPQ has treewidth at most k* modulo equivalence *if it is equivalent to a UC2RPQ of treewidth at most k. We denote by $\mathcal{H}(\mathsf{TW}(k))$ the class of UC2RPQs of treewidth at most k modulo equivalence.*

Observe that the class $\mathcal{H}(\mathsf{TW}(1))$ contains precisely the semantically acyclic UC2RPQs studied in depth in Section 3. Our main result states that the class $\mathcal{H}(\mathsf{TW}(k))$ is well-behaved in terms of evaluation:

Theorem 4.9 *Let $k \geq 1$ be a positive integer. Then the evaluation problem for UC2RPQs in $\mathcal{H}(\mathsf{TW}(k))$ is fixed-parameter tractable.*

We devote the rest of this chapter to prove Theorem 4.9. First, we show how Theorem 4.9 follows directly from a lemma that generalizes Theorem 3.31 from Section 3.5.3, to the case when $k > 1$:

Lemma 4.10 *Let $k > 1$ be a positive integer. There is an* Expspace *algorithm that on input a UC2RPQ Γ in $\mathcal{H}(\mathsf{TW}(k))$, computes a UC2RPQ Γ' such that (i) $\Gamma' \in \mathsf{TW}(2k+1)$, (ii) $\Gamma' \equiv \Gamma$, and (iii) $|\Gamma'|$ is exponential in $|\Gamma|$.*

Proof of Theorem 4.9: The case when $k = 1$ follows from Theorem 3.30 in Section 3.5.2, thus we assume $k > 1$. Let Γ be a UC2RPQ in $\mathcal{H}(\mathsf{TW}(k))$, $\mathcal{G}$ be a graph database and $\bar{n}$ be a tuple of nodes of $\mathcal{G}$. In order to check whether $\bar{n} \in \Gamma(\mathcal{G})$, we can compute Γ' with the algorithm in Lemma 4.10, and then check whether $\bar{n} \in \Gamma'(\mathcal{G})$. Computing Γ' can be done in Expspace, and thus it takes time at most $2^{2^{|\Gamma|^d}}$, for a constant $d \geq 1$. Since $\Gamma' \in \mathsf{TW}(2k+1)$, evaluating Γ' takes time $O(|\Gamma'|^c \cdot |\mathcal{G}|^c)$, for a constant $c \geq 1$. As $|\Gamma'|$ is at most exponential in $|\Gamma|$, this is $O(2^{|\Gamma|^{c'}} \cdot |\mathcal{G}|^c)$, for constants $c, c' \geq 1$. Thus the overall running time is $O(2^{2^{|\Gamma|^d}} + 2^{|\Gamma|^{c'}} \cdot |\mathcal{G}|^c)$, and hence, the algorithm is fixed-parameter tractable. □

Thus it suffices to show Lemma 4.10. Before doing this, we need to introduce some notation. The underlying graph of a graph database $\mathcal{G}$, denoted by $\mathcal{U}(\mathcal{G})$, is the graph whose vertex set is precisely the vertex set of $\mathcal{G}$ and where there is an edge $\{n, n'\}$ whenever $n \neq n'$ and (n, a, n') or (n', a, n) is an edge of $\mathcal{G}$, for some symbol a. We define the treewidth of a graph database $\mathcal{G}$ to be the treewidth of $\mathcal{U}(\mathcal{G})$. The following proposition is straightforward (recall the definition of canonical databases from Section 3.3.1):

Proposition 4.11 *Let $k > 1$ be a positive integer. Suppose Γ is a UC2RPQ in $\mathsf{TW}(k)$, that is, a UC2RPQ of treewidth at most k. Then any canonical database of Γ is also of treewidth at most k.*

Observe the difference between Proposition 4.11 and Proposition 3.23 from Section 3.4.2: the treewidth of the canonical databases of a UC2RPQ in $\mathsf{TW}(1)$ is not 1 (although they are still pseudo-acyclic), as in the case $k > 1$, where the treewidth does not change. The proof of Lemma 4.10 is based on a technical lemma analogous to Lemma 3.26:

Lemma 4.12 *Let $k > 1$ be a positive integer. Let $\mathcal{G}$ be a graph database of treewidth at most*

k and $\bar{n}$ a tuple of nodes in $\mathcal{G}$. Let $\Gamma(\bar{x})$ be a UC2RPQ with $|\bar{n}| = |\bar{x}|$ such that $\bar{n} \in \Gamma(\mathcal{G})$. There exists a C2RPQ $\alpha(\bar{x})$ with treewidth at most $2k+1$ and a finite set $\mathcal{R}_\Gamma$ of 2RPQs over Σ such that:

1. *$\mathcal{R}_\Gamma$ can be constructed from Γ in polynomial time.*
2. *$\bar{n} \in \alpha(\mathcal{G})$ and $\alpha \subseteq \Gamma$.*
3. *The number of atoms in α is at most $|\Gamma|^C$, for a constant $C \geq 1$.*
4. *Each 2RPQ mentioned in α belongs to $\mathcal{R}_\Gamma$.*

Before proving our technical lemma, we show how Lemma 4.10 follows from it.

Proof of Lemma 4.10: The idea behind the construction of Γ' is similar to that of Γ_{app} in Lemma 3.22. Given a UC2RPQ $\Gamma(\bar{x})$ over Σ our algorithm proceed as follows:

a. We construct the finite set $\mathcal{R}_\Gamma$ of 2RPQs over Σ from Lemma 4.12.
b. We iterate through every C2RPQ $\alpha'(\bar{x})$ of treewidth a most $2k+1$ with at most $|\Gamma|^C$ atoms, all of them labeled with 2RPQs in $\mathcal{R}_\Gamma$, where C is the constant in condition (3) of Lemma 4.12. If the C2RPQ α' is contained in Γ, then we add it as a disjunct to the output Γ'.

This algorithm is in Expspace. Indeed, step (a) can be done in polynomial time, and thus in Expspace. Also, note that the size of each 2RPQ in $\mathcal{R}_\Gamma$ is polynomial in $|\Gamma|$, and thus the size of every $\alpha'(\bar{x})$ visited in step (b) is also polynomial in $|\Gamma|$. From item (1) of Proposition 3.12, we know that checking whether $\alpha' \subseteq \Gamma$ can be done using exponential space in $|\alpha'| + |\Gamma|$, and thus using exponential space in $|\Gamma|$. Therefore, the whole procedure can be performed in Expspace.

As in the proof of Lemma 3.22, we have that Γ' is actually nonempty, since the C2RPQ α^* defined by

$$\alpha^*(x_1, \ldots, x_n) \leftarrow \varepsilon(x_1, x_2), \varepsilon(x_2, x_3), \ldots, \varepsilon(x_{n-1}, x_n), a_1(x_1, x_1), \ldots, a_p(x_1, x_1),$$

where $\Sigma = \{a_1, \ldots, a_p\}$, is always part of Γ'. Indeed, α^* is acyclic, and hence, of treewidth at most $2k+1$, the size of α^* is at most $|\Gamma|^C$, and α^* is always contained in Γ.

By construction, the treewidth of Γ' is at most $2k+1$, and hence, condition (i) is satisfied. Moreover, it is straightforward to check that $|\Gamma'|$ is at most exponential in $|\Gamma|$ and thus condition (iii) is also true. It remains to show condition (ii), that is, to show that Γ' is equivalent to Γ. By construction, Γ' is contained in Γ, thus it suffices to prove that $\Gamma \subseteq \Gamma'$.

By hypothesis, we know that $\Gamma \in \mathcal{H}(\mathsf{TW}(k))$, and thus, there is a UC2RPQ $\tilde{\Gamma} \in \mathsf{TW}(k)$ such that $\Gamma \equiv \tilde{\Gamma}$. We show that $\tilde{\Gamma} \subseteq \Gamma'$, which implies that $\Gamma \subseteq \Gamma'$. Let $\mathcal{G}$ be a canonical database of $\tilde{\Gamma}$ with associated mapping ν. By Proposition 4.11, we have that the treewidth of $\mathcal{G}$ is at most k. Since $\Gamma \equiv \tilde{\Gamma}$, we know that $\tilde{\Gamma} \subseteq \Gamma$, and by Proposition 3.16, we have that $\nu(\bar{x}) \in \Gamma(\mathcal{G})$. We can apply Lemma 4.12 to Γ, $\mathcal{G}$ and $\nu(\bar{x})$, to obtain the C2RPQ $\alpha(\bar{x})$. Lemma 4.12 tell us that the number of atoms in α is at most $|\Gamma|^C$, and the treewidth of α is at most $2k+1$. This implies that α is visited in step (b) of the algorithm, when Γ is given as input. Moreover, the second part of condition (2) of Lemma 4.12 implies that α is

actually part of the output, that is, $\alpha \in \Gamma'$. By the first part of condition (2), we have that $\nu(\bar{x}) \in \alpha(\mathcal{G})$. Since $\alpha \in \Gamma'$, it is the case that $\nu(\bar{x}) \in \Gamma'(\mathcal{G})$, and by Proposition 3.16, we conclude that $\tilde{\Gamma} \subseteq \Gamma'$. $\square$

We conclude by showing Lemma 4.12. The proof combines ideas from Lemma 3.26 and the analysis of tree decompositions.

Proof of Lemma 4.12: The idea behind the construction of α is similar to that of Lemma 3.26, although there are some important differences as we explain through the proof. Let $\gamma(\bar{x}) \leftarrow \mathcal{A}_1(y_1, y_1'), \ldots, \mathcal{A}_m(y_m, y_m')$ be the disjunct of Γ, such that $\bar{n} \in \gamma(\mathcal{G})$. We have a mapping h from the variables of γ to the nodes of $\mathcal{G}$ that satisfies $h(\bar{x}) = \bar{n}$ and a path π_i from $h(y_i)$ to $h(y_i')$ in $\mathcal{G}^{\pm}$ such that $\mathsf{label}(\pi_i)$ is accepted by $\mathcal{A}_i$, for each $1 \leq i \leq m$. As in Lemma 3.26, we assume minimality of π_i, for each $1 \leq i \leq m$.

The C2RPQ $\alpha(\bar{x})$ is based on a subset $\mathcal{V}$ of the nodes of $\mathcal{G}$ that we define below. Let $\mathcal{I}$ be the set $\{h(z) \mid z \text{ is a variable in } \gamma\}$. In the proof of Lemma 3.26, the set $\mathcal{V}$ basically consists of the least common ancestors of elements of $\mathcal{I}$ in the tree structure of $\mathcal{G}$. Here we take the least common ancestors but in the tree decomposition of the underlying graph $\mathcal{U}(\mathcal{G})$ of $\mathcal{G}$. Formally, let (T, λ) be a tree decomposition of $\mathcal{U}(\mathcal{G})$ of width at most k. Without loss of generality we assume that T is a rooted tree. For each $n \in \mathcal{I}$, let t^n be the node in T that is the root of the subtree in T induced by $\{t \mid n \in \lambda(t)\}$. Let $\mathcal{W} = \{t^n \mid n \in \mathcal{I}\}$ and define $\mathcal{W}'$ to be the set of nodes of T that contains $\mathcal{W}$ and all the least common ancestors in T of every pair of nodes in $\mathcal{W}$. Let T' be the (rooted) tree induced by the tree structure of T over the nodes in $\mathcal{W}'$ (as this set is closed under least common ancestors, T' is well-defined), and let λ' be the restriction of λ to the nodes of T'. Then the set $\mathcal{V}$ is defined as $\mathcal{V} = \bigcup_{t \in T'} \lambda'(t)$. Note that $\mathcal{I} \subseteq \mathcal{V}$. Observe also that $|\mathcal{V}|$ is polynomial in $|\Gamma|$. Indeed, we have that $|\mathcal{I}| \leq |\Gamma|$, $|\mathcal{W}| \leq |\mathcal{I}|$ and $|\mathcal{W}'| \leq |\mathcal{W}| + |\mathcal{W}|^2$. Thus $|\mathcal{W}'|$ is polynomial in $|\Gamma|$. Since the size of each $\lambda'(t)$ is at most $k+1$, for each $t \in \mathcal{W}'$, we conclude that $|\mathcal{V}|$ is polynomial in $|\Gamma|$.

As in Lemma 3.26, we decompose the path π_i, for each $1 \leq i \leq m$, according to $\mathcal{V}$ in the obvious way: π_i is decomposed as the concatenation $\pi_i^1 \cdots \pi_i^{k_i}$ of paths $\pi_i^1, \ldots, \pi_i^{k_i}$ where for each $1 \leq j \leq k_i$, the path π_i^j starts and ends at a node in $\mathcal{V}$ and each internal node of π_i^j is not in $\mathcal{V}$. Using the same counting argument as in Lemma 3.26, we can show that $k_i \leq |Q_i| \cdot |\mathcal{V}| - 1$, for each $1 \leq i \leq m$, where Q_i is the set of states of the NFA $\mathcal{A}_i$. In particular, k_i is polynomial in $|\Gamma|$, and hence, the number of paths involved in all the decompositions of the π_i's is polynomial in $|\Gamma|$.

We define the NFAs E_i^js as in Lemma 3.26. More precisely, let $q_0 \cdots q_\ell$ be an accepting run of $\mathcal{A}_i$ over $\mathsf{label}(\pi_i)$. Then for each $1 \leq j \leq k_i$, E_i^j is the NFA obtained from $\mathcal{A}_i$ by setting q_I^j and q_F^j as the unique initial and final state, respectively, where q_I^j is the state we reach in the run $q_0 \cdots q_\ell$ after reading $\mathsf{label}(\pi_i^1 \cdots \pi_i^{j-1})$ (this is the empty word ε when $j = 1$) and q_F^j is defined similarly, but with respect to the word $\mathsf{label}(\pi_i^1 \cdots \pi_i^j)$. Note that $\mathsf{label}(\pi_i^j)$ is accepted by E_i^j. Furthermore, E_i^j is of at most polynomial size in $|\mathcal{A}_i|$, and therefore, in $|\Gamma|$.

Now we are ready to define the C2RPQ $\alpha(\bar{x})$. The variable set of α is $\{z_n \mid n \in \mathcal{V}\}$. The free variables are $\bar{x} = (z_{p_1}, \ldots, z_{p_r})$, assuming that $\bar{n} = (p_1, \ldots, p_r)$. Finally, for each $1 \leq i \leq m$ and $1 \leq j \leq k_i$, the atom $E_i^j(z_n, z_{n'})$ is in α, where n and n' are the initial and terminal nodes of π_i^j, respectively. Observe that this is exactly the definition of the auxiliary

C2RPQ α' in Lemma 3.26. Here, the construction of α is simpler, because we do not need the treewidth of α to be at most k, but only to be bounded by a constant depending on k (in this case $2k+1$). As we shall prove, the C2RPQ α defined above satisfies all the required conditions. In contrast, the construction of α in Lemma 3.26 is more intricate because we need α to be of treewidth 1, i.e., acyclic, starting from a graph database $\mathcal{G}$ that is not acyclic (but pseudo-acyclic). This is why we need the auxiliary C2RPQs α' and α'', as well as more complex NFAs to label the atoms of α.

Using exactly the same arguments as in Lemma 3.26 (for the case of α'), it follows that α satisfies condition (2) and (3) of the lemma. Now we show that the treewidth of α is at most $2k+1$. Consider the pair (T', λ') as defined previously and define λ'' to be the mapping from the nodes of T' to subsets of variables of α defined by $\lambda''(t) = \{z_n \mid n \in \lambda'(t)\}$, for each $t \in T'$. We shall define from (T', λ'') a tree decomposition of width at most $2k+1$ for the underlying graph $\mathcal{U}(\alpha)$ of α. Since (T', λ') is a restriction of the tree decomposition (T, λ), we have that (T', λ'), and in particular, (T', λ'') satisfies the first condition of tree decompositions, namely, that $\{t \in T' \mid z \in \lambda''(t)\}$ is connected in T', for each z in $\mathcal{U}(\alpha)$. However, the second condition of tree decompositions is not satisfied by (T', λ''): for an edge $\{z, z'\}$ in $\mathcal{U}(\alpha)$, it is not clear whether there is a node $t \in T'$ such that $\{z, z'\} \subseteq \lambda''(t)$. Nevertheless, we can show that (T', λ'') satisfies the following weaker condition:

(†) Let $\{z, z'\}$ be an edge in $\mathcal{U}(\alpha)$. Then we have that either there is a $t \in T'$ such that $\{z, z'\} \subseteq \lambda''(t)$, or there are nodes t_1 and t_2 adjacent in T' such that $z \in \lambda''(t_1)$ and $z' \in \lambda''(t_2)$.

By contradiction, assume that (†) is false. Then, there is an edge $\{z, z'\}$ in $\mathcal{U}(\alpha)$ and distinct nodes t_1, t_2, t_3 in T' such that $z \in \lambda''(t_1)$, $z' \in \lambda''(t_2)$, $z, z' \notin \lambda''(t_3)$, and t_3 is in the unique path from t_1 to t_2 in T'. Assume that $\{z, z'\}$ is of the form $\{z_n, z_{n'}\}$, where n and n' are distinct nodes in $\mathcal{G}$. Since $\{z_n, z_{n'}\}$ is an edge in $\mathcal{U}(\alpha)$, there exists $1 \leq i \leq m$ and $1 \leq j \leq k_i$ such that either $E_i^j(z_n, z_{n'})$ or $E_i^j(z_n, z_{n'})$ is an atom in α. Assume that $E_i^j(z_n, z_{n'})$ is an atom in α (the other case is analogous). Then we have that the path π_i^j starts and ends at n and n' in $\mathcal{G}$, respectively. It follows that there is a path π from n to n' in $\mathcal{U}(\mathcal{G})$ such that the internal nodes of π are not in $\mathcal{V}$ (recall that $\mathcal{V} = \bigcup_{t \in T'} \lambda'(t)$; in particular, $n, n' \in \mathcal{V}$). Recall also that (T, λ) is our initial tree decomposition of width k for $\mathcal{U}(\mathcal{G})$, from where (T', λ') and (T', λ'') were defined. In particular, t_1, t_2, t_3 are nodes in T, $n \in \lambda(t_1)$, $n' \in \lambda(t_2)$, $n, n' \notin \lambda(t_3)$, and t_3 is in the unique path from t_1 to t_2 in T.

Note that if we remove t_3 from T, the resulting graph contains exactly two connected components, one containing t_1, denoted by T_1, and the other containing t_2, denoted by T_2. Let G_1 and G_2 be the subgraphs of $\mathcal{U}(\mathcal{G})$ induced by the nodes $\bigcup_{t \in T_1} \lambda(t)$ and $\bigcup_{t \in T_2} \lambda(t)$, respectively. Since $n \in \lambda(t_1)$, we have that n belongs to G_1. Similarly, as $n' \in \lambda(t_2)$, we have that n' is in G_2. Observe that n' is not in G_1. Indeed, if this is true, then there is a node $t_4 \in T_1$ with $n' \in \lambda(t_4)$. By the first condition of tree decompositions, we have that $n' \in \lambda(t_3)$, which is a contradiction. Then, since n and n' are connected by a path in $\mathcal{U}(\mathcal{G})$ and n is in G_1, but n' is not, there must be an edge $\{n_1, n_2\}$ in $\mathcal{U}(\mathcal{G})$ such that n_1 is in G_1 and n_2 is in G_2 but not in G_1. By the second condition of tree decompositions, we conclude that there is a node t^* in T such that $\{n_1, n_2\} \subseteq \lambda(t^*)$.

First, note that $t^* \neq t_3$. Indeed, if $t^* = t_3$, since $n, n' \notin \lambda(t_3)$, we have that n_1 and n_2 are internal nodes of π. Also, since $\lambda(t_3) \subseteq \mathcal{V}$, it follows that n_1 and n_2 are in $\mathcal{V}$, and thus π has internal nodes from $\mathcal{V}$, which is a contradiction. Suppose now that t^* is in T_1. Since n_2 is in G_2, it follows that $n_2 \in \lambda(t_3)$ (first condition of tree decompositions). Since $n' \notin \lambda(t_3)$, we have that $n_2 \neq n'$ and thus n_2 is an internal node of π. Again we conclude that π has an internal node from $\mathcal{V}$, which is a contradiction. The case when t^* is in T_2 is analogous. Hence, (T', λ'') satisfies condition (†).

Let $\tilde{T}$ be the tree obtained from T' by replacing each edge $e = \{t_1, t_2\}$ in T' by two edges $\{t_1, t_e\}$ and $\{t_e, t_2\}$, where t_e is a fresh node depending on e. We define a mapping $\tilde{\lambda}$ from the nodes of $\tilde{T}$ to the subsets of variables of α as follows: If t is in T', then $\tilde{\lambda}(t) = \lambda''(t)$. If $t = t_e$, for an edge $e = \{t_1, t_2\}$ of T', then $\tilde{\lambda}(t) = \lambda''(t_1) \cup \lambda''(t_2)$. By construction, $(\tilde{T}, \tilde{\lambda})$ satisfies the first condition of tree decompositions. Furthermore, since (T', λ'') satisfies condition (†), it follows that $(\tilde{T}, \tilde{\lambda})$ satisfies the second condition of tree decompositions. Thus $(\tilde{T}, \tilde{\lambda})$ is actually a tree decomposition for $\mathcal{U}(\alpha)$. Since $|\tilde{\lambda}(t)| \leq 2k + 2$, for each t in $\tilde{T}$, we conclude that the treewidth of α is at most $2k + 1$.

It remains to define the set $\mathcal{R}_\Gamma$. For an NFA $\mathcal{A}$ and states q, q' in $\mathcal{A}$, let $\mathcal{A}(q, q')$ be the NFA obtained from $\mathcal{A}$ by setting q as the initial state and q' as the only final state. We define $\mathcal{R}_\Gamma = \{\mathcal{A}(q, q') \mid \mathcal{A}$ appears in Γ and q, q' are states of $\mathcal{A}\}$. It is easy to see that $\mathcal{R}_\Gamma$ can be constructed from Γ in polynomial time. Moreover, it is clear that all 2RPQs mentioned in α belong to $\mathcal{R}_\Gamma$. Therefore conditions (1) and (4) are satisfied, which concludes the proof of the lemma.

□

Part II

Containment of regular queries

Chapter 5

Containment of regular queries: Elementary complexity bounds

In this chapter, we study the containment problem for the class of RQs. Our main result is that query containment for RQs is 2Expspace-complete, and therefore, it has elementary complexity. In order to prove this result, we introduce an equivalent query language, called *nested* UC2RPQs (nUC2RPQs), which is of independent interest. A nUC2RPQ is basically an RQ where each extensional predicate could be a 2RPQ. We show that query containment for nUC2RPQs is in 2Expspace, and consequently, we obtain a 2Expspace upper bound for containment of RQs. We also provide matching lower bounds.

To prove that containment of nUC2RPQs is in 2Expspace, we introduce the class of *flat* nUC2RPQs, which are basically unfoldings of nUC2RPQs. In particular, each nUC2RPQ can be unfolded to construct an equivalent flat nUC2RPQ of possibly exponential size. Then we show that containment of flat nUC2RPQs is in Expspace, which implies that containment of nUC2RPQs is in 2Expspace. The proof combines automata-theoretic techniques with database-theoretic techniques. We exploit techniques used before, e.g. in [33, 38, 44], to show that containment of UC2RPQs is in Expspace. Nevertheless, our proof requires a deep understanding and a significant refinement of these techniques. We show the Expspace upper bound for containment of flat nUC2RPQs in two stages:

1. First, we reduce the containment of two flat nUC2RPQs into containment of, essentially, an RPQ in a flat nUC2RPQ. The reduction is based on a *serialization* technique, where we show how to represent expansions of nUC2RPQs as words.
2. Then we tackle the reduced containment problem. The key technique used here is a suitable representation of the *partial mappings* of nUC2RPQs into words, that we call *cuts*, and that allows us to significantly extend the automata notions of previous work (see e.g. [33]). This representation is robust against nesting and does not involve a nonelementary blow up in size.

Then we study some restrictions and extensions of RQs. We first consider RQs of bounded treewidth. While containment for these queries is still 2Expspace-complete, we show that evaluation for them now becomes polynomial time solvable. Then we study RQs of bounded

depth of nesting. Evaluation for these queries is still NP-complete, but now containment becomes EXPSPACE-complete, the same as for UC2RPQs. Regarding extensions, we consider the class of *generalized* RQs that allows intentional predicates to have unbounded arity (by definition, an RQ allows intensional predicates to have only arity 2). We show that containment of generalized RQs is still 2EXPSPACE-complete, but query evaluation becomes harder, namely, PSPACE-complete. We conclude by studying generalized RQs over arbitrary relational schemas. These consist basically of Datalog programs where recursion is only used to express transitive closure. Interestingly, we prove that containment for this class is also 2EXPSPACE-complete.

Organization In Section 5.1 we introduce RQs and nUC2RPQs. The containment problem of RQs is analyzed in Sections 5.2–5.5: Section 5.2 presents and studies flat nUC2RPQs, Section 5.3 shows how to reduce from containment of flat nUC2RPQs to containment of a single-atom C2RPQ in a flat nUC2RPQ, Section 5.4 shows that the latter problem is in EXPSPACE, and Section 5.5 finishes the upper bound and provides the lower bound. Finally, in Section 5.6 we study restrictions and extensions of RQs and their impact on the complexity of evaluation and containment.

5.1 Regular queries and nested UC2RPQs

We now introduce the class of *regular queries* (RQs) and show some basic results regarding the complexity of evaluation. We also present an equivalent way of defining RQs that we call *nested* UC2RPQs (nUC2RPQs), which is better for defining the automata techniques that we use throughout the thesis. Before doing so, we need to review some definitions regarding Datalog programs (for more details see e.g. [4]).

About UCQs Recall that a (U)CQ over a finite alphabet Σ is a (U)CQ over the schema $\sigma(\Sigma)$ (the schema $\sigma(\Sigma)$ contains one binary predicate for each symbol $a \in \Sigma$; see the end of Section 2.2). In this chapter, we define a (U)CQ over Σ to be a (U)CQ over $\sigma(\Sigma^{\pm})$. Recall that $\Sigma^{\pm}$ is the finite alphabet $\Sigma \cup \{a^- \mid a \in \Sigma\}$. Thus the answer of a (U)CQ Θ over a graph database $\mathcal{G}$ is defined as $\Theta(\mathcal{D}(\mathcal{G}^{\pm}))$ (recall that $\mathcal{G}^{\pm}$ is the completion of $\mathcal{G}$; see Section 2.2). Note that there is no gain in expressive power as the atom $a^-(x, y)$ is equivalent to $a(y, x)$. We consider this definition for convenience as it makes more clear our arguments and proofs.

A (U)CQ with equality is a (U)CQ where *equality atoms* of the form $y = y'$ are allowed. Although each CQ with equality can be transformed into an equivalent CQ (without equality) via identification of variables, in some cases it will be useful to work directly with CQs with equality. If φ is a CQ with equality, then its associated CQ (without equality) is denoted by $\mathsf{neq}(\varphi)$. Note that $\mathsf{neq}(\varphi)$ is unique up to renaming of variables.

Datalog. While UC2RPQs extend UCQs with a limited form of transitive closure, Datalog extends UCQs with full recursion. A Datalog *program* Π consists of a finite set of rules of the form $S(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$, where $S, R_1, \ldots, R_m$ are predicate symbols and $\bar{x}, \bar{y}_1, \ldots, \bar{y}_m$ are tuples of variables. A predicate that occurs in the head of a rule is

called *intensional* predicate. The rest of the predicates are called *extensional* predicates. We assume that each program has a distinguished intensional predicate called Ans. Let P be an intensional predicate of a Datalog program Π and $\mathcal{D}$ a database over the extensional predicates of Π. For $i \geq 0$, $P^i_\Pi(\mathcal{D})$ denotes the collection of facts about the intensional predicate P that can be deduced from $\mathcal{D}$ by at most i applications of the rules in Π. Let $P^\infty_\Pi(\mathcal{D})$ be $\bigcup_{i\geq 0} P^i_\Pi(\mathcal{D})$. Then, the *answer* $\Pi(\mathcal{D})$ of Π over $\mathcal{D}$ is $Ans^\infty_\Pi(\mathcal{D})$.

A predicate P *depends* on a predicate Q in a Datalog program Π if Q occurs in the body of a rule ρ of Π and P is the predicate at the head of ρ. The *dependence graph* of Π is a directed graph whose nodes are the predicates of Π and whose edges capture the dependence relation: there is an edge from Q to P if P depends on Q. A program Π is *nonrecursive* if its dependence graph is acyclic, that is, no predicate depends recursively on itself. A (nonrecursive) Datalog program over a finite alphabet Σ is a (nonrecursive) Datalog program Π whose extensional predicates belong to $\sigma(\Sigma^\pm)$. The answer $\Pi(\mathcal{G})$ of a (nonrecursive) Datalog program Π over a graph database $\mathcal{G}$ over Σ is $\Pi(\mathcal{D}(\mathcal{G}^\pm))$.

5.1.1 Regular Queries (RQs)

An *extended Datalog rule* is a rule of the form $S(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$, where S is a predicate and, for each $1 \leq i \leq m$, R_i is either a predicate or an expression P^+ for a binary predicate P. An *extended* Datalog program is a finite set of extended Datalog rules. For an extended Datalog program, we define its extensional/intensional predicates and its dependence graph in the obvious way. Again we assume that there is a distinguished intensional predicate Ans. As expected, a nonrecursive extended Datalog program over an alphabet Σ is an extended Datalog program whose extensional predicates are in $\sigma(\Sigma^\pm)$ and whose dependence graph is acyclic.

Definition 5.1 (Regular query) *A* regular query *(RQ) Ω over a finite alphabet Σ is a nonrecursive extended Datalog program over Σ, where all intensional predicates, except possibly for Ans, have arity* 2.

The semantics of an extended Datalog rule is defined as in the case of a standard Datalog rule considering the semantics of an atom $P^+(y, y')$ as the pairs (n, n') that are in the transitive closure of the relation P. The semantics of a RQ is then inherited from the semantics of Datalog in the natural way. We denote by $\Omega(\mathcal{G})$ the answer of a RQ Ω over a graph database $\mathcal{G}$.

Example 5.2 Recall the examples from Section 1.1.2. We have labels `knows` and `helps` that we abbreviate as k and h, respectively. A person p is a friend of p' if p knows and helps p' at the same time. The following RQ returns all the indirect friends, that is, the persons connected by a chain of friends:

$$F(x, y) \leftarrow k(x, y), h(x, y).$$
$$Ans(x, y) \leftarrow F^+(x, y).$$

□

Example 5.3 Recall from Section 1.1.2 that a person p' is an acquaintance of p if p knows p' and they have an indirect friend in common. The pairs of person connected by a chain of acquaintances can be expressed by the following RQ:

$$\begin{aligned} F(x,y) &\leftarrow k(x,y), h(x,y). \\ A(x,y) &\leftarrow k(x,y), F^+(x,z), F^+(y,z). \\ Ans(x,y) &\leftarrow A^+(x,y). \end{aligned}$$

□

Clearly, RQs subsume UC2RPQs and they are actually strictly more expressive than UC2RPQs. In particular, the transitive closure of a binary UC2RPQ cannot be expressed as a UC2RPQ in general. For instance, using ideas from [25, 26] one can show that the queries in Examples 5.2 and 5.3 cannot be expressed by any UC2RPQ, whereas these were easily expressed as RQs.

We start our investigation of RQs by establishing some basic results regarding query evaluation. Interestingly, we show that RQs are not harder to evaluate than UC2RPQs.

We say that a Datalog program Π is *linear* if we can partition its rules into sets $\Pi_1, \ldots, \Pi_n$ such that (1) the predicates in the head of the rules in Π_i do not occur in the body of any rule in any set Π_j, with $j < i$; and (2) the body of each rule in Π_i has at most one occurrence of a predicate that occurs in the head of a rule in Π_i[1]. A binary linear Datalog program is just a linear program where all intensional predicates have arity 2, except possibly for Ans.

Note then that each expression P^+ in an extended Datalog program can be computed by the following linear Datalog rules (assuming now P^+ is a new predicate):

$$\begin{aligned} P^+(x,y) &\leftarrow P(x,y). \\ P^+(x,y) &\leftarrow P^+(x,z), P(z,y). \end{aligned}$$

Thus, every RQ Ω can be translated in polynomial time into a binary linear Datalog program Π_Ω: one just transforms Ω into a regular Datalog program by treating each of the expressions P^+ as a new predicate, and then adds the rules shown above for each such predicate P^+. It is not difficult to see that the resulting program is indeed linear: since RQs are nonrecursive we can use the same ordering on the rules of Ω to derive a partition for the rules in Π_Ω. For instance, the RQ in Example 5.3 is translated into the following linear program:

$$\begin{aligned} F(x,y) &\leftarrow k(x,y), h(x,y). \\ F^+(x,y) &\leftarrow F(x,y). \\ F^+(x,y) &\leftarrow F^+(x,z), F(z,y). \\ A(x,y) &\leftarrow k(x,y), F^+(x,z), F^+(y,z). \\ A^+(x,y) &\leftarrow A(x,y). \\ A^+(x,y) &\leftarrow A^+(x,z), A(z,y). \\ Ans(x,y) &\leftarrow A^+(x,y). \end{aligned}$$

[1]These programs are sometimes referred to as *stratified linear* programs, or *piecewise linear* programs [150].

As a consequence of this translation we can derive tight complexity bounds for the evaluation problem [44, 52].

Theorem 5.4 *The evaluation problem for RQs is* NP*-complete in general and* NLOGSPACE*-complete in data complexity, that is, when the query is considered to be fixed.*

5.1.2 Nested UC2RPQs (nUC2RPQs)

Most of our proofs assume that RQs are given in an equivalent definition called *nested* UC2RPQs. Nested UC2RPQs can be seen as a more intuitive extension of UC2RPQs, as they allow 2RPQs directly in the atoms of a rule. Also, as we shall see, nested UC2RPQs will allow us to exploit automata techniques in a cleaner way.

An *extended C2RPQ rule* is a rule of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m)$, where for each $1 \leq i \leq m$, R_i is either a predicate, a 2RPQ or an expression P^+, where P is a binary predicate. Again, for a finite set Γ of extended C2RPQ rules, we define its intensional predicates and its dependence graph in the obvious way. Note that the set of nodes of the dependence graph of Γ is the union of the intensional predicates and the 2RPQs mentioned in Γ (now the 2RPQs play the role of extensional predicates). As before, we have a special predicate *Ans* and we say that Γ is a nonrecursive set of rules over alphabet Σ if its dependence graph is acyclic and its 2RPQs are defined over Σ.

Definition 5.5 (Nested UC2RPQ) *A* nested UC2RPQ *(nUC2RPQ)* Γ *over* Σ *is a nonrecursive finite set of extended C2RPQ rules over* Σ*, where all intensional predicates, except possibly for Ans, have arity* 2*.*

The semantics of nUC2RPQs is inherited from the semantics of 2RPQs and Datalog in the natural way. As usual, $\Gamma(\mathcal{G})$ denotes the answer of the nUC2RPQ Γ over the graph database $\mathcal{G}$. The following proposition states that RQs and nUC2RPQs are equivalent; the proof is immediate from the definition.

Proposition 5.6

- *For every RQ* Ω *over* Σ *one can construct in polynomial time a nUC2RPQ* Γ_Ω *such that* $\Omega(\mathcal{G}) = \Gamma_\Omega(\mathcal{G})$ *for every graph database* $\mathcal{G}$ *over* Σ*.*
- *For every nUC2RPQ* Γ *one can construct in polynomial time a RQ* Ω_Γ *such that* $\Gamma(\mathcal{G}) = \Omega_\Gamma(\mathcal{G})$ *for every graph database* $\mathcal{G}$ *over* Σ*.*

5.2 Containment of regular queries

Now we turn to the task of checking containment. Our main result in this part is the following:

Theorem 5.7 *The containment problem for RQs is* 2EXPSPACE*-complete.*

As we have mentioned, to prove this theorem we focus on the equivalent class of nUC2RPQs. Since each RQ can be translated into an equivalent polynomially-sized nUC2RPQ and vice versa, Theorem 5.7 is a direct consequence of the following theorem.

Theorem 5.8 *The containment problem for nUC2RPQs is* 2EXPSPACE*-complete.*

We thus focus solely on Theorem 5.8. Moreover, when showing this theorem we assume without loss of generality that the queries are Boolean, since for every pair of non-Boolean nUC2RPQs $\Gamma(\bar{x})$ and $\Gamma'(\bar{x})$ over Σ we can construct Boolean nUC2RPQs Γ_b and Γ'_b such that Γ is contained in Γ' if and only if Γ_b is contained in Γ'_b. The construction is as follows. Assume that $\bar{x} = (x_1, \ldots, x_n)$. Queries Γ_b and Γ'_b are defined over $\Sigma \cup \{\$_1, \ldots, \$_n\}$, where $\$_1, \ldots, \$_n$ are fresh symbols not in Σ. Query Γ_b is obtained from Γ by adding to each rule of Γ with the *Ans* predicate in its head, an atom $\$_i(x_i, x_i)$, for each $1 \leq i \leq n$. Query Γ'_b is constructed in the same way.

To obtain the required 2EXPSPACE upper bound we use the following approach:

1. We introduce the class of *flat* nUC2RPQs, which are basically unfoldings of nUC2RPQs. In particular, each nUC2RPQ can be unfolded to construct an equivalent flat nUC2RPQ of possibly exponential size.
2. We show that checking containment of flat nUC2RPQs can be reduced in polynomial time to checking containment of a *single-atom* C2RPQ and a flat nUC2RPQ. A single-atom C2RPQ is a query of the form $\gamma \leftarrow \mathcal{A}(y, y')$, where $\mathcal{A}$ is a 2RPQ.
3. We show that containment of a single-atom C2RPQ in a flat nUC2RPQ can be done in EXPSPACE. As a consequence, we obtain that containment of nUC2RPQs can be done in 2EXPSPACE.

The organization of the proof is as follows. In Section 5.2.1 we show how to go from nUC2RPQs to flat nUC2RPQs (with a possible exponential blowup). Then Section 5.3 shows how to reduce from containment of flat nUC2RPQs to containment of a *single-atom* C2RPQ and a flat nUC2RPQ, and Section 5.4 shows that the latter problem can be done in EXPSPACE. Finally, in Section 5.5 we put all the ingredients together and provide a proof for Theorems 5.8 and 5.7. The latter section also includes the proof for the 2EXPSPACE lower bound.

5.2.1 From nUC2RPQs to flat nUC2RPQs

We start by introducing the class of flat nUC2RPQs.

Definition 5.9 (Flat nested UC2RPQ) *A* flat nested UC2RPQ *(flat nUC2RPQ)* Γ *over* Σ *is a nUC2RPQ such that*

1. *For each rule* $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m)$ *and each* $1 \leq i \leq m$, R_i *is either a 2RPQ or an expression of the form* P^+.
2. *For each intensional predicate* S, *there is a unique occurrence of* S *over rule bodies of*

Γ.

Thus in a flat nUC2RPQ, once an intensional predicate S is defined, it can be reused only once and only in the form of a transitive closure. However, note that S can occur several times in the head of rules, that is, it can be defined by more than one rule.

Example 5.10 The following flat nUC2RPQ is equivalent to the query from Example 5.3. Note that we need to rename the two occurrences of predicate F.

$$\begin{aligned} F_1(x,y) &\leftarrow k(x,y), h(x,y). \\ F_2(x,y) &\leftarrow k(x,y), h(x,y). \\ A(x,y) &\leftarrow k(x,y), F_1^+(x,z), F_2^+(y,z). \\ Ans(x,y) &\leftarrow A^+(x,y). \end{aligned}$$

□

As Example 5.10 suggests, each nUC2RPQ can be unfolded to produce an equivalent flat nUC2RPQ. Nevertheless, this process may involve an exponential blow-up in size. We formalize this intuition below.

The *depth* of a nUC2RPQ is the maximum length of a directed path from a 2RPQ to the *Ans* predicate in its dependence graph, minus 1. For instance, queries in Example 5.2 and 5.3 have depth 1 and 2, respectively (formally these queries are RQs but obviously they can be seen as nUC2RPQs too). For an intensional predicate S of a nUC2RPQ, let $\mathsf{rules}(S)$ be the set of rules whose heads mention S. Then the *height* of a nUC2RPQ is the maximum size of $\mathsf{rules}(S)$ over all its intensional predicates. The *width* of a nUC2RPQ is the maximum number of atoms in a rule body. Finally, the *weight* of a nUC2RPQ is the maximum size of a 2RPQ appearing in any rule.

Proposition 5.11 *Let Γ be a nUC2RPQ. Let d, h, w and g be the depth, height, width and weight of Γ, respectively. Then, Γ is equivalent to a flat nUC2RPQ Γ' of depth at most d, height at most $h^{O(w^d)}$, width at most w^{d+1} and weight at most g. In particular, the size of Γ' is at most double-exponential in the size of Γ.*

Proof: The idea is to unfold Γ and whenever necessary, rename intensional predicates by fresh predicate symbols. Below we sketch a procedure that constructs Γ' from Γ.

Let $\Gamma' = \emptyset$. We start by constructing a sequence $\mathcal{R}^0_{Ans}, \ldots, \mathcal{R}^{d+1}_{Ans}$ of sets of rules. Let $\mathcal{R}^0_{Ans}$ be the set of rules in Γ that belongs to $\mathsf{rules}(Ans)$. For $0 \leq i \leq d-1$, $\mathcal{R}^{i+1}_{Ans}$ is constructed from $\mathcal{R}^i_{Ans}$ as follows. Let $\rho \in \mathcal{R}^i_{Ans}$. For each atom in the body of ρ of the form $P(x,y)$, with P an intensional predicate, we choose a rule $P(x',y') \leftarrow \theta' \in \mathsf{rules}(P)$ in Γ and substitute $P(x,y)$ by θ' in ρ, renaming x', y' by x, y, respectively, and the rest of the variables by fresh variables not appearing in ρ. Note that, for each atom $P(x,y)$ we have many choices for the rule $P(x',y') \leftarrow \theta'$, thus the previous step produces a set of rules $\mathcal{R}^{i+1}_{\rho}$. We define $\mathcal{R}^{i+1}_{Ans} = \mathcal{R}^{i+1}_{\rho_1} \cup \cdots \cup \mathcal{R}^{i+1}_{\rho_k}$, where $\mathcal{R}^i_{Ans} = \{\rho_1, \ldots, \rho_k\}$.

By definition of depth, there is $\ell \leq d$ such that $\mathcal{R}^{\ell}_{Ans} = \mathcal{R}^{\ell+1}_{Ans}$. Note that each rule in $\mathcal{R}^{\ell}_{Ans}$

satisfies condition (1) in Definition 5.9, that is, each atom in the body is either a 2RPQ or an expression of the form Q^+. Observe that $|\mathcal{R}^i_{Ans}| \leq h^{1+w+\cdots+w^i}$, for $0 \leq i \leq d$, and thus $|\mathcal{R}^\ell_{Ans}| \leq h^{1+w+\cdots+w^d}$. Moreover, the number of atoms in the body of a rule of $\mathcal{R}^i_{Ans}$ is at most w^{i+1}, for $0 \leq i \leq d$, and thus at most w^{d+1} for $\mathcal{R}^\ell_{Ans}$. We define $\mathcal{S}_{Ans}$ as the set of rules obtained from $\mathcal{R}^\ell_{Ans}$ by renaming each occurrence of an expression Q^+ by I^+, where I is a fresh predicate symbol. The latter implies that rules in $\mathcal{S}_{Ans}$ satisfy condition (2) in Definition 5.9. We add to Γ' all the rules in $\mathcal{S}_{Ans}$ and we mark Ans as *seen*, and the rest of intensional predicates in Γ' as *unseen*.

For each unseen intensional predicate I in Γ', we construct a sequence $\mathcal{R}^0_I, \ldots, \mathcal{R}^{d+1}_I$ as above. Again, there is $n \leq d$, such that $\mathcal{R}^n_I = \mathcal{R}^{n+1}_I$. We thus define $\mathcal{S}_I$ in the same way as $\mathcal{S}_{Ans}$. We add all the rules in $\mathcal{S}_I$ to Γ', and we mark I as seen and all other intentional predicates in $\mathcal{S}_I$ as unseen. Note that the bounds in the size of $\mathcal{R}^\ell_{Ans}$ and the number of atoms in its rules also apply to $\mathcal{R}^n_I$. We continue iteratively until no unseen predicate is left in Γ'.

By construction, Γ' is a flat nUC2RPQ equivalent to Γ. Clearly, the depth of Γ' is at most d, and the weight of Γ' is at most g, as no 2RPQ is modified. By the bounds mentioned above, we have that the height of Γ' is at most $h^{1+w+\cdots w^d} = h^{O(w^d)}$ and the width is at most w^{d+1}. This proves the proposition. □

5.3 From flat nUC2RPQs to single-atom C2RPQs/flat nUC2RPQs

The goal of this section is to show that checking containment of two flat nUC2RPQs Γ and Γ' over Σ can be reduced to checking containment of a single-atom C2RPQ $\tilde{\gamma}$ in a flat nUC2RPQ $\tilde{\Gamma}$ over a larger alphabet Δ. We start by defining the notion of *expansion*, which is central in the analysis of flat nUC2RPQs.

Definition 5.12 (Expansions) *Let Γ be a flat nUC2RPQ over alphabet Σ and let S be an intensional predicate different from Ans. A CQ with equality π over Σ is an* expansion *of S if it is of the form*

$$\pi(x_1, x_2) \leftarrow \pi_1(y_1, y'_1), \ldots, \pi_m(y_m, y'_m)$$

and there is a rule of form $S(x_1, x_2) \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m)$ in Γ such that

1. *For each $1 \leq i \leq m$, if $R_i = \mathcal{A}$ is a 2RPQ, then $\pi_i(y_i, y'_i)$ is a CQ with equality of the form*

$$\pi_i(y_i, y'_i) \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, y'_i)$$

   such that $p \geq 0$, $r_1 \cdots r_p \in L(\mathcal{A})$, where $L(\mathcal{A})$ denotes the language accepted by $\mathcal{A}$, and the z_js are fresh variables. When $p = 0$, we have that $r_1 \cdots r_p = \varepsilon$, and $\pi_i(y_i, y'_i)$ becomes $y_i = y'_i$.

2. *If $R_i = Q^+$ for an intensional predicate Q, then $\pi_i(y_i, y_i')$ is a CQ with equality of the form*

$$\pi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \ldots, \phi_q(w_{q-1}, w_q)$$

where $q \geq 1$, $w_0 = y_i$, $w_q = y_i'$, $w_1, \ldots, w_{q-1}$ are fresh variables and, for each $1 \leq j \leq q$, there is an expansion $\zeta(t_1, t_2)$ of Q such that $\phi_j(w_{j-1}, w_j)$ is obtained from $\zeta(t_1, t_2)$ by renaming t_1, t_2 by w_{j-1}, w_j, respectively, and renaming the rest of the variables by new fresh variables. In particular, the quantified variables of distinct ϕ_i and ϕ_j are disjoint.

Expansions for the predicate Ans are defined similarly. The only difference is that for Ans, expansions are Boolean queries instead of binary. An expansion *of a flat nUC2RPQ is an expansion of its predicate Ans.*

The intuition is that expansions of flat nUC2RPQs are simply expansions of their associated equivalent Datalog program [44, 38]. As it turns out, containment of flat nUC2RPQs can be characterized in terms of containment of CQs. This is an easy consequence of the semantics of CQs [40, 141] and the fact that each flat nUC2RPQ is equivalent to the union of its expansions.

Proposition 5.13 *Let Γ and Γ' be two flat nUC2RPQs. Then, Γ is contained in Γ' if and only if, for each expansion φ of Γ, there exists an expansion φ' of Γ' and a containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$.*

Here, the definition of containment mapping is slightly different to the usual definition [40] due to the presence of inverses:

Definition 5.14 (CQ containment mappings) *Let θ and θ' be Boolean CQs over Σ. A* containment mapping *μ from θ' to θ is a mapping from the variables of θ' to the variables of θ such that, for each atom $r(y, y')$ in θ', with $r \in \Sigma^{\pm}$, either $r(\mu(y), \mu(y'))$ is in θ or $r^-(\mu(y'), \mu(y))$ is in θ.*

Recall that in Section 3.3.1 we gave a similar characterization for containment of UC2RPQs but in terms of canonical databases (see Proposition 3.16). Canonical databases and expansions are dual notions: the canonical databases of a UC2RPQ Γ are essentially the canonical databases (as defined in Section 2.1) of the expansions of Γ. Thus we can see Proposition 5.13 as a characterization dual to that in Proposition 3.16, but extended to flat nUC2RPQs. In this part, we prefer to work with expansions instead of canonical databases, since the former are closer to the syntax of flat nUC2RPQs, and thus they are more natural to work with, when automata techniques are used.

Given flat nUC2RPQs Γ and Γ' over Σ, we construct a single-atom C2RPQ $\tilde{\gamma} \leftarrow \tilde{\mathcal{A}}(y, y')$, where $\tilde{\mathcal{A}}$ is 2RPQ, and a flat nUC2RPQ $\tilde{\Gamma}$ such that Γ is contained in Γ' if and only if $\tilde{\gamma}$ is contained in $\tilde{\Gamma}$. Our reduction is based on two ideas:

1. Expansions of Γ can be "serialized" and represented as *serialized expansions*, which are words over a larger alphabet Δ. More importantly, serialized expansions constitute a regular language. Thus, we can construct a NFA $\tilde{\mathcal{A}}$ such that $L(\tilde{\mathcal{A}})$ is precisely the set of serialized expansions of Γ. This technique has been already used before in [33, 35].

2. Next we need to serialize Γ'. Proposition 5.13 tells us that Γ is contained in Γ' iff Γ' can be "mapped" to each expansion of Γ. We have replaced expansions of Γ by its serializations. By modifying the 2RPQs mentioned in Γ', we construct a flat nUC2RPQ $\tilde{\Gamma}$ such that Γ' can be mapped to an expansion φ of Γ iff $\tilde{\Gamma}$ can be mapped to the serialization of φ. As a consequence, we have that Γ is contained in Γ' iff $\tilde{\gamma}$ is contained in $\tilde{\Gamma}$. This is a new technique and constitutes the crux of the reduction.

From now, we fix flat nUC2RPQs Γ and Γ' and focus on the construction of $\tilde{\mathcal{A}}$ and $\tilde{\Gamma}$.

5.3.1 Construction of $\tilde{\mathcal{A}}$

Let M be the width of Γ, that is, the maximum number of atoms in the body of a rule in Γ. Let d be the depth of Γ. For each $0 \leq i \leq d$, let $\mathcal{V}^i = \{\star^i\} \cup \{h_1^i, \ldots, h_{2M}^i\} \times \{1, 2, \exists\}$ and $\mathcal{S}^i = \{\$^i, 1^i, 2^i\}$. Let $\mathcal{V} = \mathcal{V}^0 \cup \cdots \cup \mathcal{V}^d$ and $\mathcal{S} = \mathcal{S}^0 \cup \cdots \cup \mathcal{S}^d$. We define the alphabet $\Delta = \Sigma^{\pm} \cup \mathcal{V} \cup \mathcal{S}$. The *level* of a symbol $r \in \mathcal{V} \cup \mathcal{S}$ is j iff $r \in \mathcal{S}^j \cup \mathcal{V}^j$. For readability, sometimes we omit the superscripts of symbols in $\mathcal{V} \cup \mathcal{S}$ and refer to them using levels. For a word U in Δ, we define U^{-1} as the word over Δ (if well-defined) obtained from U by replacing $\$^j, \star^j, 1^j, 2^j$ by $\$^{j-1}, \star^{j-1}, 1^{j-1}, 2^{j-1}$, respectively, and (h_i^j, s) by (h_i^{j-1}, s), for $1 \leq i \leq 2M$ and $s \in \{1, 2, \exists\}$.

Now we explain how to represent expansions by words over Δ. This is a natural extension of the representation given in [33] for C2RPQs. The intuition is that we represent variables in an expansion by reusing symbols from $\mathcal{V}$. Symbols of the form (h_i^j, s) represent variables from the program Γ, whereas symbols of the form $\star^j$ represent fresh variables produced when we "unfold" expressions of the form Q^+ (variables $w_1, \ldots w_{q-1}$ in Definition 5.12).

Definition 5.15 (Serialized expansion) *Let π be an expansion of an intensional predicate S of Γ, different from Ans, of the form*

$$\pi(x_1, x_2) \leftarrow \pi_1(y_1, y_1'), \ldots, \pi_m(y_m, y_m')$$

defined by a rule $\rho \in \mathsf{rules}(S)$ of the form

$$S(x_1, x_2) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$$

Let $v_1, \ldots, v_N$ be an enumeration of the variables in $\{y_1, y_1', \ldots, y_m, y_m'\}$ according to the order of appearance in the body of ρ, from left to right. Let Φ be the function from $\{v_1, \ldots, v_N\}$ to $\mathcal{V}$ that maps v_i to $(h_i^d, 1)$ if $v_i = x_1$; to $(h_i^d, 2)$ if $v_i = x_2$; or to $(h_i^d, \exists)$ otherwise. Observe also that Φ is well-defined since $x_1 \neq x_2$ and $N \leq 2M$ (w.l.o.g we can assume $x_1 \neq x_2$ as equality can be simulated by an atom $\mathcal{A}_\varepsilon(x_1, x_2)$, where $L(\mathcal{A}_\varepsilon) = \{\varepsilon\}$).

The serialized expansion *W_π associated with π is the word over Δ of the form*

$$\$^d\ \Phi(y_1)\ W_1\ \Phi(y_1')\ \$^d\ \Phi(y_2)\ W_2\ \Phi(y_2')\ \$^d\ \cdots\ \$^d\ \Phi(y_m)\ W_m\ \Phi(y_m')\ \$^d$$

where for each $1 \leq i \leq m$, W_i is defined as follows:

1. *Suppose* $R_i = \mathcal{A}$ *is a 2RPQ, and* $\pi_i(y_i, y_i')$ *is of the form*

$$\pi_i(y_i, y_i') \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, y_i'),$$

for $p \geq 0$ *and* $r_1 \cdots r_p \in L(\mathcal{A})$*, then* $W_i = r_1 \cdots r_p$*.*

2. *Suppose* $R_i = Q^+$ *for a predicate* Q *and* $\pi_i(y_i, y_i')$ *is of the form*

$$\pi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \ldots, \phi_q(w_{q-1}, w_q),$$

for $q \geq 1$*,* $w_0 = y_i$ *and* $w_q = y_i'$*. Let* $\zeta_j(t_1^j, t_2^j)$ *be the expansion of* Q *defining* $\phi_j(w_{j-1}, w_j)$*, for each* $1 \leq j \leq q$*. Then,* W_i *is of the form*

$$1^d \; W_1' \; 2^d \;\; \star^d \;\; 1^d \; W_2' \; 2^d \;\; \star^d \cdots \star^d \, 1^d \; W_q' \; 2^d$$

where for each $1 \leq j \leq q$*,* $W_j' = (W_j'')^{-1}$*, where* W_j'' *is the serialized expansion associated with* $\zeta_j(t_1^j, t_2^j)$*.*

If φ *is an expansion of Ans, then* W_φ *is defined in a similar way, but now the function* Φ *always maps* v_i *to* $(h_i^d, \exists)$*, since* φ *is Boolean. We say that a word over* Δ *is a* serialized expansion *of* Γ *if it is the serialized expansion associated with some expansion of* Γ*.*

Example 5.16 Suppose Γ is a flat nUC2RPQ over $\Sigma = \{a, b\}$ of the form

$$\begin{aligned} I(x, y) &\leftarrow a(t, y), a(y, x), a(x, t). \\ I(x, y) &\leftarrow bb(x, t), (a + \varepsilon)(z, x), (b + \varepsilon)(z, y). \\ Ans &\leftarrow a^- b^*(x, y), I^+(y, z). \end{aligned}$$

Then the following CQ is a possible expansion φ of Γ:

$$\begin{aligned} \varphi &\leftarrow a^-(x, z_1), b(z_1, y), \phi_1(y, w_1), \phi_2(w_1, z). \\ \phi_1(y, w_1) &\leftarrow b(y, z_1'), b(z_1', t), z' = y, z' = w_1. \\ \phi_2(w_1, z) &\leftarrow a(t', z), a(z, w_1), a(w_1, t'). \end{aligned}$$

The serialized expansion W_φ associated with φ is

$$\begin{array}{cccccccccccccccccccccccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots & 21 & 22 & 23 & \cdots & 37 & 38 & 39 & & & & & & & & & & & & & & & & \\ \$ & h_1 & a^- b & h_2 & \$ & h_2 & 1 \cdots 2 & \star & & & & 1 \cdots 2 & & h_3 & \$ & & & & & & & & & & & & & & & & \\ \$ & f_1 & b & b & h_2 & \$ & h_3 & f_1 & \$ & h_3 & s_4 & \$ & & & & & \$ & h_1 & a & s_2 & \$ & s_2 & a & f_3 & \$ & f_3 & a & h_1 & \$ \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & & & & & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 \end{array}$$

where for readability we omit levels (symbols in the first and second row have level 1 and 0, respectively) and we write h_i, f_i, s_i for $(h_i^\ell, \exists), (h_i^\ell, 1), (h_i^\ell, 2)$, respectively. □

As it turns out, serialized expansions of Γ constitute a regular language.

Proposition 5.17 *Let* Γ *be a flat nUC2RPQ over* Σ*. There exists a NFA* $\tilde{\mathcal{A}}$ *over alphabet* Δ*, such that* $W \in L(\tilde{\mathcal{A}})$ *if and only if* W *is a serialized expansion of* Γ*. Moreover, the size of* $\tilde{\mathcal{A}}$ *is polynomial in* $|\Gamma|$*.*

Proof: We show by induction in the structure of Γ that for all intensional predicates R, we can construct NFAs $\mathcal{A}_R$ and $\mathcal{A}_R^+$ such that

1. $\mathcal{A}_R$ accepts all the serialized expansions of R "modulo levels", that is, each symbol in the word could have any level.
2. $\mathcal{A}_R^+$ accepts all the words of the form
$$1\ W_1'\ 2\ \star\ 1\ W_2'\ 2\ \star \cdots \star\ 1\ W_q'\ 2$$
where $q \geq 1$, the symbols $1, 2, \star$ may have any level and each W_i' is a serialized expansion of R modulo levels.

The base case is a predicate S that does not depend on any other predicates. Let $\rho \in \mathsf{rules}(S)$ be of the form $S(x,y) \leftarrow \mathcal{A}_1(y_1, y_1'), \ldots, \mathcal{A}_m(y_m, y_m')$, where each $\mathcal{A}_i$ is a 2RPQ. We define a NFA $\mathcal{A}_\rho$ that accepts the serialized expansions of S modulo levels that correspond to ρ. Intuitively, $\mathcal{A}_\rho$ ignores levels and checks that the input is of the form
$$\$\ \Phi(y_1)\ W_1\ \Phi(y_1')\ \$\ \Phi(y_2)\ W_2\ \Phi(y_2')\ \$\ \cdots\ \$\ \Phi(y_m)\ W_m\ \Phi(y_m')\ \$$$
where for each $1 \leq i \leq m$, W_i is accepted by the NFA $\mathcal{A}_i$. Note that $\mathcal{A}_\rho$ can be implemented with $O(|\rho|)$ states.

We define $\mathcal{A}_S$ as the union NFA $\mathcal{A}_{\rho_1} \cup \cdots \cup \mathcal{A}_{\rho_k}$, where $\mathsf{rules}(S) = \{\rho_1, \ldots, \rho_k\}$. $\mathcal{A}_S$ accepts all serialized expansions of S modulo levels and it can be implemented with $O(|\Gamma_S|)$ states, where Γ_S is the program obtained from Γ by considering S as the answer predicate. From $\mathcal{A}_S$ is trivial to construct $\mathcal{A}_S^+$ with $O(|\Gamma_S|)$ states.

In the general case, we have an intensional predicate R that depends on predicates $S_1, \ldots, S_p$. By the inductive hypothesis, we already have the NFAs $\mathcal{A}_{S_i}$ and $\mathcal{A}_{S_i}^+$, for each $1 \leq i \leq p$. Let $\rho \in \mathsf{rules}(R)$ be of the form $R(x,y) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$. Again, we construct a NFA $\mathcal{A}_\rho$ that accepts the serialized expansions of R modulo levels associated with ρ. $\mathcal{A}_\rho$ ignores levels and checks that the input is of the form
$$\$\ \Phi(y_1)\ W_1\ \Phi(y_1')\ \$\ \Phi(y_2)\ W_2\ \Phi(y_2')\ \$\ \cdots\ \$\ \Phi(y_m)\ W_m\ \Phi(y_m')\ \$$$
where for each $1 \leq i \leq m$, W_i is accepted by the NFA R_i if R_i is a 2RPQ, or by the NFA $\mathcal{A}_{S_j}^+$ if $R_i = S_j^+$, for some $j \in \{1, \ldots, p\}$. Then $\mathcal{A}_R$ is the union NFA $\mathcal{A}_{\rho_1} \cup \cdots \cup \mathcal{A}_{\rho_k}$, where $\mathsf{rules}(R) = \{\rho_1, \ldots, \rho_k\}$. The NFA $\mathcal{A}_R$ can be implemented with $O(\sum_{j=1}^{k} |\rho_j| + \sum_{j=1}^{p} |\Gamma_{S_j}|) = O(|\Gamma_R|)$ states. Again, we can easily construct $\mathcal{A}_R^+$ with $O(|\Gamma_R|)$ states.

We then define $\tilde{\mathcal{A}}$ as the product NFA $\mathcal{A}_{Ans} \times \mathcal{A}_{\mathsf{levels}}$, where $\mathcal{A}_{\mathsf{levels}}$ is a NFA that checks the correctness of levels. It is not hard to implement $\mathcal{A}_{\mathsf{levels}}$ with $O(d) = O(|\Gamma|)$ states. Clearly, $\tilde{\mathcal{A}}$ accepts precisely the serialized expansions of Γ and it has $O(|\Gamma|^2)$ states. Since the size of the alphabet Δ is polynomial in $|\Gamma|$, the size of $\tilde{\mathcal{A}}$ also is. □

5.3.2 Construction of $\tilde{\Gamma}$

Let $\tilde{\gamma}$ be the single-atom C2RPQ $\tilde{\gamma} \leftarrow \tilde{\mathcal{A}}(y, y')$. We can associate to each word $W = r_1 \cdots r_p \in L(\tilde{\mathcal{A}})$, an expansion θ^W of $\tilde{\gamma}$ of the form $\theta^W \leftarrow r_1(y, z_1), \ldots, r_p(z_{p-1}, y')$. Note that, since

$\varepsilon \notin L(\tilde{\mathcal{A}})$, then θ^W is always a CQ without equality, and thus $\mathsf{neq}(\theta^W) = \theta^W$.

We would like to define $\tilde{\Gamma}$ such that, for each expansion φ of Γ, the following are equivalent:

1. there is an expansion φ' of Γ' and a containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$.
2. there is an expansion ϱ' of $\tilde{\Gamma}$ and a containment mapping from $\mathsf{neq}(\varrho')$ to θ^{W_φ}, where W_φ is the serialized expansion associated with φ.

By Proposition 5.13, this would imply that Γ is contained Γ' iff $\tilde{\gamma}$ is contained in $\tilde{\Gamma}$. Intuitively, item (2) says that $\tilde{\Gamma}$ can be "mapped" to the word W_φ. Thus we need to emulate mappings from Γ' to $\mathsf{neq}(\varphi)$ by mappings from $\tilde{\Gamma}$ to W_φ and vice versa. Since W_φ reuses symbols from $\mathcal{V}$, the main difficulty is that we could have two distinct symbols in W_φ that represent the same variable in $\mathsf{neq}(\varphi)$. These two symbols must be indistinguishable to $\tilde{\Gamma}$. In order to define $\tilde{\Gamma}$, we first characterize indistinguishability of symbols in terms of regular languages.

Let φ be an expansion of Γ. The *internal* variables of φ are the fresh variables produced by expanding 2RPQs, that is, variables z_is in item (1) of Definition 5.12. The rest of the variables in φ are *external* variables. For example, the internal and external variables of expansion φ from Example 5.16 are $\{z_1, z_1'\}$ and $\{x, y, z, w_1, t, z', t'\}$, respectively. Recall that $\mathsf{neq}(\varphi)$ denotes the CQ without equality associated with φ. To obtain $\mathsf{neq}(\varphi)$ we iteratively eliminate equality atoms $y = y'$ and rename y and y' by a fresh variable z. This defines a renaming from the variables of φ to the variables of $\mathsf{neq}(\varphi)$ that we denote ξ_φ.

Let w be a word. For $i, j \in \{1, \ldots, |w|\}$, with $i < j$, $w[i]$ denotes the i-th symbol of w and $w[i, j]$ denotes the subword of w from position i to position j. In a serialized expansion W_φ, some symbols represent variables of φ. This is formalized as a partial mapping var from $\{1, \ldots, |W_\varphi|\}$ to the set of variables of φ. It is clear from the definition of serialized expansion that the symbol $W_\varphi[i]$ represents an external variable y in φ, whenever $i \in \{1, \ldots, |W_\varphi|\}$ and $W_\varphi[i] \in \mathcal{V}$. In this case, we let $\mathsf{var}(i) = y$. For instance, in W_φ from Example 5.16, positions $5, 7, 10, 16$ represent y in φ, positions $19, 22, 31, 33$ represent w_1 and positions $27, 29, 38$ represent z.

We represent internal variables in φ as follows. Suppose we expand a 2RPQ into $r_1 \cdots r_p$ as in item (1) of Definition 5.12 and that this word $r_1 \cdots r_p$ is the subword $W_\varphi[k+1, k+p]$ of W_φ, for some $k \in \{1, \ldots, |W_\varphi|\}$. Then $\mathsf{var}(k+i) = z_i$, for each $1 \leq i \leq p-1$. For all other positions, var is undefined. Note that each internal variable in φ is represented by a unique position. In W_φ from Example 5.16, position 3 represents z_1 and position 11 represents z_1'. Positions $4, 12$ and positions with symbols $\$, 1$ or 2 do not represent any variable in φ.

As mentioned above, different positions i, j in W_φ could represent the same variable in $\mathsf{neq}(\varphi)$. This occurs exactly when $W_\varphi[i], W_\varphi[j] \in \mathcal{V}$ and $\xi_\varphi(\mathsf{var}(i)) = \xi_\varphi(\mathsf{var}(j))$. In this case, we say that positions i and j are *equivalent*. Below we give a simple characterization of equivalent positions.

Let $1 \leq i, j \leq |W_\varphi|$ be positions such that $W_\varphi[i], W_\varphi[j] \in \mathcal{V}$. We have several cases. Suppose first that $W_\varphi[i] = W_\varphi[j] = (h_k^\ell, s)$, for some $1 \leq k \leq 2M$, $0 \leq \ell \leq d$, and $s \in \{1, 2, \exists\}$; and that the symbols $W_\varphi[i], W_\varphi[j]$ are "produced" by the same rule, that is, i

and j belongs to the positions of the symbols $\{\Phi(y_1), \Phi(y'_1), \ldots, \Phi(y_m), \Phi(y'_m)\}$ for a rule

$$S(x, y) \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m)$$

or

$$Ans \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m),$$

as defined in Definition 5.15. In Example 5.16, symbols in $\{2, 5, 7, 22, 38\}$, $\{10, 13, 15, 16, 18, 19\}$ and $\{25, 27, 29, 31, 33, 35\}$ are produced by the same rule. Clearly, i and j are equivalent, since $\mathsf{var}(i) = \mathsf{var}(j)$. In this case, we say that i and j are *horizontally* equivalent. In Example 5.16, the pairs $(5, 7)$, $(15, 18)$ and $(27, 29)$ are horizontally equivalent.

Suppose now that $t\ 1^\ell\ W'\ 2^\ell\ t'$ is a subword of W_φ and that the levels of symbols t, t' is $\ell \in \{1, \ldots, d\}$. Assume that t appears at position i in W_φ, and that $W_\varphi[j]$ is a symbol in W' of the form $(h_k^{\ell-1}, 1)$. Then we have that $\mathsf{var}(i) = \mathsf{var}(j)$ and thus i and j are equivalent. Indeed, this subword appears in W_φ precisely when we unfold an expression Q^+ (item (2), Definition 5.15) and W' represents an expansion ϕ of Q. By definition, $(h_k^{\ell-1}, 1)$ and t represents the same variable, namely the first free variable of ϕ. Analogously, i and j are equivalent if t' appears at position i and $W_\varphi[j]$ is a symbol in W' of the form $(h_k^{\ell-1}, 2)$. In either case we say that i and j are *vertically* equivalent. In Example 5.16, the pairs $(7, 10)$, $(19, 22)$, $(22, 31)$ and $(29, 38)$ are vertically equivalent.

Finally, suppose that $j = i + 1$. Since we are assuming that $W_\varphi[i], W_\varphi[j] \in \mathcal{V}$, the only possibility is that $W_\varphi[i] = (h_k^\ell, s)$ and $W_\varphi[j] = (h_{k'}^\ell, s')$, for some $1 \leq k, k' \leq 2M$, $0 \leq \ell \leq d$ and $s, s' \in \{\exists, 1, 2\}$. This happens exactly when $W_\varphi[i], W_\varphi[j]$ are produced by a rule

$$S(x, y) \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m)$$

or

$$Ans \leftarrow R_1(y_1, y'_1), \ldots, R_m(y_m, y'_m),$$

and for some $1 \leq h \leq m$, it is the case that $\Phi(y_h) = W_\varphi[i]$, $\Phi(y'_h) = W_\varphi[j]$ and $W_h = \varepsilon$ (see Definition 5.15). Since $W_h = \varepsilon$, there is an equality atom $y_h = y'_h$ in φ. Moreover, $\mathsf{var}(i) = y_h$ and $\mathsf{var}(j) = y'_h$. Thus, although $\mathsf{var}(i) \neq \mathsf{var}(j)$, we have that $\xi_\varphi(\mathsf{var}(i)) = \xi_\varphi(\mathsf{var}(j))$, that is, i and j are equivalent. In this case, we say that i and j are *semantically* equivalent. In Example 5.16, the pairs $(15, 16)$ and $(18, 19)$ are semantically equivalent.

It is easy to see that we can only have equivalence of two positions by iteratively applying the above basic type of equivalences. This is stated in the following proposition.

Proposition 5.18 *Let W_φ be the serialized expansion associated with an expansion φ of Γ. Let $i, j \in \{1, \ldots, |W_\varphi|\}$. Then i and j are equivalent if and only if there is a sequence $k_1, \ldots, k_p$ of positions in $\{1, \ldots, |W_\varphi|\}$, such that $p \geq 1$, $k_1 = i$, $k_p = j$ and for each $h \in \{1, \ldots, p-1\}$, k_h and k_{h+1} are either horizontally, vertically or semantically equivalent.*

In Example 5.16, positions 5 and 33 are equivalent. This is because $5, 7$ are horizontally equivalent, $7, 16$ are vertically equivalent, $16, 15$ are semantically equivalent, $15, 18$ are horizontally equivalent, $18, 19$ are semantically equivalent, $19, 22$ are vertically equivalent and $22, 33$ are vertically equivalent.

Now we characterize equivalence in terms of regular languages. Recall that in Section 3.3.1 we defined when a word u folds into another word w. Now we need to define the notion of folding explicitly and introduce more notation. The idea is that u folds into w (as defined in Section 3.3.1) iff there is a folding from u into w.

Definition 5.19 (Foldings) *Let w and u be words over $\Delta^{\pm} = \{a^- \mid a \in \Delta\}$, where Δ is a finite alphabet. A* folding $\mathcal{F}$ *from u into w is a sequence $\mathcal{F} = i_0, i_1, \ldots, i_{|u|}$ of positions in the set $\{0, \ldots, |w|\}$ such that, for each $1 \leq j \leq |u|$, it is the case that $i_j = i_{j-1} + 1$ and $u[j] = w[i_j]$, or $i_j = i_{j-1} - 1$ and $u[j] = w[i_{j-1}]^-$. We denote i_0 and $i_{|u|}$ by $\mathsf{first}(\mathcal{F})$ and $\mathsf{last}(\mathcal{F})$, respectively.*

Intuitively, if there is a folding from u into w, then u can be read in w by a two-way automaton that outputs symbol r, each time it is read from left-to-right, and symbol r^-, each time it is read from right-to-left. If the first symbol that is read in w is the j_1-th symbol, with $j_1 \in \{1, \ldots, |w|\}$, and similarly, the last symbol that is read is the j_2-symbol, with $j_2 \in \{1, \ldots, |w|\}$, then we say that $\mathcal{F}$ is a (j_1, j_2)-folding from u into w. Note that the first or last symbol can be read in w either from left-to-right or from right-to-left. In other words, if $\mathcal{F}$ is a (j_1, j_2)-folding from u into w, then $\mathsf{first}(\mathcal{F}) \in \{j_1 - 1, j_1\}$ and $\mathsf{last}(\mathcal{F}) \in \{j_2 - 1, j_2\}$. For instance, consider the word $w = \$y_1 b^- a y_2\$$. Then, $3, 2, 1, 2, 3, 4, 3$ is a $(3, 4)$-folding from $by_1^- y_1 b^- a a^-$ into w, and $2, 3, 4, 3, 4, 5, 6$ is a $(3, 6)$-folding of $b^- a a^- a y_2\$$ into w.

We introduce the notion of *equality words*. Equality words are words over $\Delta^{\pm}$ with the following key property: For a serialized expansion W_φ and positions $1 \leq i, j \leq |W_\varphi|$, i and j are equivalent iff there is a (i, j)-folding of some equality word into W_φ. Equality words are concatenations of *basic* equality words. We have one type of basic equality word for each type of equivalence. For a level $0 \leq \ell \leq d$, G_ℓ denotes the alphabet $\Delta^{\pm}_{\leq \ell}$, where $\Delta_{\leq \ell}$ is the union of $\Sigma^{\pm}$ and all the symbols from $\mathcal{V} \cup \mathcal{S}$ of level at most ℓ.

1. An *horizontal* equality word α is a word that satisfies the regular expression $t\ G_\ell^*\ t + t^-\ G_\ell^*\ t^-$, where, $0 \leq \ell \leq d$ and t is a symbol of the form (h_k^ℓ, s). Observe that in the definition of horizontal equivalence the fact that $W_\varphi[i]$ and $W_\varphi[j]$ are produced by the same rule is equivalent to the fact that $W_\varphi[i]$ and $W_\varphi[j]$ have the same level ℓ and each symbol between position i and j in W_φ has level at most ℓ. It follows then that i and j are horizontally equivalent iff there is a horizontal equality word α and (i, j)-folding of α into W_φ.
2. A *vertical* equality word α is a word that satisfies the regular expression $e_1 + e_2$, where $e_1 = t_1\ 1^\ell\ G_{\ell-1}^* t_2 + t_2^-\ G_{\ell-1}^* (1^\ell)^-\ t_1^-$, for $t_1 \in \mathcal{V}$ with level ℓ and t_2 of the form $(h_k^{\ell-1}, 1)$. Similarly, $e_2 = t_2\ G_{\ell-1}^* 2^\ell\ t_1 + t_1^-\ (2^\ell)^-\ G_{\ell-1}^* t_2^-$, for $t_1 \in \mathcal{V}$ with level ℓ and t_2 of the form $(h_k^{\ell-1}, 2)$. It is easy to see that i and j are vertically equivalent iff there is a vertical equality word α and (i, j)-folding of α into W_φ.
3. A *semantic* equality word α is a word that satisfies the regular expression $t_1\ t_2 + t_2^-\ t_1^-$, for t_1 of the form (h_k^ℓ, s) and t_2 of the form $(h_{k'}^\ell, s')$. Clearly, i and j are semantically equivalent iff there is a semantic equality word α and (i, j)-folding of α into W_φ.

An equality word is a word α over $\Delta^{\pm}$ that satisfies the regular expression $\delta\alpha_1\delta\alpha_2 \cdots \delta\alpha_p\delta$, where $p \geq 0$, δ is the regular expression $\varepsilon + \mathcal{V}^{\pm}$ and for each $1 \leq h \leq p$, α_h is either an horizontal, vertical or semantic equality word. Note that the empty word ε is an equality

word. The expression δ allows us to concatenate foldings of basic equality word. The following proposition is immediate from the definition of equality word:

Proposition 5.20 *Let W_φ be the serialized expansion associated with an expansion φ of Γ. Let $i, j \in \{1, \ldots, |W_\varphi|\}$. Then i and j are equivalent if and only if there is an equality word α and a (i,j)-folding $\mathcal{F}$ from α into W_φ. Moreover, if i and j are equivalent, then for any pair $(k,k') \in \{i-1, i\} \times \{j-1, j\}$, we can choose α and the (i,j)-folding $\mathcal{F}$ such that $\mathsf{first}(\mathcal{F}) = k$ and $\mathsf{last}(\mathcal{F}) = k'$.*

Example 5.21 Recall that positions 5 and 33 are equivalent in Example 5.16. This is because $5, 7$ are horizontally equivalent, $7, 16$ are vertically equivalent, $16, 15$ are semantically equivalent, $15, 18$ are horizontally equivalent, $18, 19$ are semantically equivalent, $19, 22$ are vertically equivalent and $22, 33$ are vertically equivalent. Equivalence of 5 and 33 is then witnessed by the equality word (now we indicate levels; bold symbols are due to the expression δ; $[\cdot]$ indicates basic equality words)

$$[h_2^1\ \$^1\ h_2^1]\ (\mathbf{h_2^1})^-[h_2^1\ 1^1\ \$^0\ f_1^0\ bb\ h_2^0\ \$^0\ h_3^0\ f_1^0]\ [(f_1^0)^-(h_3^0)^-]\ [h_3^0\ f_1^0\ \$^0\ h_3^0]\ (\mathbf{h_3^0})^-$$
$$[h_3^0\ s_4^0]\ (\mathbf{s_4^0})^-[s_4^0\ \$^0\ 2^1\ \star^1]\ (\star^{\mathbf{1}})^-[\star^1\ 1^1\ \$^0\ h_1^0\ a\ s_2^0\ \$^0\ s_2^0\ a\ f_3^0\ \$^0\ f_3^0]$$

□

Now we define $\tilde{\Gamma}$. Let $w = w_1 \cdots w_p$ be a word over $\Sigma^\pm$. We define $\mathsf{serial}(w)$ as the language that contains all the words over $\Delta^\pm$ of the form $\alpha_0 w_1 \alpha_1 w_2 \alpha_2 \cdots \alpha_{p-1} w_p \alpha_p$, where for each $0 \leq i \leq p$, α_i is an equality word. If L is a language over $\Sigma^\pm$, then $\mathsf{serial}(L)$ is the language over $\Delta^\pm$ defined by $\mathsf{serial}(L) = \{w' \mid w' \in \mathsf{serial}(w), \text{ for some } w \in L\}$. We have the following:

Lemma 5.22 *For each NFA $\mathcal{A}$ over $\Sigma^\pm$, there is an NFA $\mathcal{A}'$ over $\Delta^\pm$ such that $L(\mathcal{A}') = \mathsf{serial}(L(\mathcal{A}))$. Moreover, the size of $\mathcal{A}'$ is polynomial in the size of $\mathcal{A}$ and Δ.*

Proof: It is easy to construct a NFA $\mathcal{A}_{eq}$ of polynomial size in Δ that accepts all the equality words. Then the NFA $\mathcal{A}'$ guesses, on input $s_1 \cdots s_n$, positions $i_1 < \cdots < i_\ell$ such that $s_{i_1} s_{i_2} \cdots s_{i_\ell} \in L(\mathcal{A})$ and it checks that each intermediate subword $s_1 \cdots s_{i_1-1}$, $s_{i_1+1} \cdots s_{i_2-1}$, $\ldots$, $s_{i_\ell+1} \cdots s_n$ is an equality word, that is, it is accepted by $\mathcal{A}_{eq}$. We can construct $\mathcal{A}'$ such that its size is polynomial in the size of $\mathcal{A}$ and $\mathcal{A}_{eq}$, that is, polynomial in the size of $\mathcal{A}$ and Δ. □

For a NFA $\mathcal{A}$, we denote by $\mathsf{serial}(\mathcal{A})$ the NFA $\mathcal{A}'$ from Lemma 5.22. The flat nUC2RPQ $\tilde{\Gamma}$ is obtained from Γ' by replacing each 2RPQ $\mathcal{A}$ in Γ' by $\mathsf{serial}(\mathcal{A})$. Recall that $\tilde{\gamma}$ is the single-atom C2RPQ $\tilde{\gamma} \leftarrow \tilde{\mathcal{A}}(y, y')$ and that θ^W is the CQ associated with the word W. Now we show that $\tilde{\Gamma}$ satisfies our desired property:

Proposition 5.23 *Let φ be an expansion of Γ. Then the following are equivalent:*

1. *there is an expansion φ' of Γ' and a containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$.*
2. *there is an expansion ϱ' of $\tilde{\Gamma}$ and a containment mapping from $\mathsf{neq}(\varrho')$ to θ^{W_φ}.*

Proof: Let ψ be an expansion of an arbitrary flat nUC2RPQ. In the construction of ψ, when we reach the base case (1) in Definition 5.12, we expand a 2RPQ $\mathcal{A}$ by choosing a word in $L(\mathcal{A})$. We call this a *basic expansion*. We identify basic expansions with a set of natural numbers $\mathsf{BE}_\psi = \{1, \ldots, e(\psi)\}$, where $e(\psi)$ is number of times we have to expand a 2RPQ in the whole construction of ψ. We also associate with ψ functions A_ψ and S_ψ. The function A_ψ maps a basic expansion $i \in \mathsf{BE}_\psi$ to the associated 2RPQ $A_\psi(i)$ that we are expanding. The function S_ψ maps $i \in \mathsf{BE}_\psi$ to the word $S_\psi(i) \in L(A_\psi(i))$ that we choose in the expansion. Note that the internal variables are those that appear exactly when we apply a basic expansion.

Recall that $\mathsf{neq}(\psi)$ denotes the CQ without equality associated with ψ and ξ_ψ is the renaming from ψ to $\mathsf{neq}(\psi)$. Let V^{ext} and V^{int} be the external and internal variables of ψ, respectively. Then, we define the *external* and *internal* variables of $\mathsf{neq}(\psi)$ as $\xi_\psi(V^{ext})$ and $\xi_\psi(V^{int})$, respectively. Note that internal variables of ψ and $\mathsf{neq}(\psi)$ coincide, since ξ_ψ is the identity over V^{int}.

We assume that θ^{W_φ} is of the form $\theta^{W_\varphi} \leftarrow r_1(z_0, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, z_p)$, where $p \geq 1$ and $W_\varphi = r_1 \cdots r_p$. For a variable z_j in θ^{W_φ} with $0 \leq j \leq p$, we define $\mathsf{pos}(z_j) = j$.

(1) $\Rightarrow$ (2) Since $\tilde{\Gamma}$ and Γ' only differ in their 2RPQs, we can take the expansion ϱ' to be the "same" as φ': ϱ' expands intensional predicates using the same rule as φ' and expands expression Q^+ as φ' does. In particular, the set of basic expansions of ϱ' and φ' coincide, i.e, $\mathsf{BE}_{\varrho'} = \mathsf{BE}_{\varphi'}$. The only difference between ϱ' and φ' are the words we choose in basic expansions. Thus to completely define ϱ', we only need to specify the function $S_{\varrho'}$.

Let $i \in \mathsf{BE}_{\varrho'}$ and suppose that $S_{\varphi'}(i) = \varepsilon$. Then we assign $S_{\varrho'}(i) = \varepsilon$. Note that this is well-defined, that is, $S_{\varrho'}(i) = \varepsilon \in L(A_{\varrho'}(i))$, since $\varepsilon \in L(A_{\varphi'}(i))$ implies $\varepsilon \in L(\mathsf{serial}(A_{\varphi'}(i))) = L(A_{\varrho'}(i))$. If $S_{\varphi'}(i) \neq \varepsilon$, then we define $B_{\varrho'}(i)$ in such a way that $B_{\varrho'}(i) \neq \varepsilon$. We define $B_{\varrho'}(i)$ later, together with the containment mapping from $\mathsf{neq}(\varrho')$ to θ^{W_φ}. At this point, we have by definition that external variables of φ' and ϱ' coincide. Moreover, by the above (partial) definition of $S_{\varrho'}$ we have that equality atoms (between external variables) also coincide in φ' and ϱ'. Thus w.l.o.g. we can assume that the renamings $\xi_{\varphi'}$ and $\xi_{\varrho'}$ coincide over external variables. In particular, the set of external variables of $\mathsf{neq}(\varphi')$ and $\mathsf{neq}(\varrho')$ is exactly the same. We denote this set by U^{ext}.

Now we completely define $S_{\varrho'}$ and the containment mapping ν from $\mathsf{neq}(\varrho')$ to θ^{W_φ}. We start by defining ν over U^{ext}, that is, the external variables of $\mathsf{neq}(\varrho')$. By hypothesis, we have a containment mapping μ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$. Let ξ_φ be the renaming from φ to $\mathsf{neq}(\varphi)$. Let t be a variable in U^{ext}. Suppose $\mu(t)$ is a variable y in $\mathsf{neq}(\varphi)$. Then we assign $\nu(t) = z_j$, where j is any position in $\{1, \ldots, p\}$ representing the variable y, that is, such that $\xi_\varphi(\mathsf{var}(j)) = y$. Note that we could have many choices for j when y is an external variable of $\mathsf{neq}(\varphi)$, whereas we only have one choice when y is an internal variable. At this point, ν is well-defined over U^{ext}.

Let $i \in \mathsf{BE}_{\varrho'}$ such that $S_{\varphi'}(i) \neq \varepsilon$. It only remains to define $S_{\varrho'}(i)$ and the extension of ν to the internal variables produced by the basic expansion i. Suppose i is a basic expansion between external variables t and t'. Note that $\nu(t), \nu(t')$ are already defined. Observe also that ν can be extended to the internal variables in the expansion i iff there is a folding $\mathcal{F}$ of

$B_{\varrho'}(i)$ into W_φ with $\mathsf{first}(\mathcal{F}) = \mathsf{pos}(\nu(t))$ and $\mathsf{last}(\mathcal{F}) = \mathsf{pos}(\nu(t'))$.

Suppose $S_{\varphi'}(i) = s_1 \cdots s_n$, for $n \geq 1$. Let $o_1 \dots o_{n-1}$ be the internal variables associated with the basic expansion i in φ', and let $o_0 = t$ and $o_n = t'$. We examine the values $\mu(o_0), \mu(o_1), \dots, \mu(o_{n-1}), \mu(o_n)$. Let $j_1 < j_2 < \cdots < j_m$ be all the positions j in $\{0, \dots, n\}$, such that $\mu(o_j)$ is an external variable in $\mathsf{neq}(\varphi)$. Suppose first that $j_1 > 0$ and $j_m < n$. For each $1 \leq k \leq m+1$, let S_k be the subword $s_{j_{k-1}+1} \cdots s_{j_k}$ of $s_1 \cdots s_n$, where $j_0 = 0$ and $j_{m+1} = n$ (note that $S_k \neq \varepsilon$). The intuition is that S_i is "mapped" via μ to a word $S_\varphi(i')$, for some basic expansion i' of φ. Thus for each $1 \leq k \leq m+1$, we have a folding $\mathcal{F}_k$ of S_k into W_φ that simulates the mapping μ. Note that for each $1 \leq k \leq m$, $\mathsf{last}(\mathcal{F}_k) \in \{q-1, q\}$ for a position $q \in \{1, \dots, p\}$ such that $W_\varphi[q] \in \mathcal{V}$ (i.e, q represents an external variable in $\mathsf{neq}(\varphi)$), $\mathsf{first}(\mathcal{F}_{k+1}) \in \{q'-1, q'\}$ for a position $q' \in \{1, \dots, p\}$ such that $W_\varphi[q'] \in \mathcal{V}$, and q and q' are equivalent (since μ is a containment mapping). By Proposition 5.20, there is an equality word α_k and a (q, q')-folding $\mathcal{I}_k$ of α_k into W_φ with $\mathsf{first}(\mathcal{I}_k) = \mathsf{last}(\mathcal{F}_k)$ and $\mathsf{last}(\mathcal{I}_k) = \mathsf{first}(\mathcal{F}_{k+1})$.

We then define $S_{\varrho'}(i) = S_1 \alpha_1 S_2 \alpha_2 \cdots \alpha_m S_{m+1}$. The extension of ν is given by the folding $\mathcal{F}$ obtained from the concatenation of $\mathcal{F}_1, \mathcal{I}_1, \mathcal{F}_2, \cdots, \mathcal{I}_m, \mathcal{F}_{m+1}$. Observe that $\mathcal{F}$ is actually a folding from $S_{\varrho'}(i)$ into W_φ with $\mathsf{first}(\mathcal{F}) = \mathsf{pos}(\nu(o_0)) = \mathsf{pos}(\nu(t))$ and $\mathsf{first}(\mathcal{F}) = \mathsf{pos}(\nu(o_n)) = \mathsf{pos}(\nu(t'))$ as required. Note also that $S_{\varrho'}(i) \in \mathsf{serial}(s_1 \dots s_n) \subseteq L(\mathsf{serial}(A_{\varphi'}(i))) = L(A_{\varrho'}(i))$.

Suppose now that $j_1 = 0$ and $j_m < n$ (the other cases are analogous). Then $S_1 = \varepsilon$. Using the same arguments as above, we have equality words $\alpha_2, \dots, \alpha_m$ and a folding $\mathcal{F}$ from $S_2 \alpha_2 \cdots \alpha_m S_{m+1}$ into W_φ. The problem is that we could have that $\mathsf{first}(\mathcal{F}) \neq \mathsf{pos}(\nu(o_0))$. Nevertheless, we have that $\mathsf{first}(\mathcal{F}) \in \{q-1, q\}$ for a position $q \in \{1, \dots, p\}$ such that $W_\varphi[q] \in \mathcal{V}$, and $\mathsf{pos}(\nu(o_0))$ and q are equivalent. By Proposition 5.20, there is an equality word α and a $(\mathsf{pos}(\nu(o_0)), q)$-folding $\mathcal{I}$ of α into W_φ with $\mathsf{first}(\mathcal{I}) = \mathsf{pos}(\nu(o_0))$ and $\mathsf{last}(\mathcal{I}) = \mathsf{first}(\mathcal{F})$. Then in this case, $S_{\varrho'}(i) = \alpha S_2 \alpha_2 \cdots \alpha_m S_{m+1}$ and our desired folding $\mathcal{F}'$ is the concatenation of $\mathcal{I}, \mathcal{F}$.

(2) $\Rightarrow$ (1) As above, the expansion φ' is the same as ϱ'. We only need to specify $S_{\varphi'}(\cdot)$. Let $i \in \mathsf{BE}_{\varphi'} = \mathsf{BE}_{\varrho'}$ and suppose that $S_{\varrho'}(i) \in \mathsf{serial}(w)$, for some $w \in L(A_{\varphi'}(i))$. Then we assign $S_{\varphi'}(i) = w$.

By construction, the external variables of ϱ' and φ' coincide. We denote these variables by U^{ext}. Note also that, if $S_{\varrho'}(i) = \varepsilon$, the only option for w is ε and thus $S_{\varphi'}(i) = \varepsilon$. In particular, if $y = y'$ is an equality atom in ϱ' with $y, y' \in U^{ext}$, then $y = y'$ is also an equality atom in φ'. However, the converse is not true. If $S_{\varrho'}(i) \neq \varepsilon$, then it could be the case that the only option is $w = \varepsilon$. If this occurs then $S_{\varrho'}(i)$ is simply an equality word. Thus it could be possible that $y = y'$ is an equality atom in φ' but not in ϱ'. As a consequence, the external variables of $\mathsf{neq}(\varrho')$ and $\mathsf{neq}(\varphi')$ do not coincide.

For an external variable z of $\mathsf{neq}(\varphi')$, we define

$$\xi^{-1}_{\varphi'}(z) = \{y \mid z \text{ is external variable in } \varphi' \text{ and } \xi_{\varphi'}(y) = z\}$$

Now we define the containment mapping μ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$. By hypothesis, we have a containment mapping ν from $\mathsf{neq}(\varrho')$ to θ^{W_φ}. First we define μ over the external variables

of $\mathsf{neq}(\varphi')$. Let z be such a variable. Let y be any variable in $\xi^{-1}_{\varphi'}(z)$. By construction, y is also an external variable of ϱ'. Let $j = \mathsf{pos}(\nu(\xi_{\varrho'}(y)))$. We have two cases: (i) $W_\varphi[j]$ or $W_\varphi[j+1]$ belongs to $\mathcal{V}$, or (ii) $W_\varphi[j], W_\varphi[j+1] \notin \mathcal{V}$. If case (i) holds, then let $q \in \{j, j+1\}$ such that $W_\varphi[q] \in \mathcal{V}$. In this case we define $\mu(z) = \xi_\varphi(\mathsf{var}(q))$, that is, $\mu(z)$ is the variable represented by q. Note that if both $W_\varphi[j], W_\varphi[j+1] \in \mathcal{V}$, $\mu(z)$ is independent of the choice of q as in this case j and $j+1$ represent the same variable. If case (ii) holds then we have that $W_\varphi[j], W_\varphi[j+1] \in \Sigma^\pm$ and thus j represents an internal variable in $\mathsf{neq}(\varphi)$. In this case we define $\mu(z) = \xi_\varphi(\mathsf{var}(j))$.

We show that $\mu(z)$ is independent of the choice of y. Let $y, y' \in \xi^{-1}_{\varphi'}(z)$. Then there exists a sequence $t_0 = t_1, t_1 = t_2, \cdots t_{\ell-1} = t_\ell$ of equality atoms in φ', where $\ell \geq 1$, $t_0 = y$ and $t_\ell = y'$. Since the t_ks belongs to U^{ext}, they are also external variables of ϱ'. As we mentioned above, for each $0 \leq k \leq \ell - 1$, $t_\ell = t_{\ell+1}$ is either an equality atom in ϱ' or there is a basic expansion of the form $a_1(t_\ell, y_1), a_2(y_1, y_2), \cdots, a_n(y_{n-1}, t_{\ell+1})$ where $a_1 \ldots a_n$ is a nonempty equality word. As a consequence, we have in $\mathsf{neq}(\varrho')$ a sequence of variables $v_0, \ldots, v_r$ such that $v_0 = \xi_{\varrho'}(y)$, $v_r = \xi_{\varrho'}(y')$ and for each $0 \leq k \leq r-1$, ϱ' contains atoms $a^k_1(v_k, y_1), a_2(y_1, y_2), \cdots, a^k_n(y_{n-1}, v_{k+1})$ with $a^k_1 \cdots a^k_n$ a nonempty equality word. Let $0 \leq k \leq r-1$ and let $j = \mathsf{pos}(\nu(v_k))$ and $j' = \mathsf{pos}(\nu(v_{k+1}))$. Note that, since equality words always start and end with symbols from $\mathcal{V}^\pm$, then j and j' satisfy case (i) in the definition of μ. Then we have position $q \in \{j, j+1\}$ and $q' \in \{j', j'+1\}$ and a (q, q')-folding of $a^k_1 \cdots a^k_n$ into W_φ. It follows that q and q' are equivalent. By an inductive reasoning, we conclude that there are positions q, q' with $q \in \{\mathsf{pos}(\nu(v_0)), \mathsf{pos}(\nu(v_0)) + 1\}$ and $q' \in \{\mathsf{pos}(\nu(v_r)), \mathsf{pos}(\nu(v_r)) + 1\}$ such that q and q' are equivalent. This implies that the value of $\mu(z)$ is independent of y and y'.

It remains to show that μ can be extended to the internal variables of $\mathsf{neq}(\varphi')$. Recall that internal variables of φ' and $\mathsf{neq}(\varphi')$ coincide. Let i be a basic expansion of φ' of the form $w_1(y_0, y_1), w_2(y_1, y_2), \cdots, w_n(y_{n-1}, y_n)$ with $n \geq 1$, where $y_0, y_n \in U^{ext}$. Then $\mathsf{neq}(\varphi')$ contains atoms $w_1(z, y_1), w_2(y_1, y_2), \cdots, w_n(y_{n-1}, z')$, where $z = \xi_{\varphi'}(y_0)$ and $z' = \xi_{\varphi'}(y_n)$. It suffices to show that μ can be extended as a containment mapping to $y_1, \ldots, y_{n-1}$ ($\mu(z)$ and $\mu(z')$ are already defined).

Since $w_1(y_0, y_1), w_2(y_1, y_2), \cdots, w_n(y_{n-1}, y_n)$ is a basic expansion in φ', there is a basic expansion in ϱ' of the from $s_1(y_0, t_1), s_2(t_1, t_2), \cdots, s_r(t_{r-1}, y_n)$ with $r \geq 1$ and $s_1 \cdots s_r \in \mathsf{serial}(w_1 \ldots w_n)$. Thus $\mathsf{neq}(\varrho')$ contains atoms $s_1(\xi_{\varrho'}(y_0), t_1), s_2(t_1, t_2), \cdots, s_r(t_{r-1}, \xi_{\varrho'}(y_n))$. By simulating ν over these atoms and ignoring equality word in $s_1 \cdots s_r$, we can easily extend μ to $y_1, \ldots, y_{n-1}$. Note that this extension is well-defined since we can define $\mu(z)$ and $\mu(z')$ starting from $y_0 \in \xi^{-1}_{\varphi'}(z)$ and $y_n \in \xi^{-1}_{\varphi'}(z')$, respectively. □

As a corollary of Proposition 5.13 and 5.23 we have that:

Corollary 5.24 *Γ is contained in Γ' if and only if $\tilde{\gamma}$ is contained in $\tilde{\Gamma}$.*

Finally note that $\tilde{\gamma}$ and $\tilde{\Gamma}$ can be constructed in polynomial time from Γ and Γ'. Thus we have shown the following theorem.

Theorem 5.25 *There is a polynomial time reduction from the containment problem of flat nUC2RPQs to the containment problem of a single-atom C2RPQ in a flat nUC2RPQ.*

5.4 Containment of single-atom C2RPQs in flat nUC2RPQs

Next we show that containment of a single-atom C2RPQ in a flat nUC2RPQ is in Expspace. We exploit automata-theoretic techniques along the lines of [33, 44, 38]. The main idea is to reduce the problem to checking emptiness of a suitable doubly exponential-sized NFA.

Let us remark that it is by no means obvious how to extend previous techniques [33, 38] to handle nUC2RPQs. In [33], it is shown that checking containment of UC2RPQs Γ and Γ' is in Expspace. This is done by reducing the problem to checking emptiness of a NFA $\mathcal{A}$ that accepts precisely the counterexamples for containment of Γ in Γ', that is, the (word representation of) expansions θ of Γ such that Γ' cannot be mapped to θ. The main construction behind $\mathcal{A}$ is a *two-way* NFA (2NFA) $\mathcal{A}_{\Gamma'}$ that accepts all the expansions θ of Γ such that Γ' can be mapped to θ. Basically, the 2NFA $\mathcal{A}_{\Gamma'}$ guesses the images of each variable of Γ' and then checks that these form a valid mapping from Γ' to θ. Since each atom in Γ' is a 2RPQ $\mathcal{A}$, this can be done easily with a 2NFA by guessing a folding of a word $w \in L(\mathcal{A})$ into θ.

When we have atoms of the form $P^+(x, y)$ in Γ', we cannot apply this idea anymore. The natural extension of $\mathcal{A}_{\Gamma'}$ to this case, will guess a sequence $\phi_1(x, w_1), \ldots, \phi_p(w_{p-1}, y)$ of expansions of P and mappings from these expansions to θ. This would require guessing an unbounded number of images, as the number of variables involved in $\phi_1, \ldots, \phi_p$ is unbounded. Thus it is not clear at all how to extend $\mathcal{A}_{\Gamma'}$ to work with a flat nUC2RPQ Γ'. Instead, we follow a different approach, and construct $\mathcal{A}_{\Gamma'}$ directly as a (one-way) NFA. The automaton $\mathcal{A}_{\Gamma'}$ scans the input θ from left to right and in each step, it guesses a "partial mapping" from Γ' to θ. This is formalized with the notion of *cut*, which we define below.

5.4.1 Cuts

To decide whether a single-atom C2RPQ $\gamma \leftarrow \mathcal{A}(y, y')$ is contained in a (Boolean) flat nUC2RPQ Γ, we have to check that for each expansion θ of γ, there is an expansion φ of Γ and a containment mapping from $\mathsf{neq}(\varphi)$ to $\mathsf{neq}(\theta)$ (Proposition 5.13). We can assume w.l.o.g. that $\varepsilon \notin L(\mathcal{A})$ and replace $\mathsf{neq}(\theta)$ by θ (see the beginning of Section 5.3.2). Note also that expansions for the single-atom C2RPQ γ are CQs of a very particular form, that we call *linear CQs*: they are sequences $w_0(z_0, z_1), \ldots, w_{k-1}(z_{k-1}, z_k)$ where $w_0 \cdots w_{k-1} \in L(\mathcal{A})$ and each z_j is distinct. Thus we can directly identify expansions of γ with words in $L(\mathcal{A})$. A *linearization* of Γ is a linear CQ θ such that there is an expansion φ of Γ and a containment mapping from $\mathsf{neq}(\varphi)$ to θ. Thus the key idea of the proof is to show that the set linearizations (viewed as words) of a flat nUC2RPQ Γ can be characterized by an NFA.

Recall we are assuming w.l.o.g. that our single-atom C2RPQ γ and the flat nUC2RPQ Γ are Boolean queries. Nevertheless, to develop the notion of cut it will be convenient to work with binary flat nUC2RPQs. We will come back to Boolean queries in Section 5.4.4.

Definition 5.26 (Cuts) *Let $\Gamma(x, y)$ be a binary flat nUC2RPQ and recall that* $\mathsf{rules}(Ans)$ *is the set of rules where Ans occur in their head. A* cut *of Γ is a tuple of the form $C = (\rho, Inc, Mark, States)$, where*

- *ρ is a rule in* $\mathsf{rules}(Ans)$*, say of the form*
$$Ans(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m),$$
- *Inc is a subset of the set of variables that appear in ρ, called the* set of variables included in *C;*
- *Mark is a subset of Inc, called the* set of marked variables of *C; and*
- *States is an m-tuple $(s_1, \ldots, s_m)$ such that for each $1 \leq j \leq m$:*
  - *If P_j is a 2RPQ, then s_j is a state of P_j,*
  - *if P_j is an expression P^+, then s_j is a cut of the subquery Γ_P, which is obtained from Γ by considering P as the answer predicate.*

Furthermore, initial cuts *are those in which Inc is empty, and* final cuts *are those in which Inc corresponds to the set of all variables in the rule ρ. We say that a cut* includes *a variable x if it belongs to Inc, and* marks *a variable x if it belongs to Mark.*

Let $\theta \leftarrow w_0(z_0, z_1), \ldots, w_{k-1}(z_{k-1}, z_k)$ be a linear CQ and consider the ordering $z_0 < z_1 \cdots < z_k$. Let $p \in \{0, \ldots, k\}$. Then the intuition is that a cut $C = (\rho, Inc, Mark, States)$ describes a partial mapping of Γ to θ up to all variables smaller than or equal to z_p. The set *Inc* represents variables of Γ that are actually mapped to variables smaller or equal than z_p and the set *Mark* are those that are mapped precisely to variable z_p. For each atom $P_j(u_j, v_j)$ of ρ that is "cut" by the cut C, that is, such that $u_j \in Inc$ and $v_j \notin Inc$, or $v_j \in Inc$ and $u_j \notin Inc$, we store some information in C that will allow us to extend C to another cut in position $p + 1$. If P_j is an NFA, this information is simply a state of P_j. If $P_j = P^+$, this is again a cut for the subquery Γ_P. This is summarized in the tuple *States*. When the j-th atom of ρ is not cut by C, then the value of the j-th coordinate of *States* is irrelevant.

The sets of cuts of a flat nUC2RPQ Γ is denoted by $\mathrm{C}uts(\Gamma)$. Before continuing we show polynomial and exponential bounds of the size and number of cuts of flat nUC2RPQs:

Lemma 5.27 *Let Γ be a binary flat nUC2RPQ. Then $|Cuts(\Gamma)|$ is at most exponential in $|\Gamma|$ and the size of each cut in $Cuts(\Gamma)$ is polynomial in $|\Gamma|$.*

Proof: We show that each cut can be represented by a word of length at most $C|\Gamma|$ over an alphabet of size at most $C'|\Gamma|$, for constants C, C'. This proves that the size of each cut is polynomial, and that the size of $\mathrm{C}uts(\Gamma)$ is exponential, as the number of cuts would be at most $(C'|\Gamma|)^{C|\Gamma|}$.

To describe a cut $C = (\rho, Inc, Mark, States)$, we can write ρ, then list the variables in *Inc* and *Mark*, and write, for each atom $P_j(u_j, v_j)$ of ρ with P_j a 2RPQ, the state s_j. At this

point, the space used is proportional to $|\rho|$. Then we proceed recursively. For each atom $P^+(u_j, v_j)$ of ρ, we write the description of the cut s_j of Γ_P. Note that, if $P^+(u_j, v_j)$ and $Q^+(u_k, v_k)$ are distinct atoms in ρ, since Γ is a flat nUC2RPQ, then the rules in Γ_P and Γ_Q are disjoint. This implies that the size of our representation of cuts is proportional to $|\Gamma|$. □

5.4.2 Transition System Based on Cuts

We represent each linear CQ $\theta \leftarrow w_0(z_0, z_1), \ldots, w_{k-1}(z_{k-1}, z_k)$ by the word $w = w_0, \ldots, w_{k-1}$, and each partial mapping from a flat nUC2RPQ Γ to θ as a pair $Cuts(\Gamma) \times \{0, \ldots, k\}$. Our next step is to define a transition system $T_{(\Gamma,w)}$ defined over cuts of Γ and positions of a word w over $\Sigma^{\pm}$. A *configuration* of $T_{(\Gamma,w)}$ is just a pair from $Cuts(\Gamma) \times \{0, \ldots, k\}$, and represents that a certain cut is assigned to a certain position of w. The system $T_{(\Gamma,w)}$ relates configurations according to a transition relation $\Rightarrow_{(\Gamma,w)}$ that ranges over $(Cuts(\Gamma) \times \{0, \ldots, k\}) \times (Cuts(\Gamma) \times \{0, \ldots, k\})$. As usual $\Rightarrow^*_{(\Gamma,w)}$ denotes the reflexive and transitive closure of the relation $\Rightarrow_{(\Gamma,w)}$.

Let us shed light on the intuition behind the system. We note first that our transition system, while non-deterministic, can only *advance* to configurations relating greater or equal positions in w. The idea is that a run of $T_{(\Gamma,w)}$ should non-deterministically guess the greatest cuts, in terms of variables, that can be mapped to each position in w. For the same reason, the transition system can only move towards configurations whose cuts include at least those variables included in previous configurations.

Example 5.28 Consider a query over alphabet $\Sigma = \{a, b\}$ given by the single rule ρ: $\Gamma(x, y) \leftarrow a^+(x, z), a^-(z, y)$, stating that there is a path labeled with a^+ between x and z, and a reverse a-labelled edge between z and y. It is not difficult to see that the CQ $\theta \leftarrow a(u_1, u_2), a(u_2, u_3)$ is a linearization of Γ. Indeed, for example, the expansion $a(x, x')$, $a(x', z)$, $a(y, z)$ can be mapped onto θ.

The word associated to θ is $w = aa$. Assume now that the NFA for a^+ is $(\{q_0, q_f\}, \Sigma, q_0, \{q_f\}, \delta)$, with $\delta(q_0, a) = q_f$ and $\delta(q_f, a) = q_f$ and the NFA for a^- is $(\{p_0, p_f\}, \Sigma, p_0, \{p_f\}, \delta)$, with $\delta(q_0, a^-) = p_f$.

A valid run for $T_{(\Gamma,w)}$ over w starts in the initial cut $(\rho, \{\}, \{\}, (q_0, p_0))$ at the beginning of the word and then advances to cut $(\rho, \{x\}, \{x\}, (q_0, p_0))$ while still in position 0 of w. This means we have non-deterministically guessed that we will start checking the conjunct $a^+(x, z)$ of ρ. We then advance to position 1 in w and cut $(\rho, \{x\}, \{\}, (q_f, p_0))$, which reflects the transition of $a^+(x, z)$ when reading an a, and then to $(\rho, \{x, y\}, \{y\}, (q_f, p_f))$ as we guess that we now start checking the second conjunct as well. Since y is the second variable of $a^-(z, y)$, we need to satisfy the automaton for a^- *in reverse*, and this is why we start in state p_f in the second position of the tuple of states. We then advance to position 2 and cut $(\rho, \{x, y\}, \{\}, (q_f, p_0))$, reflecting the transition of the NFAs (the second in reverse), and finally to the cut $(\rho, \{x, y, z\}, \{z\}, (q_f, p_0))$. Since this last cut is final, we determine that $T_{(\Gamma,w)}$ can advance from an initial state to a final state. □

To define the relation $\Rightarrow_{(\Gamma,w)}$, we fix a nested UC2RPQ Γ and a word $w = w_0, \ldots, w_{k-1}$.

We begin with the set of transitions that relates configurations in subsequent positions.

Recall that for a symbol $r \in \Sigma^{\pm}$, r^- denotes the symbol a^- if $r = a \in \Sigma$, or the symbol a if $r = a^-$ for $a \in \Sigma$. Let $p \in \{0, \ldots, k-1\}$. We have that $(C, p) \Rightarrow_{(\Gamma,w)} (C', p+1)$, if C and C' are cuts of the form $C = (\rho, \mathit{Inc}, \mathit{Mark}, \mathit{States})$ and $C = (\rho, \mathit{Inc}, \emptyset, \mathit{States}')$ with $\mathit{States} = (s_1, \ldots, s_m)$, $\mathit{States}' = (s'_1, \ldots, s'_m)$ and ρ of the form

$$\mathit{Ans}(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m)$$

such that for each $1 \leq j \leq m$, one of the following holds:

- A.1 $u_j, v_j \in \mathit{Inc}$ and $s_j = s'_j$,
- A.2 $u_j, v_j \notin \mathit{Inc}$ and $s_j = s'_j$,
- A.3 $u_j \in \mathit{Inc}$, $v_j \notin \mathit{Inc}$, and it holds that either P_j is an NFA and there is a w_p-transition in P_j from s_j to s'_j, or $P_j = P^+$ and $(s_j, p) \Rightarrow_{(\Gamma_P,w)} (s'_j, p+1)$,
- A.4 $v_j \in \mathit{Inc}$, $u_j \notin \mathit{Inc}$, and it holds that either P_j is an NFA and there is a w_p^--transition in P_j from s'_j to s_j, or $P_j = P^+$ and $(s_j, p) \Rightarrow_{(\Gamma_P,w)} (s'_j, p+1)$.

In other words, the only way to move to a greater position by $\Rightarrow_{(\Gamma,w)}$ is to move from a cut C to a cut C' that includes the same variables as C, unmark all marked variables and where for each atom $P_j(u_j, v_j)$ with exactly one variable in Inc, it must be the case that we can advance from p to $p+1$, reading w_p, either according to $\Rightarrow_{(\Gamma_P,w)}$ or to the NFA of P_j, depending on whether P_j is an NFA or the transitive closure of the predicate P.

Recall the notion of folding (Definition 5.19) from Section 5.3.2. We say that $\mathcal{F}$ is a $[p, p']$-folding if $p = \mathsf{first}(\mathcal{F})$ and $p' = \mathsf{last}(\mathcal{F})$. Note that this is slightly different from a (p, p')-folding as defined in Section 5.3.2, where p and p' are the positions of the first and last symbol read by the folding, respectively.

Next we describe the piece of $\Rightarrow_{(\Gamma,w)}$ that relates cuts in the same position. Intuitively, these represent two types of transitions: Either we are including new variables in C, or we are synchronizing the content of the tuple *States*.

It is best if we start defining those for the case of flat nUC2RPQs of depth 0. Thus, assume for now that Γ has nesting depth 0. We then have that $(C, p) \Rightarrow_{(\Gamma,w)} (C', p)$, for cuts $C = (\rho, \mathit{Inc}, \mathit{Mark}, \mathit{States})$ and $C' = (\rho, \mathit{Inc}', \mathit{Mark}', \mathit{States}')$, with $\mathit{States} = (s_1, \ldots, s_m)$, $\mathit{States}' = (s'_1, \ldots, s'_m)$, $\mathit{Inc} \subseteq \mathit{Inc}'$, $\mathit{Mark}' = \mathit{Mark} \cup \mathit{Inc}' \setminus \mathit{Inc}$ and ρ of the form

$$\mathit{Ans}(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m).$$

if for each $1 \leq j \leq m$, one of the following holds:

- B.1 Both C and C' include both u_j and v_j,
- B.2 Both C and C' do not include any of u_j and v_j,
- B.3 C does not include any of u_j and v_j, C' includes and marks u_j but does not include v_j and s'_j is an initial state.
- B.4 C does not include any of u_j and v_j, C' includes and marks v_j but does not include u_j and s'_j is a final state.

B.5 C includes only u_j, C' also includes and marks v_j and $s_j = s'_j$ is a final state,

B.6 C includes only v_j, C' also includes and marks u_j and $s_j = s'_j$ is an initial state,

B.7 Both C and C' include only u_j but not v_j and there is a $[p, p]$-folding τ such that there is a run for the automaton for P_j, reading τ, that starts in state s_j and ends in state s'_j, or

B.8 Both C and C' include only v_j but not u_j and there is a $[p, p]$-folding τ such that there is a run for the automaton for P_j, reading τ^-, that starts in state s'_j and ends in state s_j, where $\tau^- = \tau_\ell^-, \ldots, \tau_1^-$, if $\tau = \tau_1, \ldots, \tau_\ell$.

Intuitively, $(C, p) \Rightarrow_{(\Gamma,w)} (C', p)$ if for every atom $P_j(u_j, v_j)$ of Γ we have that both of u_j or v_j have already been included in C (condition B.1); or that neither u_j not v_j have been included in C or C' (condition B.2); or that we are guessing that p is the position where the first variable of $P_j(u_j, v_j)$ is witnessed (conditions B.3-B.4), or a previous position has already witnessed one variable of $P_j(u_j, v_j)$ and p is the position where the last variable of $P_j(u_j, v_j)$ is witnessed (conditions B.5-B.6), or it is the case that we are in the middle of a run for the NFA of P_j, and there is a valid run advancing from state s_j to s'_j that can be folded from p to p (conditions B.7-B.8).

For simplicity, we excluded the case when C does not include any of u_j and v_j and C' includes both u_j and v_j. This is not a problem since we can assume w.l.o.g. that for each $1 \leq j \leq m$, $u_j \neq v_j$, by adding atoms $\varepsilon(u_j, v_j)$ whenever necessary. Note also that the set *Mark* is completely determined by *Inc*: it always contains the new variables added to *Inc* at the current position.

For queries with higher depth, the definition of $\Rightarrow_{(\Gamma,w)}$ is analogous, but now s_j and s'_j could be cuts, instead of states of an NFA. Thus we need to generalize the notions of initial/final states (conditions B.3-B.6) and $[p, p]$-foldings (conditions B.7-B.8) to the context of cuts. We explain these generalizations below.

Let (C, p) and (C', p') be two configurations where $p, p' \in \{0, \ldots, k\}$. We say that (C, p) and (C', p') define an *accepting run* for the flat nUC2RPQ $\Gamma(x, y)$ over w if the following holds:

- C marks x and C' marks y, and
- there is a sequence of configurations $(C_0, p_0), (C_1, p_1), \ldots, (C_n, p_n)$ such that
  - $p_0 \leq p_1 \leq \cdots \leq p_n$, $p_0 = 0$, $p_n = k$,
  - C_0 and C_n are initial and final cut of Γ, respectively,
  - $(C_i, p_i) \Rightarrow_{(\Gamma,w)} (C_{i+1}, p_{i+1})$, for each $1 \leq i \leq n-1$, and
  - (C, p) and (C', p') appear in the sequence.

The idea is that (C, p) and (C', p') define an accepting run exactly when $\Gamma(x, y)$ can be mapped to w in such a way that x and y are mapped to positions p and p', respectively.

Let us give some intuition about the subsequent technical definitions. For queries of higher depth, we must also consider atoms of the form $\Gamma^+(x, y)$ where $\Gamma(x, y)$ is a flat nested UC2RPQ itself. Suppose that $\Gamma^+(x, y)$ can be mapped to a word w in such a way that x and y are mapped to positions p_0 and p_ℓ, respectively (for simplicity, assume $p_0 < p_\ell$). Our

transition system must capture this by a sequence of configurations $(C_1, q_1), \ldots, (C_n, q_n)$, that will corresponds to the coordinate of *States* associated with the atom $\Gamma^+(x, y)$. This sequence must satisfy that C_i is a cut of Γ, $q_1 = p_0, q_n = p_\ell$ and $q_1 \leq \cdots \leq q_n$. Moreover, C_1 must be "initial", C_n must be "final", and the transition from (C_j, q_j) to (C_{j+1}, q_{j+1}) is due to $\Rightarrow_{(\Gamma,w)}$ or due to special transitions we call "transitive runs" (which generalize conditions B.7-B.8).

Suppose $\phi_1(z'_0, z'_1), \phi_2(z'_1, z'_2), \ldots, \phi_q(z'_{q-1}, z'_q)$ is the expansion of $\Gamma^+(x, y)$ that can be mapped to w, where each ϕ_i is an expansion of $\Gamma(x, y)$. Let $\eta(z'_0), \ldots, \eta(z'_q)$ be the positions in $\{0, \ldots, |w|\}$, where the variables $z'_0, \cdots z'_q$ are mapped. Note that $\eta(z'_0) = p_0$ and $\eta(z'_q) = p_\ell$. Assume that $\eta(z'_0) < \eta(z'_1) < \cdots < \eta(z'_q)$. In this case our sequence $(C_1, q_1), \ldots, (C_n, q_n)$ is as follows. The configuration (C_1, q_1) must be initial: C_1 marks x and there is an initial cut C_I with $(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C_1, q_1)$. We need this last requirement because x may not be the first variable (form left to right) that is mapped to w. This will be condition (1) in the definition of initial cut w.r.t a position. Similarly, (C_n, q_n) must be final: C_n marks y and there is a final cut C_F with $(C_n, q_n) \Rightarrow^*_{(\Gamma,w)} (C_F, k)$. Again we need this last requirement because y may not be the last variable that is mapped to w. This is condition (1) in the definition of final cut w.r.t a position.

All the other transitions from (C_j, q_j) to (C_{j+1}, q_{j+1}) are given by the accepting runs of the expansions $\phi_1, \ldots, \phi_q$, except when we switch from expansion ϕ_i to expansion ϕ_{i+1}. Instead, this corresponds to a transition from a configuration (C_j, q_j) to (C_{j+1}, q_{j+1}), where $q_{j+1} = q_j$, C_j is final at position q_j, C_{j+1} marks x and there is an initial cut C_I with $(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C_{j+1}, q_{j+1})$. This corresponds to the notion of transitive run (see the definition below).

Unfortunately, the general situation is more intricate as the order of the positions $\eta(z'_0)$, $\ldots$, $\eta(z'_q)$ could be arbitrary. Thus the idea is to simplify the cycles in $\eta(z'_0), \ldots, \eta(z'_q)$ by strengthening the definition of initial/final cuts and of transitive run. Next we formalize this idea.

We say that a cut C is an *initial cut* for $\Gamma(x, y)$ over w at position p, with $p \in \{0, \ldots, k\}$ if one of the following two cases holds:

1. C marks x and there is initial cut C_I of Γ such that $(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C, p)$.
2. There is a nonempty sequence $p_0, p_1, p_2, \ldots, p_\ell$ of positions in $\{0, \ldots, k\}$ and cuts $C^x, C^y_0, C^x_0, C^y_1, C^x_1, , \ldots, C^y_\ell, C^x_\ell$ such that
   - $p_\ell < p$,
   - (C^x, p) and (C^y_0, p_0) define an accepting run for $\Gamma(x, y)$ over w,
   - For each $0 \leq i \leq \ell - 1$, (C^x_i, p_i) and (C^y_{i+1}, p_{i+1}) define an accepting run for $\Gamma(x, y)$ over w, and
   - C^x_ℓ marks x and there is an initial cut C_I of Γ such that

$$(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C^x_\ell, p_\ell) \Rightarrow^*_{(\Gamma,w)} (C, p)$$

A cut C is a *final cut* for $\Gamma(x, y)$ over w at position $p \in \{0, \ldots, k\}$, if one of the following two cases holds:

1. There is a cut C^y and final cut C_F of Γ such that C^y marks y, $(C,p) \Rightarrow^*_{(\Gamma,w)} (C^y,p) \Rightarrow^*_{(\Gamma,w)} (C_F,k)$.
2. There is a nonempty sequence $p_0, p_1, p_2, \ldots, p_\ell$ of positions in $\{0,\ldots,k\}$ and cuts $C_0^y, C_0^x, C_1^y, C_1^x, , \ldots, C_\ell^y, C_\ell^x, C^y$ such that
   - $p < p_i$, for each $0 \leq i \leq \ell$,
   - C_0^y marks y and there is a final cut C_F of Γ such that

$$(C,p) \Rightarrow^*_{(\Gamma,w)} (C_0^y, p_0) \Rightarrow^*_{(\Gamma,w)} (C_F,k)$$

   - For each $0 \leq i \leq \ell-1$, (C_i^x, p_i) and (C_{i+1}^y, p_{i+1}) define an accepting run for $\Gamma(x,y)$ over w, and
   - (C_ℓ^x, p_ℓ) and (C^y, p) define an accepting run for $\Gamma(x,y)$ over w.

Finally, we say that (C,p) and (C',p), with $p \in \{0,\ldots,k\}$, define a *transitive run* for $\Gamma(x,y)$ over w if

- C' marks x and there is an initial cut C_I with $(C_I,0) \Rightarrow^*_{(\Gamma,w)} (C',p)$, and
- C is a final cut for $\Gamma(x,y)$ over w at position p.

We can now finish the definition of $\Rightarrow_{(\Gamma,w)}$ for queries of higher depth. Let then Γ be again a binary flat nUC2RPQ, and let ρ be a rule in Γ of the form

$$Ans(x,y) \leftarrow P_1(u_1,v_1), \ldots, P_m(u_m,v_m),$$

Then we have that $(C,p) \Rightarrow_{(\Gamma,w)} (C',p)$, for cuts $C = (\rho, \mathit{Inc}, \mathit{Mark}, \mathit{States})$ and $C' = (\rho, \mathit{Inc}', \mathit{Mark}', \mathit{States}')$ in $\mathrm{C}uts(\Gamma)$, with $\mathit{Inc} \subseteq \mathit{Inc}'$, $\mathit{Mark}' = \mathit{Mark} \cup \mathit{Inc}' \setminus \mathit{Inc}$, $\mathit{States} = (s_1,\ldots,s_m)$ and $\mathit{States}' = (s'_1,\ldots,s'_m)$, if for each $1 \leq j \leq m$ where P_j is a 2RPQ one of conditions B.1-B.8 hold; and for each $1 \leq j \leq m$ where P_j is an expression P^+, one of the following conditions hold:

D.1 Both C and C' include both u_j and v_j,

D.2 Both C and C' do not include any of u_j and v_j,

D.3 C does not include any of u_j and v_j; C' includes and marks u_j, but does not include v_j; and s'_j is an initial cut for $\Gamma_P(u_j,v_j)$ over w at position p,

D.4 C does not include any of u_j and v_j; C' includes and marks v_j, but does not include u_j; and s'_j is an initial cut for $\Gamma_P(v_j,u_j)$ over w at position p,

D.5 C includes u_j but not v_j; C' includes and marks v_j; $s'_j = s_j$ and s'_j is a final cut for $\Gamma_P(u_j,v_j)$ over w at position p,

D.6 C includes v_j but not u_j; C' includes and marks u_j; $s'_j = s_j$ and s'_j is a final cut for $\Gamma_P(v_j,u_j)$ over w at position p,

D.7 Both C and C' include u_j but not v_j, and either $(s_j,p) \Rightarrow_{(\Gamma_P,w)} (s'_j,p)$, or (s_j,p) and (s'_j,p) define a transitive run for $\Gamma_P(u_j,v_j)$ over w,

D.8 Both C and C' contain v_j but not u_j, and either $(s_j,p) \Rightarrow_{(\Gamma_P,w)} (s'_j,p)$, or (s_j,p) and (s'_j,p) define a transitive run for $\Gamma_P(v_j,u_j)$ over w.

The following lemma states the correctness of our system $T_{(\Gamma,w)}$:

Lemma 5.29 *Let $\Gamma(x,y)$ be a flat nUC2RPQ, $w = w_0,\ldots,w_{k-1}$ a word over $\Sigma^{\pm}$ and $p,p' \in \{0,\ldots,k\}$. There are pairs (C,p) and (C',p') over $Cuts(\Gamma)\times\{0,\ldots,k\}$ that define an accepting run for $\Gamma(x,y)$ over w if and only if there is an expansion φ of Γ and a containment mapping from $\mathsf{neq}(\varphi)$ to the linear CQ $\theta^w \leftarrow w_0(z_0,z_1),\ldots w_{k-1}(z_{k-1},z_k)$ that maps x to z_p and y to $z_{p'}$.*

Proof: ($\Longrightarrow$) The proof is by induction on the depth of Γ. We start with the base case when Γ is a UC2RPQ. Assume that there are pairs (C,p) and (C',p') over $\mathrm{C}uts(\Gamma)\times\{0,\ldots,k\}$ that define an accepting run for Γ over w. Then we assume that $C = (\rho, \mathit{Inc}, \mathit{Mark}, \mathit{States})$ and $C' = (\rho, \mathit{Inc}', \mathit{Mark}', \mathit{States}')$ in $\mathrm{C}uts(\Gamma)$, where ρ is one of the rules in Γ, of the form

$$Ans(x,y) \leftarrow P_1(u_1,v_1),\ldots,P_m(u_m,v_m),$$

where each P_j is an NFA.

Let $\mathcal{S}$ be the sequence of configurations that witnesses the accepting run of (C,p) and (C',p'). We use $\mathcal{S}$ to define the following expansion π of Γ:

$$\pi(x,y) \leftarrow \pi_1(u_1,v_1),\ldots\pi_m(u_m,v_m).$$

Next we define π_j, for each $1 \le j \le m$. Consider then an arbitrary j in $\{1,\ldots,m\}$. Let (C_0,p_0) be the configuration in $\mathcal{S}$ such that C_0 marks for the first time the variable u_j. Likewise, define (C_ℓ,p_ℓ) with respect to v_j. We proceed depending on which of p_0 or p_ℓ is greater.

Assume first that $p_0 \le p_\ell$. Let $s_j^0,\ldots,s_j^\ell$ be the sequence corresponding to the j-th element of *States* in each of the cuts $C_0,\ldots,C_\ell$ between C_0 and C_ℓ in the sequence $\mathcal{S}$. Then by definition of $\Rightarrow_{(\Gamma,w)}$, it must be the case that s_j^0 is an initial state and s_j^ℓ is a final state of P_j.

The sequence $s_j^0,\ldots,s_j^\ell$ defines a valid run for the NFA P_j over a word τ that can be $[p_0,p_\ell]$-folded into w. Indeed, if going from a cut C_a to a cut C_b in the sequence $C_0,C_1,\ldots,C_\ell$ we advance from position q to position $q+1$, then by definition we have that s_j^b can be obtained by a transition of the NFA when in state s_j^a and reading w_q. Otherwise, if from C_a to C_b we do not advance, but rather stay in position q, then there is a $[q,q]$-folding into w that defines a run from s_j^a to s_j^b. The word τ is then obtained by concatenating all these foldings together with the symbols $w_{p_0},\ldots,w_{p_\ell}$ in the order given by the sequence $C_0,C_1,\ldots,C_\ell$.

Finally, if $\tau = r_1,\ldots,r_{|\tau|}$ is the word that can be $[p_0,p_\ell]$-folded into w, π_j is defined as the CQ

$$\pi_j(u_j,v_j) \leftarrow r_1(u_j,z_1'),r_2(z_1',z_2'),\ldots,r_{|\tau|}(z_{|\tau|-1}',v_j).$$

Note that the existence of a containment mapping from $\pi_j(u_j,v_j)$ to θ^w is immediate from the construction.

For the case when $p_\ell < p_0$, the construction is analogous, except that in this case we define a word τ that can be $[p_0,p_\ell]$-folded into w, since we start in a final state in C_ℓ and finish with the initial state in C_0.

After the completion of this procedure we have created an expansion for each of the atoms of ρ, and shown that there is a containment mapping from each of these to θ^w. Since the images of the variables that are shared by two π_j and $\pi_{j'}$, $j \neq j'$, are given by the positions in which the cuts mark a variable the first time, and this can only happen once, we have that the expansion of Γ given by the union of the expansions of the atoms can be mapped to θ^w as well. Moreover, since C marks x and C' marks y, we have that this mapping sends x to z_p and y to $z_{p'}$, as required. This finishes the base case.

The case when Γ is an arbitrary flat nUC2RPQ follows along the same lines than the base case. Assume again that $C = (\rho, \mathit{Inc}, \mathit{Mark}, \mathit{States})$ and $C' = (\rho, \mathit{Inc}', \mathit{Mark}', \mathit{States}')$ in $\mathrm{C}\mathit{uts}(\Gamma)$, where ρ is one of the rules in Γ, of the form

$$\mathit{Ans}(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m),$$

where P_j may now be either an NFA or an expression P^+.

Let $\mathcal{S}$ be the sequence of configurations that witnesses the accepting run of (C, p) and (C', p'). Again, for a given j such that $1 \leq j \leq m$, we show how to define the expansion π_j, corresponding to the part of the expansion that relates to $P_j(u_j, v_j)$. If P_j is a 2RPQ, we define π_j exactly as in the base case. Thus we assume that P_j is an expression of the form P^+. As before, let (C_0, p_0) be the configuration in $\mathcal{S}$ such that C_0 marks for the first time the variable u_j. Likewise, define (C_ℓ, p_ℓ) with respect to v_j. We again have two cases, depending on p_0 and p_ℓ. Since they are analogous we only show the case when $p_0 \leq p_\ell$.

We shall define an expansion π_j of $P^+(u_j, v_j)$ and a containment mapping μ_j from π_j to θ^w such that $\mu_j(u_j) = z_{p_0}$ and $\mu_j(v_j) = z_{p_\ell}$. If t is a configuration, then $t[1]$ and $t[2]$ denote its associated cut and position, respectively. It suffices to define a sequence $c_1, \ldots, c_{2n}$ of configurations of $T_{(\Gamma_P, w)}$ such that (i) $c_1[2] = p_0$ and $c_n[2] = p_\ell$, (ii) for each $1 \leq i \leq n$, $c_{2i-1}[1]$ marks u_j and $c_{2i}[1]$ marks v_j, (iii) for each $1 \leq i \leq n-1$, $c_{2i}[2] = c_{2i+1}[2]$, and (iv) for each $1 \leq i \leq n$, c_{2i-1} and c_{2i} define an accepting run for $\Gamma_P(u_j, v_j)$. Indeed, if such a sequence exists, then by inductive hypothesis we have expansions $\phi_1, \ldots, \phi_n$ of $\Gamma_P(u_j, v_j)$ and containment mappings $\nu_1, \ldots, \nu_n$ from theses expansions into θ^w that can be merged to define the expansion π_j of $P^+(u_j, v_j)$ and the containment mapping μ_j into θ^w. As in the base case, we can merge all these π_js and μ_js to produce our required expansion of Γ and the containment mapping into θ^w.

Let $s_j^0, \ldots, s_j^\ell$ be the sequence corresponding to the j-th element of *States* in each of the cuts $C_0, \ldots, C_\ell$ between C_0 and C_ℓ in the sequence $\mathcal{S}$, and $q_0 \leq q_1 \leq \cdots \leq q_\ell$ be its corresponding positions ($q_0 = p_0$ and $q_\ell = p_\ell$). The sequence $c_1, \ldots, c_{2n}$ is defined as follows. By definition of $\Rightarrow_{(\Gamma, w)}$ (condition D.3), s_j^0 is an initial cut for $\Gamma_P(u_j, v_j)$ at position p_0. If this is because of condition (1) of initial cut, then s_j^0 marks u_j and there is an initial cut s_I such that $(s_I, 0) \Rightarrow^*_{(\Gamma_P, w)} (s_j^0, p_0)$. In this case we let $c_1 = (s_j^0, p_0)$ and call c_1 the *current* configuration. If condition (2) holds, then we have positions $r_0, \ldots, r_f$ and cuts $c^x, c_0^y, c_0^x, c_1^y, c_1^x, , \ldots, c_f^y, c_f^x$ such that

- $r_f < p_0$,
- (c^x, p_0) and (c_0^y, r_0) define an accepting run for $\Gamma_P(u_j, v_j)$ over w,

- For each $0 \leq i \leq f-1$, (c_i^x, r_i) and (c_{i+1}^y, r_{i+1}) define an accepting run for $\Gamma_P(u_j, v_j)$ over w, and
- c_f^x marks u_j and there is an initial cut c_I of Γ_P such that

$$(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} (c_f^x, r_f) \Rightarrow^*_{(\Gamma_P,w)} (s_j^0, p_0)$$

In this case we let $c_1 = (c^x, p_0)$, $c_2 = (c_0^y, r_0)$, $c_3 = (c_0^x, r_0)$, $c_4 = (c_0^y, r_1)$, and so on, until we define $c_h = (c_f^x, r_f)$, which becomes the current configuration. Note that in any case, for the current configuration c, we have that $c[1]$ marks u_j, $(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} c \Rightarrow^*_{(\Gamma_P,w)} (s_j^0, p_0)$, for some initial cut c_I.

Let g be the greatest position in $\{0, \ldots, \ell\}$ such that $(s_j^0, p_0) \Rightarrow^*_{(\Gamma_P,w)} (s_j^g, q_g)$. By definition of $\Rightarrow_{(\Gamma,w)}$, either $(s_j^g, q_g) = (s_j^\ell, p_\ell)$ or (s_j^g, q_g) and (s_j^{g+1}, q_g) define a transitive run for $\Gamma_P(u_j, v_j)$ (condition D.7). If the latter case holds, we continue the construction of $c_1, \ldots, c_{2n}$ as follows. If s_j^g is a final cut at position q_g due to condition (1) in the definition of final cuts, and this is witnessed by a cut c^y, then the next pair in the sequence is the current configuration c and (c^y, q_g), and (s_j^{g+1}, q_g) becomes the current configuration. We have that

$$(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} c \Rightarrow^*_{(\Gamma_P,w)} (s_j^0, p_0) \Rightarrow^*_{(\Gamma_P,w)} (s_j^g, q_g) \Rightarrow^*_{(\Gamma_P,w)} (c^y, q_g) \Rightarrow^*_{(\Gamma_P,w)} (c_F, k)$$

for some initial and final cuts c_I and c_F. Thus c and (c^y, q_g) actually define an accepting run.

Suppose now that condition (2) holds. Let $r_0, \ldots, r_f$ and $c_0^y, c_0^x, c_1^y, c_1^x, , \ldots, c_f^y, c_f^x, c^y$ be the positions and cuts witnessing the transitive run of (s_j^g, q_g) and (s_j^{g+1}, q_g), thus we have

- $q_g < r_i$, for each $0 \leq i \leq f$,
- c_0^y marks v_j and there is a final cut c_F of Γ_P such that

$$(s_j^g, q_g) \Rightarrow^*_{(\Gamma_P,w)} (c_0^y, r_0) \Rightarrow^*_{(\Gamma_P,w)} (c_F, k)$$

- For each $0 \leq i \leq f-1$, (c_i^x, r_i) and (c_{i+1}^y, r_{i+1}) define an accepting run for $\Gamma_P(u_j, v_j)$ over w,
- (c_f^x, r_f) and (c^y, q_g) define an accepting run for $\Gamma_P(u_j, v_j)$ over w, and
- s_j^{g+1} marks u_j and there is an initial cut c_I of Γ_P such that $(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} (s_j^{g+1}, q_g)$.

In this case, the next pair in the sequence $c_1, \ldots, c_{2n}$ is the current configuration c and (c_0^y, r_0). Again, it is easy to see that c and (c_0^y, r_0) define an accepting run. The next pairs in the sequence correspond to $(c_0^x, r_0), (c_1^y, r_1)$; $(c_1^x, r_1), (c_2^y, r_2)$; $\cdots$; $(c_f^x, r_f), (c^y, q_g)$. The current configuration again becomes $c = (s_j^{g+1}, q_g)$. Observe that in any case, the current configuration c satisfies that $c[1]$ marks u_j, $(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} c$, for some initial cut c_I. Thus we can repeat this construction by considering the greatest position $\hat{g}$ in $\{g+2, \ldots, \ell\}$ such that $(s_j^{g+1}, q_g) \Rightarrow^*_{(\Gamma_P,w)} (s_j^{\hat{g}}, q_{\hat{g}})$, and so on, until $(s_j^{\hat{g}}, q_{\hat{g}}) = (s_j^\ell, p_\ell)$.

In this situation, we have that the current configuration satisfies $(c_I, 0) \Rightarrow^*_{(\Gamma_P,w)} c \Rightarrow^*_{(\Gamma_P,w)} (s_j^\ell, p_\ell)$, for some initial cut c_I, and $c[1]$ marks u_j. By condition D.5, we have that (s_j^ℓ, p_ℓ) is a final cut for $\Gamma_P(u_j, v_j)$ at position p_ℓ. If this is due to condition (1) of final cuts, then we finish the construction of $c_1, \ldots, c_{2n}$ with the pair c, (c^y, p_ℓ), where c^y is the witness for condition

(1). Otherwise, we have positions $r_0, \ldots, r_f$ and cuts $c_0^y, c_0^x, c_1^y, c_1^x, , \ldots, c_f^y, c_f^x, c^y$ witnessing the fact that (s_j^ℓ, p_ℓ) is final. In this case, we finish the construction of $c_1, \ldots, c_{2n}$ with the pairs $c, (c_0^y, r_0); (c_0^x, r_0), (c_1^y, r_1); \cdots; (c_f^x, r_f), (c^y, p_\ell)$.

($\Longleftarrow$) The proof is again by induction. For the base case, assume that there is an expansion π for a rule ρ of Γ that can be mapped into θ^w. As usual π is of the form

$$\pi(x, y) \leftarrow \pi_1(u_1, v_1), \ldots, \pi_m(u_m, v_m),$$

where for each $1 \leq j \leq m$ we have that $\pi_j(u_j, v_j)$ is a CQ of the form

$$\pi_j(u_j, v_j) \leftarrow r_1(u_j, z_1'), r_2(z_1', z_2'), \ldots, r_{|\tau|}(z_{|\tau|-1}', v_j),$$

and such that $\tau = r_1, \ldots, r_{|\tau|}$ belongs to the language of the NFA P_j of the j-th atom. Now let p_0 and p_ℓ be the positions of w such that z_{p_0} and z_{p_ℓ} are the images of u_j and v_j according to the containment mapping from π_j to θ^w. Now note that τ can be $[p_0, p_\ell]$-folded into w. Again we have two cases depending on which of p_0 or p_ℓ is bigger.

If $p_0 \leq p_\ell$, then we can divide τ into $2\ell - 1$ subwords $\alpha_1, \beta_1, \ldots, \alpha_{\ell-1}, \beta_{\ell-1}, \alpha_\ell$ such that each α_i can be $[p_i, p_i]$-folded into w and each β_i correspond to the symbol w_{p_i}. Furthermore, from the division of τ we divide the run of the NFA for P_j to create a sequence of states $q_0, q_{\alpha_1}, q_{\beta_1}, \ldots, q_{\beta_{\ell-1}}, q_{\alpha_\ell}$: it starts at q_0 and each q_{α_i} and q_{β_i} correspond to the state of the run for τ after reading the corresponding subword. Note, in particular, that q_{α_ℓ} is a final state. If $p_\ell < p_0$ we can proceed in the same way, except our sequence of states would start in a final run and would mimic an inverse run, until it finishes in the initial state.

The sequences of states and the containment mapping from π to θ^w gives us the key to construct a sequence of configurations. The idea is the following:

- Variables u_j and v_j should be included in the positions p_0 and p_ℓ, respectively.
- For every $1 \leq j \leq m$, the advancement of the j-th state of the cuts must follow the sequence $q_0, q_{\alpha_1}, q_{\beta_1}, \ldots, q_{\beta_{\ell-1}}, q_{\alpha_\ell}$.

By coordinating all these requirements together we obtain a sequence of configurations that start in position 0 of w and end in position k. It must be the case that this sequence contains configurations (C, p) and (C', p'), where C marks x and C' marks y. Thus (C, p) and (C', p') define an accepting run for Γ.

For the inductive case we need to account atoms in Γ that are flat nUC2RPQs themselves. Assume that the rule of Γ that gives rise to π is of form

$$Ans(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m),$$

where each P_j is either an NFA or the transitive closure of P. Thus, π contains atoms $\pi_j(u_j, v_j)$ for $1 \leq j \leq m$. Then note that the expansions for atoms $\pi_j(u_j, v_j)$ of π in which $P_j = P^+$ are of the form

$$\pi_j(u_j, v_j) \leftarrow \phi_1(z_0', z_1'), \ldots, \phi_q(z_{q-1}', z_q'),$$

where $z'_0 = u_j$ and $z'_q = v_j$, and each ϕ_i is an expansion for the query $\Gamma_P(u_j, v_j)$. Just as in the base case let us assume that the image for u_j is position p_0, which is smaller than p_ℓ, the image for v_j (the other case is analogous).

It suffices to construct a sequence $(c_1, q_1), \ldots, (c_n, q_n)$ of configurations of $T_{(\Gamma_P, w)}$ with $q_1 \leq \cdots \leq q_n$, such that (i) $q_1 = p_0$ and (c_1, q_1) is an initial cut for $\Gamma_P(u_j, v_j)$ at position p_0, (ii) $q_n = p_\ell$ and (c_n, p_n) is a final cut for $\Gamma_P(u_j, v_j)$ at position p_ℓ, and (iii) for each $1 \leq i \leq n-1$, either $(c_i, q_i) \Rightarrow_{(\Gamma_P, w)} (c_{i+1}, q_{i+1})$, or (c_i, q_i) and (c_{i+1}, q_{i+1}) define a transitive run for $\Gamma_P(u_j, v_j)$. As in the base case, our required accepting run is constructed by synchronizing each of the sequences $(c_1, q_1), \ldots, (c_n, q_n)$, for each $1 \leq j \leq m$.

Let $\eta(z'_0), \ldots, \eta(z'_q)$ be the positions in w where $z'_0, \ldots, z'_q$ are mapped. In particular, $\eta(z'_0) = p_0$ and $\eta(z'_q) = p_\ell$. We assume without loss of generality that all the $\eta(z'_1), \ldots, \eta(z'_{q-1})$ are distinct, and $\eta(z'_i) \notin \{p_0, p_\ell\}$ for each $1 \leq i \leq q-1$. We have that each expansion ϕ_i can be mapped into θ^w. Thus by inductive hypothesis, there are pairs $(c^x_i, \eta(z'_{i-1})), (c^y_i, \eta(z'_i))$, for each $1 \leq i \leq q$, such that $(c^x_i, \eta(z'_{i-1}))$ and $(c^y_i, \eta(z'_i))$ define an accepting run for $\Gamma_P(u_j, v_j)$.

We construct the sequence $(c_1, q_1), \ldots, (c_n, q_n)$ as follows. If $p_0 = p_\ell$ then the sequence is given by the accepting run of (c^x_1, p_0) and (c^y_1, p_0). Thus we assume $p_0 < p_\ell$. If $\eta(z'_i) > p_0$, for each $1 \leq i \leq q$, then we let $e_0 = 0$ and we let $\mathcal{S}_0$ be the sequence witnessing the accepting run of (c^x_1, p_0) and $(c^y_1, \eta(z'_1))$. In this case, we let $(c_1, q_1) = (c^x_1, p_0)$. Otherwise, let e_0 be the greatest position in $\{1 \ldots, q\}$ such that $\eta(z'_{e_0}) < p_0$. In this case, we let $\mathcal{S}_0$ be the sequence witnessing the accepting run of $(c^x_{e_0+1}, \eta(z'_{e_0}))$ and $(c^y_{e_0+1}, \eta(z'_{e_0+1}))$, and let (c_1, q_1) be the last configuration in $\mathcal{S}_0$ of the form (c^*, p_0). It is easy to see that in the first case, condition (1) in the definition of initial cut holds, and in the second case, condition (2) holds. Thus in any case (c_1, q_1) is an initial cut for $\Gamma_P(u_j, v_j)$ at position p_0.

Now let g_1 be the smallest position in $\{\eta(z'_{e_0+1}), \eta(z'_{e_0+2}), \ldots, \eta(z'_q)\}$ and let e_1 be the (unique) position in $\{e_0 + 1, \ldots, q\}$ such that $\eta(z'_{e_1}) = g_1$. Note that $p_0 < g_1 \leq p_\ell$. If $g_1 < p_\ell$, then the next elements in the sequence $(c_1, q_1), \ldots, (c_n, q_n)$ are the configurations in $\mathcal{S}_0$ that appear after (c_1, q_1) until we reach the last configuration in $\mathcal{S}_0$ of the form (c^*, g_1). Since $\eta(z'_i) > g_1$, for each $e_0 + 1 \leq i \leq e_1$, it follows that (c^*, g_1) and $(c^x_{e_1+1}, g_1)$ define a transitive run for $\Gamma_P(u_j, v_j)$. Thus we can define $(c^x_{e_1+1}, g_1)$ as the next element in our sequence.

Let $\mathcal{S}_1$ be the witness of the accepting run of $(c^x_{e_1+1}, g_1)$ and $(c^y_{e_1+1}, \eta(z'_{e_1+1}))$. Let g_2 be the smallest position in $\{\eta(z'_{e_1+1}), \eta(z'_{e_1+2}), \ldots, \eta(z'_q)\}$ and e_2 be the position in $\{e_1 + 1, \ldots, q\}$ with $\eta(z'_{e_2}) = g_2$. Note that $p_0 < g_1 < g_2 \leq p_\ell$. By the previous reasoning, we can continue the construction of $(c_1, q_1), \ldots, (c_n, q_n)$, starting from $(c^x_{e_1+1}, g_1)$, the sequence $\mathcal{S}_1$, g_2 and e_2. We repeat this until $g_i = p_\ell$. At this point, the last defined configuration in the sequence $(c_1, q_1), \ldots, (c_n, q_n)$ is the last configuration in $\mathcal{S}_{i-1}$ of the form $(c^*, g_i) = (c^*, p_\ell)$. It must be the case that c^* is a final cut for $\Gamma_P(u_j, v_j)$ at position p_ℓ. $\square$

5.4.3 Cut Automata

A straightforward idea to continue with the proof is to use the system $T_{(\Gamma,w)}$ to create a deterministic finite automaton that accepts all words that represent the set of linearizations of Γ. However, after a careful analysis one realizes that doing this in a straightforward way results in a much more expensive algorithm, so a little bit of extra work has to be done to avoid additional exponential blowups in our algorithm. In a nutshell, the idea is to extend the alphabet with information about cuts.

Formally, let $\Gamma(x, y)$ be a binary flat nUC2RPQ over Σ. From Σ we construct the extended alphabet $\Sigma(\Gamma)$ as follows.

- Let $\mathcal{A}$ be a 2RPQ, and assume that Q is the set of states of $\mathcal{A}$. Then each symbol of $\Sigma(\mathcal{A})$ consists of a set of pairs of states of $\mathcal{A}$. In other words, $\Sigma(\mathcal{A}) = 2^{Q\times Q}$.
- Let now Γ be the flat nUC2RPQ. Let $\mathcal{I}_{\text{ans}}(\Gamma)$ and $\mathcal{R}_{\text{ans}}(\Gamma)$ be the set of intentional predicates and 2RPQs, respectively, occurring in rules of Γ with the Ans predicate in their heads. Let $\mathrm{C}uts(\Gamma)^2 = \mathrm{C}uts(\Gamma) \times \mathrm{C}uts(\Gamma)$. We define $\mathsf{Same}(\Gamma) = 2^{\mathrm{C}uts(\Gamma)^2}$, $\mathsf{Initial}(\Gamma) = \mathsf{Final}(\Gamma) = 2^{\mathrm{C}uts(\Gamma)} \times 2^{\mathrm{C}uts(\Gamma)}$, $\mathsf{Transitive}(\Gamma) = 2^{\mathrm{C}uts(\Gamma)^2} \times 2^{\mathrm{C}uts(\Gamma)^2}$. Then each symbol of $\Sigma(\Gamma)$ contains an element from $\mathsf{Same}(\Gamma)$, $\mathsf{Initial}(\Gamma)$, $\mathsf{Final}(\Gamma)$ and $\mathsf{Transitive}(\Gamma)$, and for each 2RPQ $\mathcal{A} \in \mathcal{R}_{\text{ans}}(\Gamma)$, a symbol from $\Sigma(\mathcal{A})$ and for each $P \in \mathcal{I}_{\text{ans}}(\Gamma)$, a symbol from $\Sigma(\Gamma_P)$ (recall Γ_P denotes the subquery of Γ, where P is now the answer predicate). In other words,

$$\Sigma(\Gamma) = \mathsf{Same}(\Gamma) \times \mathsf{Initial}(\Gamma) \times \mathsf{Final}(\Gamma) \times \mathsf{Transitive}(\Gamma) \times \mathsf{X}_{\mathcal{A}\in\mathcal{R}_{\text{ans}}(\Gamma)}\Sigma(\mathcal{A}) \times \mathsf{X}_{\mathsf{P}\in\mathcal{I}_{\text{ans}}(\Gamma)}\Sigma(\Gamma_{\mathsf{P}})$$

The following is shown directly from Lemma 5.27.

Lemma 5.30 *Let Γ be a binary flat nU2CRPQ. Then the number of different symbols $|\Sigma(\Gamma)|$ of $\Sigma(\Gamma)$ is exponential in $|\mathrm{C}uts(\Gamma)|$. In particular, $|\Sigma(\Gamma)|$ is double exponential in $|\Gamma|$.*

We now give intuition regarding this construction. Let w be a word from $\Sigma(\Gamma) \cup \Sigma^{\pm}$, of the form $w = u_0a_0u_1 \cdots a_{k-1}u_k$, where the u_is and a_is belongs to $\Sigma(\Gamma)$ and $\Sigma^{\pm}$, respectively. Let $\tau(w) = a_0, \ldots, a_{k-1}$ be the projection of w over $\Sigma^{\pm}$.

For $p \in \{0, \ldots, k\}$, the symbol $u_p \in \Sigma(\Gamma)$ contains, firstly, an element from $\mathsf{Same}(\Gamma)$, that is, a subset of $\mathrm{C}uts(\Gamma) \times \mathrm{C}uts(\Gamma)$, which corresponds to all those pairs (C, C') such that $(C, p) \Rightarrow^*_{(\Gamma,\tau(w))} (C', p)$. The symbol u_p also contains an element from $\mathsf{Initial}(\Gamma)$, which is a pair $(\mathcal{C}, \mathcal{C}')$ of subsets of $\mathrm{C}uts(\Gamma)$. The intuition is that $\mathcal{C}$ contains all the initial cuts for $\Gamma(x, y)$ over $\tau(w)$ at position p. Similarly, $\mathcal{C}'$ contains the initial cuts but for $\Gamma(y, x)$. Analogously, if $(\mathcal{C}, \mathcal{C}')$ is the element in u_p from $\mathsf{Final}(\Gamma)$, then $\mathcal{C}$ (resp. $\mathcal{C}'$) contains all the final cuts for $\Gamma(x, y)$ (resp. $\Gamma(y, x)$) over $\tau(w)$ at position p. Finally, if $(\mathcal{P}, \mathcal{P}') \in \mathsf{Transitive}(\Gamma)$, then $\mathcal{P}$ (resp. $\mathcal{P}'$) contains all those pairs of cuts (C, C') such that (C, p) and (C', p) define a transitive run for $\Gamma(x, y)$ (resp. $\Gamma(y, x)$) over $\tau(w)$.

For each 2RPQ $\mathcal{A} \in \mathcal{R}_{\text{ans}}(\Gamma)$ with set of states Q, the symbol u_p contains a subset of $Q \times Q$ that represents all the pairs (q, q') such that there is a word w' that can be read from q to q'

in $\mathcal{A}$, and that can be $[p,p]$-folded into $\tau(w)$. For each intentional predicate $P \in \mathcal{I}_{\text{ans}}(\Gamma)$, the symbol of $\Sigma(\Gamma_P)$ appearing in u_p satisfies the above conditions but w.r.t Γ_P.

If a word w from $\Sigma(\Gamma) \cup \Sigma^{\pm}$ satisfies all of the above conditions, we say that w has *valid annotations* w.r.t. Γ. We show:

Lemma 5.31 *Let $\Gamma(x,y)$ be a flat nU2CRPQ over Σ. Then there is a NFA $\mathcal{A}^{va}_{\Gamma}$ that accepts a word w over $\Sigma(\Gamma) \cup \Sigma^{\pm}$ if and only if w has* valid annotations *w.r.t. Γ. Furthermore, one can build such a NFA such that its number of states is exponential in $|Cuts(\Gamma)|$.*

Proof: Let $\Gamma(x,y)$ be a binary flat nU2CRPQ over Σ. The automaton $\mathcal{A}^{\text{va}}_{\Gamma}$ that checks that a word $w = u_0 a_0 \cdots u_{k-1} a_{k-1} u_k$ from $\Sigma(\Gamma) \cup \Sigma^{\pm}$ has valid annotations results of the intersection of several automata.

First, it is trivial to construct a NFA A^f that checks that w has the form $u_0 a_0 \cdots u_{k-1} a_{k-1} u_k$, where $k \geq 1$ and the u_is and a_is belong to $\Sigma(\Gamma)$ and $\Sigma^{\pm}$, respectively.

Now, for each 2RPQ $\mathcal{A} \in \mathcal{R}_{\text{ans}}(\Gamma)$, we build an NFA $A_{\mathcal{A}}$ that check that the symbols from $\Sigma(\mathcal{A})$ appearing in $u_0, \ldots, u_k$ are correct. At each position p, the automaton $A_{\mathcal{A}}$ stores two subsets $\mathcal{Q}^{\leftarrow}_p$ and $\mathcal{Q}^{\rightarrow}_p$ of $Q \times Q$:

- $\mathcal{Q}^{\leftarrow}_p$ contains all the pairs (q,q') such that there is a word w' that can be read from q to q' in $\mathcal{A}$, and that can be $[p,p]$-folded into $\tau(w)$ using only positions smaller or equal to p.
- $\mathcal{Q}^{\rightarrow}_p$ contains all the pairs (q,q') such that there is a word w' that can be read from q to q' in $\mathcal{A}$, and that can be $[p,p]$-folded into $\tau(w)$ using only positions greater or equal to p.

Then $A_{\mathcal{A}}$ checks that u_p is consistent with $\mathcal{Q}^{\leftarrow}_p$ and $\mathcal{Q}^{\rightarrow}_p$, that is, the symbol $\mathcal{Q}_p \in \Sigma(\mathcal{A})$ appearing in u_p is exactly the transitive closure of $\mathcal{Q}^{\leftarrow}_p \cup \mathcal{Q}^{\rightarrow}_p$. Note that $\mathcal{Q}^{\leftarrow}_0 = \{(q,q) \mid q \in Q\}$ and that $\mathcal{Q}^{\leftarrow}_{p+1}$ is determined by $\mathcal{Q}^{\leftarrow}_p$ and a_p. Indeed, $\mathcal{Q}^{\leftarrow}_{p+1}$ is the reflexive and transitive closure of all the pairs (q,q') such that there is a pair $(\hat{q},\hat{q}') \in \mathcal{Q}^{\leftarrow}_p$ with $\hat{q} \in \delta_{\mathcal{A}}(q, a_p^-)$ and $q' \in \delta_{\mathcal{A}}(\hat{q}', a_p)$, where $\delta_{\mathcal{A}}$ is the transition function of $\mathcal{A}$. Similarly, $\mathcal{Q}^{\rightarrow}_k = \{(q,q) \mid q \in Q\}$ and $\mathcal{Q}^{\rightarrow}_p$ is determined by $\mathcal{Q}^{\rightarrow}_{p+1}$ and a_p. Then $\mathcal{Q}^{\leftarrow}_p$ and $\mathcal{Q}^{\rightarrow}_p$ can be easily computed by an NFA. Note that the number of states of $A_{\mathcal{A}}$ is exponential with respect to $|Q|$, and thus w.r.t $|\text{C}uts(\Gamma)|$.

To check the correctness of the symbols from $\mathsf{Same}(\Gamma)$, we define an NFA A^{same} as follows. Let p be a position in $\{0, \ldots, k\}$ and (C, C') be cuts of Γ. Suppose that C and C' are defined by a rule ρ and let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and $P_1, \ldots, P_r$ be all the 2RPQs and intentional predicates appearing in ρ. Then it is not hard to see that whether $(C,p) \Rightarrow^*_{(\Gamma, \tau(w))} (C', p)$ is completely determined by the symbols in u_p from $\Sigma(\mathcal{A}_1), \ldots, \Sigma(\mathcal{A}_n), \Sigma(\Gamma_{P_1}), \ldots, \Sigma(\Gamma_{P_r})$. Thus A^{same} can check directly that in each u_p, the symbol from $\mathsf{Same}(\Gamma)$ is consistent with the symbols from $\Sigma(\mathcal{A}_1), \ldots, \Sigma(\mathcal{A}_n), \Sigma(\Gamma_{P_1}), \ldots, \Sigma(\Gamma_{P_r})$.

Now we build an NFA A^{final} that checks the correctness of the symbols from $\mathsf{Final}(\Gamma)$. This automaton is the intersection of two NFAs $A^{\text{final}}_{x,y}$ and $A^{\text{final}}_{y,x}$ that check the first and second

component, respectively, of the symbols from $\mathsf{Final}(\Gamma)$. We only define $A^{\text{final}}_{x,y}$, as the definition of $A^{\text{final}}_{y,x}$ is completely analogous. If $\mathcal{F}_p$ is the first component of the symbol from $\mathsf{Final}(\Gamma)$ in u_p, then $\mathcal{F}_p$ must correspond to the set of final cuts of $\Gamma(x,y)$ over $\tau(w)$ at position p. The NFA $A^{\text{final}}_{x,y}$, after reading u_p, stores the following information:

- The set $\mathcal{E}^{\leftarrow}_p$ of all cuts C for which there is an initial cut C_I of Γ with $(C_I, 0) \Rightarrow^*_{(\Gamma,\tau(w))} (C, p)$.
- The set $\mathcal{E}^{\rightarrow}_p$ of all cuts C for which there is a final cut C_F of Γ with $(C, p) \Rightarrow^*_{(\Gamma,\tau(w))} (C_F, k)$.
- The set $\mathcal{FR}_p$ of all the pairs (C, C') such that (C, p) and (C', p) define a forward run for $\Gamma(x,y)$ over $\tau(w)$. We say that pairs (C, p) and (C', p) define a *forward run* for $\Gamma(x,y)$ over $\tau(w)$, if there is a nonempty sequence $p_0, \ldots, p_\ell$ of positions in $\{0, \ldots, k\}$ and cuts $C^y_0, C^x_0, C^y_1, C^x_1, \ldots, C^y_\ell, C^x_\ell$ such that
  - All the p_is are distinct and $p \leq p_i$, for each $0 \leq i \leq \ell$,
  - C^y_0 marks y and there is a final cut C_F of Γ such that
$$(C, p) \Rightarrow^*_{(\Gamma,w)} (C^y_0, p_0) \Rightarrow^*_{(\Gamma,w)} (C_F, k)$$
  - For each $0 \leq i \leq \ell - 1$, the pairs (C^x_i, p_i) and (C^y_{i+1}, p_{i+1}) define an accepting run for $\Gamma(x,y)$ over $\tau(w)$,
  - C^x_ℓ marks x and there is a final cut C_F of Γ such that
$$(C', p) \Rightarrow^*_{(\Gamma,w)} (C^x_\ell, p_\ell) \Rightarrow^*_{(\Gamma,w)} (C_F, k)$$

$A^{\text{final}}_{x,y}$ computes $\mathcal{E}^{\leftarrow}_p$, $\mathcal{E}^{\rightarrow}_p$ and $\mathcal{FR}_p$ as follows:

- Note that $C \in \mathcal{E}^{\leftarrow}_p$ iff there are cuts C', C'' such that $C' \in \mathcal{E}^{\leftarrow}_{p-1}$, $(C', p-1) \Rightarrow_{(\Gamma,\tau(w))} (C'', p)$ and $(C'', p) \Rightarrow^*_{(\Gamma,\tau(w))} (C, p)$. Thus $\mathcal{E}^{\leftarrow}_p$ is determined by $\mathcal{E}^{\leftarrow}_{p-1}$, u_p and a_{p-1}. Thus $A^{\text{final}}_{x,y}$ can guess the $\mathcal{E}^{\leftarrow}_p$s, starting from $\mathcal{E}^{\leftarrow}_0$, which corresponds to the cuts C for which there is an initial cut C_I of Γ with $(C_I, 0) \Rightarrow^*_{(\Gamma,\tau(w))} (C, 0)$.
- Analogous to $\mathcal{E}^{\leftarrow}_p$, we have that $\mathcal{E}^{\rightarrow}_p$ is determined by $\mathcal{E}^{\rightarrow}_{p+1}$, u_p and a_p. Thus we can guess the $\mathcal{E}^{\rightarrow}_p$s and check that $\mathcal{E}^{\rightarrow}_k$ corresponds to the cuts C for which there is an final cut C_F of Γ with $(C, k) \Rightarrow^*_{(\Gamma,\tau(w))} (C_F, k)$.
- It is not hard to check that $\mathcal{FR}_p$ is determined by $\mathcal{FR}_{p+1}$, u_p, a_p, $\mathcal{E}^{\leftarrow}_p$ and $\mathcal{E}^{\rightarrow}_p$. Thus $A^{\text{final}}_{x,y}$ guesses the $\mathcal{FR}_p$s and checks that $\mathcal{FR}_k$ corresponds to the pairs (C, C') for which there are cuts (D_y, D'_x) such that $(C, k) \Rightarrow^*_{(\Gamma,\tau(w))} (D_y, k)$, $(C', k) \Rightarrow^*_{(\Gamma,\tau(w))} (D'_x, k)$, D_y and D'_x marks y and x respectively, and $D_y, D'_x \in \mathcal{E}^{\rightarrow}_k$.

Note that we obtain an equivalent definition of final cut for $\Gamma(x,y)$ at position p if we impose that all the p_is are distinct. As a consequence, we have that C is a final cut for $\Gamma(x,y)$ at position p, that is, $C \in \mathcal{F}_p$ iff one of the following conditions hold:

1. there is a cut C^y that marks y with $(C, p) \Rightarrow^*_{(\Gamma,\tau(w))} (C^y, p)$ and $C^y \in \mathcal{E}^{\rightarrow}_p$. This corresponds to condition (1) in the definition of final cut.
2. there are pairs (D, D_y) and $(\hat{C}, \hat{C})' \in \mathcal{FR}_{p+1}$ such that $(C, p) \Rightarrow^*_{(\Gamma,\tau(w))} (D, p) \Rightarrow_{(\Gamma,\tau(w))} (\hat{C}, p+1)$, D_y marks y, $D_y \in \mathcal{E}^{\leftarrow}_p$ and $(D_y, p) \Rightarrow_{(\Gamma,\tau(w))} (\hat{C}', p+1)$. This corresponds to condition (2) in the definition of final cut.

It follows that $\mathcal{F}_p$ is determined by $\mathcal{FR}_{p+1}, a_p, u_p, \mathcal{E}_p^{\leftarrow}$ and $\mathcal{E}_p^{\rightarrow}$, thus $A_{x,y}^{\text{final}}$ can check whether the $\mathcal{F}_p$s are correct. Note that the number of states of $A_{x,y}^{\text{final}}$ is exponential in $|\mathrm{C}uts(\Gamma)|$.

It is straightforward to construct from A^{final}, an automaton A^{tr} that checks the correctness of the symbols from $\mathsf{Transitive}(\Gamma)$.

In the case of $\mathsf{Initial}(\Gamma)$, the construction is more involved. Again we focus in the NFA $A_{x,y}^{\text{initial}}$ that checks the correctness of the first component $\mathcal{I}_p$ which corresponds to the initial cuts for $\Gamma(x,y)$ at position p. We say that pairs (C,p) and (C',p) define a *backward run* for $\Gamma(x,y)$ over $\tau(w)$, if there is a nonempty sequence $p_0,\ldots,p_\ell$ of positions in $\{0,\ldots,k\}$ and cuts $C_0^y, C_0^x, C_1^y, C_1^x, \ldots, C_\ell^y, C_\ell^x$ such that

- All the p_is are distinct and $p_i \leq p$, for each $0 \leq i \leq \ell$,
- C_0^y marks y and there is an initial cut C_I of Γ such that

$$(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C_0^y, p_0) \Rightarrow^*_{(\Gamma,w)} (C,p)$$

- For each $0 \leq i \leq \ell - 1$, the pairs (C_i^x, p_i) and (C_{i+1}^y, p_{i+1}) define an accepting run for $\Gamma(x,y)$ over $\tau(w)$,
- C_ℓ^x marks x and there is an initial cut C_I of Γ such that

$$(C_I, 0) \Rightarrow^*_{(\Gamma,w)} (C_\ell^x, p_\ell) \Rightarrow^*_{(\Gamma,w)} (C',p)$$

We denote by $\mathcal{BR}_p$ the set of pairs (C,C') such that the pairs (C,p) and (C',p) define a backward run. Observe that we obtain an equivalent definition of initial cut at position p if we additionally impose that all the p_is are distinct and $p_i \neq p$, for each $0 \leq i \leq \ell$. From this observation, we have that C is an initial cut of $\Gamma(x,y)$ at position p, that is, $C \in \mathcal{I}_p$ iff one of the following conditions hold:

1. C marks x and $C \in \mathcal{E}_p^{\leftarrow}$. This corresponds to condition (1) in the definition of initial cut.
2. there is a cut C^x, pairs (C_1,C_2), (C_3,C_4), $\ldots$, (C_{2h-1},C_{2h}) from $\mathcal{FR}_{p+1}$ and pairs (D_1,D_2), (D_3,D_4), $\ldots$, (D_{2h-1},D_{2h}) from $\mathcal{BR}_{p-1}$, for $h \geq 1$ such that (i) C^x marks x, $C^x \in \mathcal{E}_p^{\leftarrow}$ and $(C^x,p) \Rightarrow^*_{(\Gamma,\tau(w))} (C_1,p+1)$, (ii) $(D_{2i-1},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C_{2i},p+1)$, for each $1 \leq i \leq h$, (iii) $(D_{2i},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C_{2i+1},p+1)$, for each $1 \leq i \leq h-1$, (vi) $(D_{2h},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C,p)$. This corresponds to condition (2) when $p_0 > p$.
3. there is a cut C^x, pairs $(C_1,C_2),(C_3,C_4),\ldots,(C_{2h-1},C_{2h})$ from $\mathcal{FR}_{p+1}$ and pairs (D_1,D_2), $(D_3,D_4),\ldots,(D_{2h+1},D_{2h+2})$ from $\mathcal{BR}_{p-1}$, for $h \geq 0$ such that (i) C^x marks x, $C^x \in \mathcal{E}_p^{\rightarrow}$ and $(D_1,p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C^x,p)$, (ii) $(D_{2i},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C_{2i-1},p+1)$, for each $1 \leq i \leq h$, (iii) $(D_{2i+1},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C_{2i},p+1)$, for each $1 \leq i \leq h$, (vi) $(D_{2h+2},p-1) \Rightarrow^*_{(\Gamma,\tau(w))} (C,p)$. This corresponds to condition (2) when $p_0 < p$.

From this characterization we have that the correctness of $\mathcal{I}_p$ depends only on $\mathcal{BR}_{p-1}$, $\mathcal{FR}_{p-1}$, a_{p-1}, u_p, a_p, $\mathcal{E}_p^{\leftarrow}$ and $\mathcal{E}_p^{\rightarrow}$. The only missed information is the $\mathcal{BR}_p$s, which can be easily computed using similar ideas as in the case of $\mathcal{FR}_p$. Thus $A_{x,y}^{\text{initial}}$ can check the correctness of $\mathcal{I}_p$ using all this information. Note again that the number of state of A^{initial} is exponential in $|\mathrm{C}uts(\Gamma)|$.

Finally, for each $P \in \mathcal{I}_{\text{ans}}(\Gamma)$, we need to check the correctness of the symbols in $\Sigma(\Gamma_P)$. This is done recursively using the previous ideas.

Then the automaton $\mathcal{A}_\Gamma^{\text{va}}$ is the product of all the NFAs defined above. If we unfold the recursive definition of $\mathcal{A}_\Gamma^{\text{va}}$, we have in this product, for each 2RPQ $\mathcal{A}$ in Γ, an NFA of the form $A_\mathcal{A}$, and for each intensional predicate P in Γ an NFA of the form $A^{\text{same}} \times A^{\text{initial}} \times A^{\text{final}} \times A^{\text{tr}}$ associated with the program Γ_P. Thus $\mathcal{A}_\Gamma^{\text{va}}$ is the product of $O(|\Gamma|) = O(|\text{C}uts(\Gamma)|)$ NFAs, each with a number of states exponential in $|\text{C}uts(\Gamma)|$. It follows that the number of states of $\mathcal{A}_\Gamma^{\text{va}}$ is also exponential in $|\text{C}uts(\Gamma)|$. □

We can finally proceed to build the desired NFA $\mathcal{A}_\Gamma$ that gives the words corresponding to the linearizations of a flat nUC2RPQ Γ. This NFA needs to simulate the system $T_{(\Gamma,w)}$, from an initial cut of Γ to a final cut in which all variables have already been mapped. Of course, we have to check, for every subquery $\Gamma_P(u_j, v_j)$ of Γ, whether this query is indeed satisfied when starting in the position assigned to variable u_j and finishing on the position assigned to v_j. Since we might need to check for more than one such query at any given point, synchronizing all these checks is non-trivial. We do it by relying on the annotations added to words, as explained above.

Lemma 5.32 *Given a binary flat nUC2RPQ $\Gamma(x, y)$ over Σ, one can construct a NFA $\mathcal{A}_\Gamma$ over alphabet $\Sigma(\Gamma) \cup \Sigma^\pm$ such that $\mathcal{A}_\Gamma$ accepts a word w with valid annotations w.r.t. Γ if and only if there are pairs (C, p) and (C', p') that either define an accepting run for $\Gamma(x, y)$ over w, or an accepting run for $\Gamma(y, x)$ over w. Further, the number of states of $\mathcal{A}_\Gamma$ is the size of $Cuts(\Gamma)$.*

Proof: The set of states of $\mathcal{A}_\Gamma$ is simply $\text{C}uts(\Gamma)$. Initial and final states of $\mathcal{A}_\Gamma$ correspond to initial and final cuts, respectively. Let $w = u_0 a_0 u_1 \cdots a_{k-1} u_k$ be a word with valid annotations w.r.t. Γ, and $\tau(w) = a_0 \cdots a_{k-1}$ its projection to $\Sigma^\pm$. Note that existence of a pair (C, p) and (C', p') as in the lemma is equivalent to the existence of a sequence of cuts $C_0, D_0, C_1, D_1, \ldots, C_k, D_k$ such that C_0 is an initial cut, D_k is a final cut, $(C_i, i) \Rightarrow^*_{(\Gamma,\tau(w))} (D_i, i)$, for $0 \leq i \leq k$, and $(D_i, i) \Rightarrow_{(\Gamma,\tau(w))} (C_i, i+1)$, for $0 \leq i \leq k-1$. Observe that whether $(C_i, i) \Rightarrow^*_{(\Gamma,\tau(w))} (D_i, i)$ depends only on the symbol u_i and whether $(D_i, i) \Rightarrow_{(\Gamma,\tau(w))} (C_i, i+1)$ only on the symbol a_i. Thus $\mathcal{A}_\Gamma$ can guess such a sequence while reading w. □

5.4.4 Main Proof

We finally have all the ingredients to show the main result of this section. Now we assume that flat nUC2RPQs are Boolean queries as in Section 5.3.

Theorem 5.33 *The problem of checking whether a single-atom C2RPQ is contained in a flat nUC2RPQ is in* EXPSPACE.

Proof: Let $\gamma \leftarrow \mathcal{A}(y, y')$ be the single-atom C2RPQ, where $\mathcal{A}$ is 2RPQ, and Γ the Boolean flat nUC2RPQ. We proceed as follows:

- We build a binary flat nUC2RPQ $\hat{\Gamma}(x, y)$ from Γ by choosing for each rule in Γ with the *Ans* predicate in its head, two free variables arbitrarily (w.l.o.g. each rule have at least two variables). We build $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ as in Lemma 5.31. The number of states of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ is exponential in $|\mathrm{C}uts(\hat{\Gamma})|$.
- We construct a NFA $\mathcal{A}_\gamma$ that accepts all words over $\Sigma(\hat{\Gamma}) \cup \Sigma^{\pm}$ of the form $u_0\ a_0\ u_1 \cdots a_{k-1}\ u_k$ where the $a_0 \cdots a_{k-1} \in L(\mathcal{A})$. Clearly the number of states of $\mathcal{A}_\gamma$ is polynomial in γ.
- We build the NFA $\mathcal{A}_{\hat{\Gamma}}$, as in Lemma 5.32. The number of states of $\mathcal{A}_{\hat{\Gamma}}$ is $|\mathrm{C}uts(\hat{\Gamma})|$.

From Lemma 5.32 and 5.29, it follows that the language of $\overline{\mathcal{A}_{\hat{\Gamma}}}$, the complement of $\mathcal{A}_{\hat{\Gamma}}$, intersected with the language of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ is precisely those words w with valid annotations such that its projection $\tau(w)$ over $\Sigma^{\pm}$ is not a linearization of Γ. Thus, if we intersect this language with the one of $\mathcal{A}_\gamma$, we have that the resulting intersection is nonempty if and only if there is an expansion θ of γ that is not a linearization of Γ, i.e., if γ is not contained in Γ.

Thus to check that γ is not contained in Γ, we check that the language of the product NFA of $\mathcal{A}_\gamma$, $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ and $\overline{\mathcal{A}_{\hat{\Gamma}}}$ is not empty. The number of states of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ and $\overline{\mathcal{A}_{\hat{\Gamma}}}$ is single exponential in $|\mathrm{C}uts(\hat{\Gamma})|$, and thus double exponential in $|\Gamma|$. Then we can check non emptiness in Expspace using a standard on-the-fly implementation. □

5.5 Putting it all together: Upper and lower bounds for containment of nUC2RPQs

Theorems 5.33 and 5.25 provide an immediate proof that the containment problem for flat nUC2RPQs is in Expspace. However, we cannot use this result directly to show a 2Expspace upper bound for the containment problem of two nUC2RPQs, as unfolding a nUC2RPQ Γ may result in a flat nUC2RPQ Γ' that is of double-exponential size with respect to $|\Gamma|$, and thus the number of cuts in Γ' might be of triple-exponential size with respect to $|\Gamma|$.

The goal of this section is to show that the containment problem for both RQs and nUC2RPQs is 2Expspace-complete (Theorems 5.7 and 5.8). Section 5.5.1 shows how to refine the constructions in the previous sections to arrive at the desired 2Expspace algorithm, and Section 5.5.2 contains the matching lower bound.

5.5.1 Upper Bound

Recall that the transformation from nUC2RPQs to flat nUC2RPQs creates queries that may be of double exponential size with respect to the original queries, but whose widths and

depths are bounded exponentially and polynomially, respectively (Proposition 5.11). The idea is then to redo the results from the previous section but now bounding the algorithms in terms of the width and depth of queries.

Lemma 5.34 *Let Γ be a binary flat nUC2RPQ with depth d, height h, width w and weight g. Then the size of Cuts(Γ) is at most $(2hg)^{O(w^{d+1})}$.*

Proof: Recall that the depth of a flat nUC2RPQ is the maximum length of a directed path from a 2RPQ to the *Ans* predicate in its dependence graph, minus 1; the height is the maximum size of $\mathsf{rules}(S)$ over all its intensional predicates; the width is the maximum number of atoms in a rule body; and the weight is the maximum size of a 2RPQ appearing in any rule.

Below we bound the number of cuts of a flat nUC2RPQ in terms of these four variables. Let us first count the number of cuts for queries of depth 0. Let Γ be such a query, and assume its height, width and weight is h, w and g, respectively.

Consider first a rule $\rho \in \mathsf{rules}(Ans)$ of the form

$$Ans(x, y) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m),$$

where $m \leq w$. The number of cuts for Γ of the form $(\rho, Inc, Mark, States)$ corresponds to the number of choices for *Inc*, *Mark* and *States* for this given rule. In this case the number of subsets *Inc* is at most $2^{(2w)}$, and likewise for *Mark*. Regarding *States*, we have at most w different NFAs and the number of states in each of these is bounded by g, so the total number of different tuples is bounded by g^w. Thus for each rule $\rho \in \mathsf{rules}(Ans)$, the number of cuts associated to ρ is bounded by $2^{2w}2^{2w}g^w$. There are at most h rules in Γ, so the total number of cuts is bounded by $h2^{2w}2^{2w}g^w$ and thus by $(2hg)^{4w}$.

For queries of depth 1, we can get almost the same bounds, except this time the size of *States* is not bounded by g, but by the number of cuts of the subqueries in the rules of Γ, which we know is bounded by $(2hg)^{4w}$. In total this gives $h2^{2w}2^{2w}((2hg)^{4w})^w$, which is bounded by $(2hg)^{4w+4w^2}$. We can follow this idea and apply an inductive argument to show that the number of cuts of Γ is bounded by $(2hg)^{4w+4w^2+\cdots 4w^{d+1}}$, which is $(2hg)^{O(w^{d+1})}$. □

Having this lemma we can now state the 2Expspace algorithm for checking containment of nUC2RPQs (or RQs):

1. Given nUC2RPQs Ω and Ω', we unfold these queries to construct flat nUC2RPQs Γ and Γ'. By Proposition 5.11, we have that (i) $|\Gamma'|$ is doubly-exponential in $|\Omega'|$, (ii) the width of Γ' is single-exponential in $|\Omega'|$, and (iii) the depth of Γ' is polynomial in $|\Omega'|$. Similarly for Γ and Ω.
2. We construct from Γ and Γ' the single-atom C2RPQ $\tilde{\gamma}$ and the flat nUC2RPQ $\tilde{\Gamma}$ defined in the reduction from Section 5.3. This is a polynomial-time reduction, thus basically all the bounds from (1) still hold: $|\tilde{\gamma}|$ is doubly-exponential in $|\Omega|$; $|\tilde{\Gamma}|$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$ (note that $\tilde{\Gamma}$ now depends on Ω too); the width of $\tilde{\Gamma}$ is single-exponential in $|\Omega'|$; the depth of $\tilde{\Gamma}$ is polynomial in $|\Omega'|$.

3. Using Lemma 5.34 we obtain that the number of cuts of the binary flat nUC2RPQ $\hat{\Gamma}$ associated with $\tilde{\Gamma}$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$. It follows that the number of states of the automata $\mathcal{A}_{\tilde{\gamma}}$, $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ and $\overline{\mathcal{A}_{\hat{\Gamma}}}$ from Theorem 5.33 is triple-exponential in $|\Omega|$ and $|\Omega'|$. To decide whether $\Omega \subseteq \Omega'$, we check the intersection of these automata for nonemptiness, which can be done in 2Expspace.

5.5.2 Lower Bound

By combining techniques from [44, 33], we can show a matching lower bound for containment of nUC2RPQs, and conclude Theorems 5.8 and 5.7.

Theorem 5.35 *The containment problem for nUC2RPQs is* 2Expspace*-hard.*

Proof: The proof is based on ideas from [44]. We reduce from the following 2Expspace-complete problem: Given a deterministic Turing machine M and a positive integer n, decide whether M accepts the empty tape using 2^{2^n} space. A configuration of M can be described by a word of length 2^{2^n}. The *symbols* of the word are either symbols of the alphabet or *composite* symbols. A composite symbol is a pair (s, a), where s is a state of M and a is in Ms alphabet. Intuitively, a symbol (s, a) indicates that M is in state s and is scanning the symbol a. It is well-known that the successor relation between configurations depends only in local constraints: we can associate with the transition function δ two ternary relations I_M, F_M and a 4-ary relation B_M on symbols that characterizes the successor relation. If $\bar{a} = a_1 \cdots a_m$ and $\bar{b} = b_1, \cdots, b_m$ are two configurations, then $\bar{b}$ is a successor of $\bar{a}$ iff $(a_1, a_2, b_1) \in I_M$, $(a_{m-1}, a_m, b_m) \in F_M$ and $(a_{i-1}, a_i, a_{i+1}, b_i) \in B_M$, for each $1 < i < m$.

We construct two nUC2RPQs Γ and Γ' that encode accepting computations of M. An expansion of Γ represents a sequence of configurations. The role of Γ' is to detect *errors* that prevent this sequence from being an accepting computation. Thus, accepting computations are identified with expansion of Γ without errors, that is, such that Γ' cannot be mapped to it. Hence, we shall have that M accepts the empty tape iff Γ is not contained in Γ'. This implies that the containment problem is 2Expspace-hard.

To detect errors, we need to compare corresponding positions in successive configurations. To do this, we address each position with a 2^n-bit address. Thus, each position in a configuration will be encoded by 2^n rule unfoldings. As in [44], we encode *carry* bits in addition to address bits, so the successor relation becomes local. If $\bar{a} = a_1 \cdots a_{2^n}$ and $\bar{b} = b_1 \cdots b_{2^n}$ are two 2^n-bit address, then $\bar{b} = \bar{a} + 1$ iff there is a 2^n-carry bit $\bar{c} = c_1 \cdots c_{2^n}$ such that (i) $c_{2^n} = 1$, (ii) $c_i = 1$ iff $a_{i+1} = 1$, $c_{i+1} = 1$, for $1 \leq i < 2^n$, and (iii) $b_i = 0$ iff $a_i = c_i$, for $1 \leq i \leq 2^n$.

Now we define Γ. We have extensional predicates `E,$, Start, IsAddress, IsSymbol, Zero, One, Carry0` and `Carry1`. For each configuration symbol a, we also have an extensional predicate $\mathtt{Q}_a$. The intensional predicates are *Bit*, *ConfAddress*, *ConfSymbol*, *Comp* and

Final. The query is Boolean, so *Ans* is a 0-ary predicate. We have rules

$$\begin{aligned}
Bit(x,y) &\leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), \texttt{Carry0}(x,x), \texttt{E}(x,y).\\
Bit(x,y) &\leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), \texttt{Carry1}(x,x), \texttt{E}(x,y).\\
Bit(x,y) &\leftarrow \texttt{IsAddress}(x,x), \texttt{One}(x,x), \texttt{Carry0}(x,x), \texttt{E}(x,y).\\
Bit(x,y) &\leftarrow \texttt{IsAddress}(x,x), \texttt{One}(x,x), \texttt{Carry1}(x,x), \texttt{E}(x,y).\\
ConfAddress(x,y) &\leftarrow Bit^+(x,y).
\end{aligned}$$

Above, the variable x represents an address position. We consider the four possible values for the address bit and carry bit. The atom $\texttt{E}(x,y)$ connects adjacent positions. We also have rules

$$\begin{aligned}
ConfSymbol(x,y) &\leftarrow ConfAddress(x,z), \texttt{IsSymbol}(z,z), \texttt{Q}_a(z,z), \texttt{E}(z,y).\\
ConfSymbol(x,y) &\leftarrow ConfAddress(x,z), \texttt{IsSymbol}(z,z), \texttt{Q}_a(z,z), \$(z,y).\\
&\qquad \text{(for each symbol } a)\\
Comp(x,y) &\leftarrow ConfSymbol^+(x,y).
\end{aligned}$$

The predicate *ConfSymbol* describes an address, followed by a symbol a. The atom $\texttt{E}(z,y)$ connects symbols in the same configuration. The atom $\$(z,y)$ connects symbols in successive configurations. The predicate *Comp* encodes sequences of "blocks" of the form address-symbol. To encode the end of the computation, we put in Γ rules of the form

$$Final(x,y) \leftarrow ConfAddress(x,y), \texttt{IsSymbol}(y,y), \texttt{Q}_a(y,y).$$

for symbols a of the form $a = (s, a')$, where s is an accepting state. Finally, to encode a computation we use the following rule

$$Ans \leftarrow \texttt{Start}(x,x), Comp(x,z), Final(z,y).$$

The intuition is that each expansion of Γ corresponds to a potential accepting computation of M, that is, a sequence of address-symbol blocks, ending in an address-accepting symbol block.

Now we construct Γ' to detect errors in expansions of Γ. For each $0 \leq i \leq n$, we have intensional predicates $dist_i, dist_{\leq i}, dist_{<i}$ and $equal_i$. For each $0 < i \leq n$, we have a rule $dist_i(x,y) \leftarrow dist_{i-1}(x,z), dist_{i-1}(z,y)$. We also have rules $dist_0(x,y) \leftarrow \texttt{E}(x,y)$ and $dist_0(x,y) \leftarrow \$(x,y)$. Clearly, the predicate $dist_i(x,y)$ holds precisely when there is a path of length 2^i from x to y, consisting of E-labeled or \$-labeled edges.

For each $0 < i \leq n$, we have two rules

$$\begin{aligned}
dist_{\leq i}(x,y) &\leftarrow dist_{\leq i-1}(x,z), dist_{\leq i-1}(z,y).\\
dist_{<i}(x,y) &\leftarrow dist_{<i-1}(x,z), dist_{\leq i-1}(z,y).
\end{aligned}$$

We also have rules $dist_{\leq 0}(x,y) \leftarrow \texttt{E}(x,y)$, $dist_{\leq 0}(x,y) \leftarrow \$(x,y)$, $dist_{\leq 0}(x,x) \leftarrow \mathsf{true}$ and $dist_{<0}(x,x) \leftarrow \mathsf{true}$. Here, $dist_{\leq i}$ holds precisely when there is a path of length at most 2^i from x to y, and $dist_{<i}(x,y)$ holds precisely when there is a path of length at most $2^i - 1$ from x to y (again, the paths consist of E-labeled or \$-labeled edges). Rules of the form $S(x,x) \leftarrow \mathsf{true}$ can be easily simulated by Datalog rules.

Now we define the $equal_i$ predicates. For each $0 < i \leq n$, we have rules

$$equal_i(x,y) \leftarrow equal_{i-1}(x,y), equal_{i-1}(x',y'), dist_{i-1}(x,x'), dist_{i-1}(y,y').$$
$$equal_0(x,y) \leftarrow \#_1(x,x'), \#_2(y,y'), \texttt{?}(x,x), \texttt{?}(y,y).$$
$$\text{(for each } \#_1, \#_2 \in \{\texttt{E}, \texttt{\$}\} \text{ and } \texttt{?} \in \{\texttt{Zero}, \texttt{One}\})$$

We are only interested in the behavior of $equal_i$ over expansions of Γ. If an atom of the form $S(x,x)$ appears in an expansion, we say that the variable x is *labeled* with S. Hence, expansions of Γ are basically directed paths whose edges are labeled by `E` or `$`, and whose variables (or nodes) are labeled with symbols in

$$\{\texttt{Start, IsAddress, IsSymbol, Zero, One, Carry0, Carry1}\} \cup \{\texttt{Q}_a \mid \text{ for each symbol } a\}$$

It is easy to see that in such a model, $equal_i(x,y)$ holds precisely when the directed paths of length 2^i starting at x and y have the same variable labels, with the possible exception of the last variable.

To detect errors and "filter out" expansions of Γ, we use ideas from [44]. First, we need to verify that the first block of the expansion corresponds to an address of length 2^n followed by a symbol. We do this by putting in Γ' the rules

$$Ans \leftarrow \texttt{Start}(x,x), dist_{<n}(x,y), \texttt{\$}(y,z).$$
$$Ans \leftarrow \texttt{Start}(x,x), dist_{<n}(x,y), \texttt{IsSymbol}(y,y).$$
$$Ans \leftarrow \texttt{Start}(x,x), dist_{n}(x,y), \texttt{IsAddress}(y,y).$$

The first rule finds expansions where one of the first 2^n edges is a `$`-labeled edge. The second rule finds expansions where one of the first 2^n variables is a *symbol* variable, that is, a variable labeled with `IsSymbol`. The last rule detects expansions where the (2^n+1)-th variable is an *address* variable, that is, a variable labeled with `IsAddress`. For each $\texttt{?} \in \{\texttt{E}, \texttt{\$}\}$, we add rules

$$Ans \leftarrow \texttt{IsSymbol}(x,x), \texttt{?}(x,y), dist_{<n}(y,z), \texttt{\$}(z,z').$$
$$Ans \leftarrow \texttt{IsSymbol}(x,x), \texttt{?}(x,y), dist_{<n}(y,z), \texttt{IsSymbol}(z,z).$$
$$Ans \leftarrow \texttt{IsSymbol}(x,x), \texttt{?}(x,y), dist_{n}(y,z), \texttt{IsAddress}(z,z).$$

The first rule find expansions where one of the first 2^n+1 (except by the first one) edges, after a symbol variable, is a `$`-labeled edge. The second rule find expansions where one of the first 2^n variable, after a symbol variable, is a symbol variable. The two last rules find expansions where the (2^n+1)-th variable, after a symbol variable, is an address variable.

So far we have ensured that we have filtered out all expansions that do not correspond to sequences of blocks of 2^n address variables followed by a symbol variable. Now, we need to check that the address bits indeed act as 2^n-bit counter. That is, the first address is $0,\ldots,0$ and two adjacent addresses are successive. For example, a possible error is that the first address is not $0,\ldots,0$. Such an error can be found by the following rule

$$Ans \leftarrow \texttt{Start}(x,x), dist_{<n}(x,y), \texttt{One}(y,y).$$

Another possible error is when the i-th carry bit is 0, but the $(i+1)$-th carry and address bit are 1. This can be detected by the rule

$$Ans \leftarrow \texttt{IsAddress}(x,x), \texttt{E}(x,y), \texttt{Carry0}(x,x), \texttt{One}(y,y), \texttt{Carry1}(y,y).$$

A more interesting case is when the i-th carry and address bit are the same but the i-th address bit in the next address is 1, instead of 0. This is detected by the following rules

$$Ans \leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), \texttt{Carry0}(x,x), dist_n(x,y), \texttt{E}(y,z), \texttt{One}(z,z).$$
$$Ans \leftarrow \texttt{IsAddress}(x,x), \texttt{One}(x,x), \texttt{Carry1}(x,x), dist_n(x,y), \texttt{E}(y,z), \texttt{One}(z,z).$$

Note that corresponding address variables in successive addresses are exactly at distance $2^n + 1$. All other errors can be easily detected by similar rules.

We now have to ensure that every sequence of addresses starting with $0, \ldots, 0$ describe a single configuration, that is, configurations change exactly when the address is $1, \ldots, 1$. Thus, there are two types of error here: (1) a configuration changes when the address is not $1, \ldots, 1$, or (2) a configuration does not change when the address is $1, \ldots, 1$. Recall that changes of configuration are detected by the symbol \$. Errors of type (1) can be detected by the rule

$$Ans \leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), dist_{\leq n}(x,y), \texttt{IsSymbol}(y,y), \$(y,z).$$

To detect errors of type (2), we need to introduce new intensional predicates $AllOnes_i$, for each $0 \leq i \leq n$, such that $AllOnes_i(x,y)$ holds precisely when there is a directed path from x to y of length 2^i such that all the variables in the path are labeled with `One`, with the possible exception of the last variable y. These predicates can be defined as follows

$$AllOnes_i(x,y) \leftarrow AllOnes_{i-1}(x,z), AllOnes_{i-1}(z,y). \qquad \text{(for each } 0 < i \leq n)$$
$$AllOnes_0(x,y) \leftarrow \texttt{E}(x,y), \texttt{One}(x,x).$$

Then errors of type (2) can be detected with $Ans \leftarrow AllOnes_n(x,y), \texttt{E}(y,z)$.

We have ensured so far that we have a sequence of configurations of length 2^{2^n} with the proper sequence of addresses. We now have to ensure that this sequence of configurations indeed represents a legal computation of the machine M. In order to do this, we need to introduce new intensional predicates *SameConf* and *NextConf*. Intuitively, $SameConf(x,y)$ holds exactly when x and y are variables in the same configuration. Similarly, $NextConf(x,y)$ holds exactly when x and y are in adjacent configurations. These predicates can be defined as follows

$$SameConf(x,y) \leftarrow \texttt{E}^+(x,y).$$
$$NextConf(x,y) \leftarrow SameConf(x,z), \$(z,z'), SameConf(z',y).$$

Now we can verify that the first configuration is actually the initial configuration. Suppose that $\perp$ corresponds to the *blank* symbol and s_0 to the initial state, so the initial configuration is $(s_0, \perp) \cdot \perp^{2^{2^n}-1}$. Then, we can use the following rules

$$Ans \leftarrow \texttt{Start}(x,x), dist_n(x,y), \texttt{Q}_a(y,y). \quad \text{(for each symbol } a \neq (s_0, \perp))$$
$$Ans \leftarrow \texttt{Start}(x,x), dist_n(x,y), SameConf(y,z), \texttt{IsSymbol}(z,z), \texttt{Q}_a(z,z).$$
(for each symbol $a \neq \perp$)

Finally, we have to detect errors between corresponding symbols in two successive configurations, that is, when such symbols do not obey the restrictions imposed by the relations I_M, F_M and B_M. For example, a violation of B_M, that is, a tuple $(a, b, c, d) \notin B_M$, can be found by rules in Γ' of the form

$$\begin{aligned} Ans \leftarrow\ & \mathtt{Q}_a(x_1, x_1), \mathtt{E}(x_1, z_2), dist_n(z_2, x_2), \\ & \mathtt{Q}_b(x_2, x_2), \mathtt{E}(x_2, z_3), dist_n(z_3, x_3), \mathtt{Q}_c(x_3, x_3), \\ & dist_n(z, x), \mathtt{Q}_d(x, x), NextConf(z_2, z), equal_n(z_2, z). \end{aligned}$$

Here, the variables x_1, x_2 and x_3 point to three consecutive symbols a, b and c in the same configuration. The variable z_2 points to the beginning of the address preceding x_2. Similarly, x points to the symbol d and z to the beginning of the address preceding x. The atom $NextConf(z_2, z)$ guarantees that a, b, c and d appears in successive configurations, and $equal_n(z_2, z)$ guarantees that the addresses starting at z_2 and z are the same, so b and d appears in corresponding positions. We add these rules for each $(a, b, c, d) \notin B_M$. We can easily define rules that detect violations of the relations I_M and F_M. Finally, observe that the construction of Γ and Γ' can be carried out in polynomial time in n and the size of M. □

5.6 Restrictions and variants of RQs

In this section we analyze some variants of RQs and study them in terms of the complexity of evaluation and containment. We begin with two different restrictions on RQs: bounded treewidth and bounded depth. We see how the restriction on queries of bounded treewidth results in better query evaluation properties, while restricting the depth of queries provide better bounds for containment. We then study how to lift our results to show that the same complexity bounds apply for the containment of RQs with predicates of unbounded arity, although this generalization does carry an impact on query evaluation.

Bounded treewidth RQs It is well-known that (U)CQs of bounded treewidth can be evaluated in polynomial time (see Section 4.1). This tractability result can be easily extended to bounded treewidth (U)C2RPQs, see beginning of Section 4.2. We show this good behavior also holds for bounded treewidth RQs.

Let ρ be an extended Datalog rule $S(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$. The *underlying* CQ of ρ is the Boolean CQ over the schema $T_1, \ldots, T_m$, where the arity of T_i coincide with $|\bar{y}_i|$, of the form $\theta \leftarrow T_1(\bar{y}_1), \ldots, T_m(\bar{y}_m)$. The *treewidth* of an extended Datalog rule is the treewidth if its underlying CQ. The *treewidth* of an RQ is the maximum treewidth over its rules.

Proposition 5.36 *Let $k \geq 1$ be a positive integer. Then the evaluation problem for RQs of treewidth at most k can be solved in polynomial time.*

Proof: Let Ω be an RQ with treewidth at most k, $\mathcal{G}$ be a database, and $\bar{n}$ be a tuple of nodes in $\mathcal{G}$. We can directly evaluate over $\mathcal{G}$, all intensional predicates in Ω distinct from Ans and

its transitive closures, in a bottom-up fashion. Since intensional predicates are binary and rules have treewidth at most k, this can be done in polynomial time in $|\Omega|$ and $|\mathcal{G}|$. After all intensional predicates distinct from Ans are computed, we can test whether $\bar{n} \in \Omega(\mathcal{G})$ also in polynomial time. □

Note that the queries of the reduction in Theorem 5.5.2 are of bounded treewidth. Thus, although evaluation is more efficient for RQs of bounded treewidth, containment is still 2EXPSPACE-hard in this case.

Bounded depth RQs The *depth* of an RQ is the maximum length of a directed path from an extensional predicate to the Ans predicate in its dependence graph, minus 1.

Proposition 5.37 *Let $d \geq 1$ be a positive integer. Then the containment problem for RQs of depth at most d is in* EXPSPACE.

Proof: We simply use the same algorithm for containment of RQs (Section 5.5.1). A straightforward complexity analysis shows that this algorithm becomes EXPSPACE when the depth is bounded by a constant. □

Note that evaluation is still NP-hard for RQs of bounded depth, as it is hard for CQs, which are RQs of depth 0.

5.6.1 Intensional predicates with unbounded arity: generalized RQs

By definition, each intentional predicate in an RQ, distinct from Ans, has arity 2. If we drop this condition we arrive at what we call *generalized* RQs.

Definition 5.38 (Generalized RQ) *A* generalized *RQ over Σ is a nonrecursive extended Datalog program over Σ.*

Note that intensional predicates in generalized RQs may be of any arity, but the transitive closure of extended rules may only be applied to binary predicates.

Example 5.39 Recall our database of persons and its relationships from Examples 5.2 and 5.3, with relations `knows` and `helps`, abbreviated k and h, respectively. Suppose now that we are interested in cliques of indirect friends: we say that persons p_1, p_2 and p_3 form a potential group whenever all of them are indirect friends to each other. We can compute potential groups using the following RQ:

$$F(x, y) \leftarrow k(x, y), h(x, y).$$
$$Ans(x, y, z) \leftarrow F^+(x, y), F^+(y, x), F^+(x, z), F^+(z, x), F^+(y, z), F^+(z, y).$$

With generalized RQs we can do more: the following query computes all pairs of persons that are linked via a chain of people sharing friends in potential groups (we use G to compute potential groups):

$$\begin{aligned} F(x,y) &\leftarrow k(x,y), h(x,y). \\ G(x,y,z) &\leftarrow F^+(x,y), F^+(y,x), F^+(x,z), F^+(z,x), F^+(y,z), F^+(z,y). \\ C(u,v) &\leftarrow G(u,y,z), G(v,y,z). \\ Ans(x,y) &\leftarrow C^+(x,y). \end{aligned}$$

□

Observe first that each generalized RQ can be unfolded to produce an equivalent RQ, which means that RQs and generalized RQs are equivalent in expressive power. For instance, in the previous example we could have created a rule for predicate C by unfolding each instance of predicate G (and carefully renaming the variables), obtaining a program with just binary relations. Nevertheless, generalized RQs are more succinct than RQs, which may have an impact in the complexity of evaluation and containment.

On the negative side, we show that evaluation of generalized RQs is actually harder that the case of RQs. This is a direct consequence of the fact that generalized RQs subsume nonrecursive Datalog over graph databases, and evaluation for the latter class is known to be Pspace-complete (see e.g. [155]).

Proposition 5.40 *The evaluation problem for generalized RQs is* Pspace*-complete.*

On the positive side, we have that checking containment for generalized RQs is not harder than the case of RQs.

Proposition 5.41 *The containment problem for generalized RQs is* 2Expspace*-complete.*

Proof: Using the same argument as in the case of nUC2RPQs, we have that Proposition 5.11 still applies when we start with a generalized RQ Γ. Then the proposition follows from results in Section 5.5.1. □

We conclude by defining a family of queries that are more succinct than RQs, but that preserves the complexity upper bounds for evaluation and containment. For $r \geq 1$, a *r-generalized* RQ over Σ is a nonrecursive extended Datalog program over Σ, where all intensional predicates, except possibly for Ans, have arity at most r. Note that RQs are contained in the class of 2-generalized RQs. The following proposition is immediate:

Proposition 5.42 *Let $r \geq 1$ be a positive integer.*

- *The evaluation problem for r-generalized RQs is* NP*-complete.*
- *The containment problem for r-generalized RQs is* 2Expspace*-complete.*

5.6.2 Beyond Graph Databases

We can push generalized RQs further to define an analog query language that can be defined over any possible relational schema and not necessarily over schemas with binary relations (graph databases). As our last result, we study nonrecursive extended datalog programs over any possible relational schema. This language is, in a sense, the most general version of RQs that one could think of: both extensional and intentional predicates may have arbitrary arity.

First, observe that the complexity of evaluation does not change: it is PSPACE-complete in general and NP-complete if the arity of the intentional predicates is bounded. Interestingly, we show that containment of generalized RQs over relational databases also does not change, that is, it is still 2EXPSPACE-complete.

Proposition 5.43 *The containment problem for generalized RQs over relational schemas is* 2EXPSPACE*-complete.*

Proof: We reduce containment from the relational case to the graph case. Given two generalized RQs Ω_1 and Ω_2 over a relational schema, we construct in polynomial time generalized RQs Ω'_1 and Ω'_2 over a binary schema such that Ω_1 is contained in Ω_2 iff Ω'_1 is contained in Ω'_2. Then the result follows from Proposition 5.41.

We use an idea from [38] used to reduce containment of Datalog in UCQs from the relational to the graph case. Ω'_1 is obtained from Ω_1 as follows:

- Ω'_1 initially contains all rules of Ω_1.
- For each extensional predicate R of arity $n > 2$ appearing in Ω_1, we define an intensional predicate I_R and n fresh binary extensional predicates $R_1, \ldots, R_n$, which represent the components of R. Then we replace in Ω'_1 each occurrence of R by I_R and add the rule $I_R(x_1, \ldots, x_n) \leftarrow R_1(y, x_1), \ldots, R_n(y, x_n)$, where y is a fresh variable that represents the tuple $(x_1, \ldots, x_n)$.
- For each unary extensional predicate R appearing in Ω_1, we define an intensional predicate I_R and a fresh binary extensional predicate R_u. Then we replace in Ω'_1 each occurrence of R by I_R and add the rule $I_R(x) \leftarrow R_u(x, x)$.

Similarly, we define Ω'_2 from Ω_2. Using the same reasoning as in [38], it follows that Ω_1 is contained in Ω_2 iff Ω'_1 is contained in Ω'_2. □

Part III

Wrapping up

Chapter 6

Conclusions and future work

We have studied the complexity of two fundamental graph query languages. In the first part of this thesis, we considered the class of UC2RPQs and analyzed its complexity in terms of query evaluation. In the second part, we studied the expressive class of RQs and studied the complexity of query containment in depth. Our two main results in this work are the following:

1. Query evaluation for UC2RPQs of bounded treewidth modulo equivalence is fixed-parameter tractable (Theorem 4.9, Section 4.2).
2. Query containment for RQs is 2Expspace-complete (Theorem 5.7, Section 5.2).

Our first result shows that the class of UC2RPQs of bounded treewidth modulo equivalence is well-behaved, just like the case of UCQs. Evaluating these queries takes time $O(f(|\Gamma|) + 2^{|\Gamma|^d} \cdot |\mathcal{G}|^c)$, for constant $d, c \geq 1$ (see the proof of Theorem 4.9). Although the function f is double exponential, the factor multiplying the term that depends on $|\mathcal{G}|$ is only single exponential in $|\Gamma|$, and thus this running may be feasible in practice, especially for small queries Γ. Note that the running time improves when we repeatedly evaluate Γ over constantly changing data, which is very common in database applications. Indeed, the term $f(|\Gamma|)$ is the cost of computing the equivalent query Γ' with treewidth at most $2k+1$ ($k \geq 1$ is fixed), which only depends on Γ and thus it is computed only once. After this step is done, the cost of evaluating Γ over $\mathcal{G}$ is only $O(2^{|\Gamma|^d} \cdot |\mathcal{G}|^c)$.

The second result shows that RQs achieve a good balance between expressiveness and complexity: they are sufficiently expressive to subsume UC2RPQs, they are not harder to evaluate than UC2RPQs, and they have an elementary containment problem. This class also enjoys natural closure properties. Moreover, all generalizations of RQs known to date worsen the complexity of the containment problem to nonelementary or even undecidable. All of these nice properties show that RQs constitute a well-behaved class that deserves further investigation, and an interesting candidate for a general-purpose graph query language.

Our results about RQs can also be translated to the context of relational databases. Since query containment for Datalog is undecidable, much research has focused on finding fragments of Datalog that are expressive enough for applications but have a decidable containment

problem [23, 25, 26, 54, 38, 33, 44, 140]. The class of generalized RQs over relational schemas studied in Section 5.6.2 provides such a natural fragment (see [153] for a recent discussion about generalized RQs). This class consists of all Datalog queries where recursion is used only to express transitive closure, and thus it can express connectivity properties used in practical applications such as declarative networking [124]. Theorem 5.43 shows that this fragment of Datalog has an elementary containment problem.

Of course it is too early to state that our results are directly applicable in practice. For instance, our 2EXPSPACE-completeness result for containment of RQs means that we cannot hope to have a better than triply exponential time upper bound for this problem. However, these are theoretical worst-case complexity bounds, which not always predict algorithmic behavior on real-world instances. It is common to see algorithms that perform well on real-world instances, in spite of pessimistic worst-case complexity bounds (see e.g. [53, 125]). Therefore, careful empirical research is required in order to assess the practical usefulness of our results.

6.1 Future work

We provided complexity results for UC2RPQs and RQs that answer several of the initial question for this research, but there are still many open questions and possible directions for future research. We conclude by briefly naming a few of them:

Complexity issues for UC2RPQs of bounded treewidth modulo equivalence We proved that evaluation of UC2RPQs of bounded treewidth modulo equivalence is fixed-parameter tractable. But is it also polynomial? Tractability of UCQs of bounded treewidth modulo equivalence follows from a sophisticated characterization of the problem in terms of winning strategies in the existential pebble game, but we do not know whether those techniques can be extended to deal with UC2RPQs. Another open question is the complexity of verifying whether a UC2RPQ is of treewidth k modulo equivalence, for a fixed $k \geq 1$. We showed that this is EXPSPACE-complete when $k = 1$, but we do not whether this is decidable for $k > 1$. What makes the difference is that Lemma 4.12 is not tight as Lemma 3.26 (there is a gap between k and $2k + 1$ in the treewidth), and it is not clear how to improve this.

Characterizing fixed-parameter tractablility for UC2RPQs Do UC2RPQs of bounded treewidth equivalence characterize fixed-parameter tractable evaluation of UC2RPQs? As we mentioned in Section 4.1, this is actually the case for UCQs [86]. Unfortunately, the techniques from [86] are especially tailored to work with UCQs and we do not know whether this characterization result holds for UC2RPQs.

Beyond UC2RPQs Instead of working with UC2RPQs, one could also consider the class of *unions of conjunctive nested* 2RPQs (UCN2RPQs), which properly extends the former by including an existential test operator inspired in XPath [16, 135]. Bounded treewidth of UCN2RPQs also leads to tractability, and thus it makes sense to study bounded treewidth modulo equivalence in this extended setting. The problem is relevant since several linear-time

query languages for graph databases are contained in the class of UCN2RPQs but not in the class of UC2RPQs [94, 16].

Enumeration In the evaluation problem, we are given a query q, a database $\mathcal{D}$, and a tuple $\bar{a}$, and we have to decide whether $\bar{a}$ belongs to the answer of q over $\mathcal{D}$. However, in many practical applications it is also useful to *enumerate* the tuples that belong to the answer of q over $\mathcal{D}$. The complexity of this enumeration problem has been studied for several query languages; e.g., (U)CQs [29, 82]. It follows from [82] that the tuples in the evaluation of a semantically acyclic UCQs can be enumerated with *polynomial delay*. This means that there is an algorithm that, given a semantically acyclic UCQ Θ and a database $\mathcal{D}$, enumerates $\Theta(\mathcal{D})$ (without repetitions) in such a way that: (i) the first tuple is produced in polynomial time, and (ii) each subsequent tuple is produced within polynomial time from the previously produced tuple. Notice that this tractability result actually extends Theorem 3.3. An interesting open question is whether we can obtain a similar tractability result for enumerating the answer of semantically acyclic UC2RPQs, or more generally, of UC2RPQs of bounded treewidth modulo equivalence.

CSP for C2RPQs Our work on (U)C2RPQs can be viewed as opening a new line of research in constraint satisfaction. It is well known that there is an intimate connection between CQ evaluation and constraint satisfaction problems [40, 69]. This problem is NP-complete in general, but there is an extensive body of research identifying tractable cases, either by fixing the database and focusing on *expression complexity*, or by studying the *combined complexity* [151] of restricted classes of queries. The same approach of fixing the database or restricting the class of queries can also be applied to the evaluation of C2RPQs. In the constraint satisfaction literature, these are known as *non-uniform* and *structural* restrictions, respectively [85, 28]. For example, in the case of CQs, understanding non-uniform restrictions is related to the well-known CSP Dichotomy Conjecture of Feder and Vardi [69], which has been responsible for a myriad of deep mathematical works [143, 96, 27, 20, 107]. Two interesting examples are the logical [69, 109, 108, 107] and the algebraic approach [30, 102] to constraint satisfaction. It is an interesting line of research to study the applicability of these approaches to the case of C2RPQs.

Containment of Datalog in RQs A possible research direction is to study the containment problem of a Datalog program in an RQ. Containment of Datalog in UC2RPQs was already studied in [38], and it was shown to be EXPTIME-complete. Decidability for the case of RQs again follows from Courcelle's theorem [55, 56]; nevertheless the precise complexity is open. Although it is not clear how to extend the techniques presented in Chapter 5 to containment of Datalog in RQs, we conjecture that this problem is elementary.

RQs and description logics Another open problem is containment of RQs under the presence of description logic constraints. This problem has been studied for several graph query languages, and for most of them the complexity is higher in the constraint case (see e.g. [21, 39, 22]). Following this direction, it would also be interesting to study if the techniques introduced in this paper can be used to study the containment problem for *guarded regular queries* [23], a language that shares a number of similarities with RQs.

Practical issues As we already discussed at the beginning of this section, it is too early to state that our main results can be of practical use. In order to assess the usefulness of our algorithms, it would be interesting to carry out an empirical analysis, and also develop heuristics and optimization techniques for these algorithms, according to the particular application domain.

Bibliography

[1] C. Aberger, S. Tu, K. Olukotun, and C. Ré. EmptyHeaded: A relational engine for graph processing. In *SIGMOD*, 2016.

[2] C. Aberger, S. Tu, K. Olukotun, and C. Ré. Old techniques for new join algorithms: A case study in RDF processing. CoRR, abs/1602.03557, 2016.

[3] S. Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[5] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[6] F. Afrati, M. Joglekar, C. Ré, S. Salihoglu, and J. D. Ullman. GYM: A multiround join algorithm in mapreduce. CoRR, abs/1410.4156, 2014.

[7] K. Amroun, Z. Habbas, and W. Aggoune-Mtalaa. DBToaster: Higher-order delta processing for dynamic, frequently fresh views. *Proceedings of the VLDB Endowment*, 5(10):968–979, 2012.

[8] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys*, 40(1), 2008.

[9] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and Josep Larriba-Pey. Benchmarking database systems for social network applications. In *GRADES*, 2013.

[10] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987.

[11] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[12] P. Barceló. Querying graph databases. In *PODS*, pages 175–188, 2013.

[13] P. Barceló, L. Libkin, A. W. Lin, and P. Wood. Expressive languages for path queries over graph-structured data. *TODS*, 37(4), 2012.

[14] P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries.

In *PODS*, pages 249–260, 2012.

[15] P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries. *SIAM Journal of Computing*, 43(3):1085–1130, 2014.

[16] P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *AMW*, pages 180–195, 2012.

[17] P. Barceló, M. Romero, and M. Y. Vardi. Semantic acyclicity on graph databases. In *PODS*, pages 237–248, 2013.

[18] P. Barceló, M. Romero, and M. Y. Vardi. Does query evaluation tractability help query containment? In *PODS*, pages 188–199, 2014.

[19] P. Barceló, M. Romero, and M. Y. Vardi. Semantic acyclicity on graph databases. *To appear in the SIAM Journal on Computing*, 2016.

[20] L. Barto and M. Kozik. Constraint satisfaction problems solvable by local consistency methods. *Journal of the ACM*, 61(1):1–19, 2014.

[21] Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Nested regular path queries in description logics. In *KR*, 2014.

[22] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Šimkus. Conjunctive regular path queries in lightweight description logics. In *IJCAI*, pages 761–767, 2013.

[23] Meghyn Bienvenu, Magdalena Ortiz, and Mantas Šimkus. Navigational queries based on frontier-guarded datalog: Preliminary results. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, page 162, 2015.

[24] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing,*, 25:1305–1317, 1996.

[25] P. Bourhis, M. Krötzsch, and S. Rudolph. How to best nest regular path queries. In *Description Logics*, 2014.

[26] P. Bourhis, M. Krötzsch, and S. Rudolph. Reasonable highly expressive query languages. In *IJCAI*, pages 2826–2832, 2015.

[27] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

[28] A. Bulatov. On the csp dichotomy conjecture. In *CSR*, pages 331–344, 2011.

[29] A. Bulatov, V. Dalmau, M. Grohe, and D. Marx. Enumerating homomorphisms. *Journal of Computer and System Sciences*, 78(2):638–650, 2012.

[30] A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

[31] P. Buneman. Semistructured data. In *PODS*, pages 117–121, 1997.

[32] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD*, pages 505–516, 1996.

[33] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[34] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing for regular path queries with inverse. In *PODS*, pages 58–66, 2000.

[35] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443–465, 2002.

[36] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query answering and query containment over semistructured data. In *DBLP*, pages 40–61, 2002.

[37] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning of regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[38] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Decidable containment of recursive queries. *Theoretical Computer Science*, 336(1):33–56, 2005.

[39] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.

[40] M. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *STOC*, pages 77–90, 1977.

[41] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

[42] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *ICDE*, pages 190–200, 1995.

[43] S. Chaudhuri and M. Y. Vardi. Optimization of real conjunctive queries. In *PODS*, pages 59–70, 1993.

[44] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *Journal of Computer and System Sciences*, 54(1):61–78, 1997.

[45] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[46] H. Chen. On the complexity of existential positive queries. *TOCL*, 15(1), 2014.

[47] H. Chen. The fine classification of conjunctive queries and parameterized logarithmic

space. *TOCT*, 7(2), 2015.

[48] H. Chen and V. Dalmau. Beyond hypertree width: decomposition methods without decompositions. In *CP*, pages 167–181, 2005.

[49] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[50] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 26(1):64–69, 1983.

[51] M. Consens and A. O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *Hypertext*, pages 269–292, 1989.

[52] M. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *PODS*, pages 404–416, 1990.

[53] B. Cook, A. Podelski, and A. Rybalchenko. Proving program termination. *Communications of the ACM*, 54(5):88–98, 2011.

[54] S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *STOC*, pages 477–490, 1988.

[55] B. Courcelle. The monadic second-order logic of graphs. i. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[56] B. Courcelle. Recursive queries and context-free graph grammars. *Theoretical Computer Science*, 78(1):217–244, 1991.

[57] I. Cruz, A. O. Mendelzon, and P. Wood. A graphical query language supporting recursion. In *SIGMOD*, pages 323–330, 1987.

[58] Cypher. The Cypher query language, Neo4j. https://neo4j.com/developer/cypher-query-language/, 2013.

[59] V. Dalmau, Ph. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP*, pages 310–326, 2002.

[60] Dex. DEX query language, Sparsity Technologies. http://www.sparsity- technologies.com/dex.php, 2013.

[61] R. Diestel. *Graph theory*. Springer, 2010.

[62] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In *Principles of Knowledge Representation, Studies in Logic, Language and Information, CSLI Publications*, pages 193–238, 1996.

[63] R. Downey and M. Fellows. *Parameterized complexity*. Springer-Verlag, 1999.

[64] R. Downey, M. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In *Conf. Discrete Mathematics and Theoretical Computer Science*, pages 194–213, 1996.

[65] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[66] W. Fan. Graph pattern matching revised for social network analysis. In *ICDT*, pages 8–21, 2012.

[67] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Graph pattern matching: from intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[68] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Homomorphism revisited for graph matching. *PVLDB*, 3(1):1161–1172, 2010.

[69] T. Feder and M. Y. Vardi. Monotone monadic snp and constraint satisfaction. In *STOC*, pages 612–622, 1993.

[70] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying integrity constraints on web-sites. In *IJCAI*, pages 614–619, 1999.

[71] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Declarative specification of web sites with Strudel. *VLDB*, 9(1):38–55, 2000.

[72] G. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs. *Information Sciences*, 298:390–406, 2015.

[73] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS*, pages 139–148, 1998.

[74] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6):716–752, 2002.

[75] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer, 2006.

[76] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[77] L. Ghionna, L. Granata, G. Greco, and F. Scarcello. Hypertree decompositions for query optimization. In *ICDE*, pages 36–45, 2007.

[78] L. Ghionna, L., G. Greco, and F. Scarcello. H-DB: A hybrid quantitative-structural SQL optimizer. In *CIKM*, pages 2573–2576, 2011.

[79] G. Gottlob, G. Greco, N. Leone, and F. Scarcello. Hypertree decompositions: Questions and answers. In *PODS*, pages 57–74, 2016.

[80] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries.

Journal of the ACM, 48(3):431–489, 2001.

[81] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[82] G. Greco and F. Scarcello. Structural tractability of enumerating CSP solutions. *Constraints*, 18:38–74, 2013.

[83] M. Grohe. The parameterized complexity of database queries. In *PODS*, pages 82–92, 2001.

[84] M. Grohe. Parameterized complexity for the database theorist. *ACM SIGMOD Record*, 31(4):86–96, 2002.

[85] M. Grohe. The structure of tractable constraint satisfaction problems. In *MFCS*, pages 58–72, 2006.

[86] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):38–74, 2007.

[87] M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *STOC*, pages 89–98, 2014.

[88] M. Grohe and D. Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):1–20, 2014.

[89] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.

[90] A. Gupta and J. D. Ullman. Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In *Workshop on Deductive Databases*, page 195, 1992.

[91] C. Gutierrez, C. Hurtado, A. Mendelzon, and J. Pérez. Foundations of semantic web databases. *Journal of Computer and System Sciences*, 77(3):520–541, 2011.

[92] R. H. Guting. GraphDB: Modeling and querying graphs in databases. In *VLDB*, pages 297–308, 1994.

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

[94] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic logic*. MIT Press, 2000.

[95] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. http://www.w3.org/tr/sparql11-query, 2013.

[96] P. Hell and J. Nešetřil. On the complexity of h-coloring. *J. Comb. Theory*, 48(1):92–110, 1990.

[97] P. Hell and J. Nešetřil. *Graphs and homomorphisms*. Oxford University Press, 2004.

[98] J. Hellings. Conjunctive context-free path queries. In *ICDT*, pages 119–130, 2014.

[99] J. Hellings, B. Kuijpers, J. Van den Bussche, and X. Zhang. Walk logic as a framework for path query languages on graph databases. In *ICDT*, pages 117–128, 2013.

[100] J. E. Hopcroft and J. D. Ullman. *Introduction to automata Theory, languages, and computation*. Addison Wesley, 1979.

[101] InfiniteGraph. Infinitegraph release 3.1 by objectivity inc. http://www.objectivity.com/infinitegraph, 2013.

[102] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

[103] The Apache Jena Manual (2015). http://jena.apache.org, 2015.

[104] W. Kim. Object-oriented databases: definition and research directions. *IEEE Trans. Knowl. Data Eng.*, 2(3):327–341, 1990.

[105] G. Klyne and J. Carroll. Resource description framework (RDF) concepts and abtract syntax. W3C recommendation. http://www.w3.org/tr/2004/rec-rdf-concepts-20040210/, 2004.

[106] Ph. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51:110–134, 1995.

[107] Ph. Kolaitis and M. Y. Vardi. Conjunctive query-containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.

[108] Ph. Kolaitis and M. Y. Vardi. A game-theoretic approach to constraint satisfaction. In *AAAI*, pages 175–181, 2000.

[109] Ph. Kolaitis and M. Y. Vardi. A logical approach to constraint satisfaction. *Lecture Notes in Computer Science*, 5250:125–155, 2008.

[110] A. Koschmieder and U. Leser. Regular path queries on large graphs. In *SSDBM*, pages 177–194, 2012.

[111] E. Kostylev, J. Reutter, and D. Vrgoc. Containment of data graph queries. In *ICDT*, pages 131–142, 2014.

[112] E. Kostylev, J. L. Reutter, M. Romero, and D. Vrgoc. SPARQL with property paths. In *ISWC*, pages 3–18, 2015.

[113] Z. Lacroix, H. Murthy, F. Naumann, and L. Raschid. Links and paths through life sciences data sources. In *DILS*, pages 203–211, 2004.

[114] E. Ladner, R. Lipton, and L. Stockmeyer. Alternating pushdown and stack automata.

SIAM Journal on Computing, 13(1):135–155, 1984.

[115] N. Lehmann and J. Pérez. Implementing graph query languages over compressed data structures: A progress report. In *AMW*, 2015.

[116] U. Leser. A query language for biological networks. *Bioinformatics*, 21(2):33–39, 2005.

[117] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[118] A. Y. Levy and M. Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1984.

[119] A. Y. Levy and M. Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[120] A. Y. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *PODS*, pages 163–173, 1995.

[121] L. Libkin. *Elements of finite model theory*. Springer, 2004.

[122] L. Libkin, W. Martens, and D. Vrgŏc. Querying graph databases with XPath. In *ICDT*, pages 129–140, 2013.

[123] L. Libkin, J. Reutter, and D. Vrgoč. Trial for RDF: adapting graph query languages for RDF data. In *PODS*, pages 201–212, 2013.

[124] B. Loo, T. Condie, M. Garofalakis, D. Gay, J. Hellerstein, P. Maniatis, T. Ramakrishnan, R. Roscoe, and I. Stoica. Declarative networking. *Communications of the ACM*, 52(11):87–95, 2009.

[125] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[126] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM*, 60(6):1–51, 2013.

[127] Neo4j. Neo4j, the graph database. http://www.neo4j.org/, 2013.

[128] M. Nolé and C. Sartiani. A distributed implementation of GXPath. In *EDBT/ICDT Workshops*, 2016.

[129] F. Olken. Graph data management for molecular biology. *Journal of Integrative Biology*, 7(1):75–78, 2003.

[130] Ch. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1986.

[131] Ch. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

[132] J. Paredaens, P. Peelman, and L. Tanca. G-Log: A graph-based query language. *IEEE Trans. Knowl. Data Eng*, 7(3):436–453, 1995.

[133] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In *ISWC*, pages 30–43, 2006.

[134] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):436–453, 2009.

[135] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics*, 8(4):255–270, 2010.

[136] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C recommendation 15 january 2008. http://www.w3.org/tr/rdf-sparql-query/+.

[137] J. L. Reutter, M. Romero, and M. Y. Vardi. Regular queries on graph databases. In *ICDT*, pages 177–194, 2015.

[138] J. L. Reutter, M. Romero, and M. Y. Vardi. Regular queries on graph databases. *To appear in the ACM Transactions on Computer Systems*, 2016.

[139] R. Ronen and O. Shmueli. SoQL: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.

[140] S. Rudolph and M. Krötzsch. Flag & check: Data access with monadically defined queries. In *PODS*, pages 151–162, 2013.

[141] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operator. *Journal of the ACM*, 27(4):633–655, 1980.

[142] M. San Martin, C. Gutierrez, and P. T. Wood. SNQL: A social networks query and transformation language. In *AMW*, 2011.

[143] T. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[144] O. Shmueli. Equivalence of Datalog queries is undecidable. *J. Log. Program*, 15(3):231–241, 1993.

[145] M. Shoaran and A. Thomo. Fault-tolerant computation of distributed regular path queries. *Theoretical Computer Science*, 410(1):62–77, 2009.

[146] M. Sipser. *Introduction to the Theory of Computation*. Course Technology; 3 edition, 2012.

[147] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

[148] D. Surinx, G. Fletcher, M. Gyssens, D. Leinders, J. van den Bussche, D. Van Gucht, S. Vansummeren, and Y. Wu. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL*, 23(5):759–788, 2015.

[149] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test selectivity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13(3):566–579, 1984.

[150] Jeffrey D Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1989.

[151] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.

[152] M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.

[153] M. Y. Vardi. A theory of regular queries. In *PODS*, pages 1–9, 2016.

[154] Open Link Virtuoso (2015). http://virtuoso.openlinksw.com/, 2015.

[155] Sergei Vorobyov and Andrie Voronkov. Complexity of nonrecursive logic programs with complex values. In *PODS*, pages 244–253, 1998.

[156] W3C. Semantic web: The W3C consortiums vision of the web of linked data. http://www.w3.org/standards/semanticweb/, 2013.

[157] P. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

[158] N. Yakovets, P. Godfrey, and J. Gryz. WAVEGUIDE: Evaluating SPARQL property path queries. In *EDBT*, pages 525–528, 2015.

[159] N. Yakovets, P. Godfrey, and J. Gryz. Query planning for evaluating SPARQL property paths. In *SIGMOD Conference*, pages 1875–1889, 2016.

[160] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

[161] M. Yannakakis. Perspectives on database theory. In *FOCS*, pages 224–246, 1995.