Theory Comput Syst
DOI 10.1007/s00224-016-9676-2

Regular Queries on Graph Databases

Juan L. Reutter[1] · **Miguel Romero**[2] · **Moshe Y. Vardi**[3]

Abstract Graph databases are currently one of the most popular paradigms for storing data. One of the key conceptual differences between graph and relational databases is the focus on navigational queries that ask whether some nodes are connected by paths satisfying certain restrictions. This focus has driven the definition of several different query languages and the subsequent study of their fundamental properties. We define the graph query language of *Regular Queries*, which is a natural extension of unions of conjunctive 2-way regular path queries (UC2RPQs) and unions of conjunctive nested 2-way regular path queries (UCN2RPQs). Regular queries allow expressing complex regular patterns between nodes. We formalize regular queries as nonrecursive Datalog programs extended with the *transitive closure* of binary predicates. This language has been previously considered, but its algorithmic properties are not well understood. Our main contribution is to show *elementary* tight bounds for the containment problem for regular queries. Specifically, we show that this problem is 2EXPSPACE-complete. For all extensions of regular queries known to date, the containment problem turns out to be non-elementary.

✉ Miguel Romero
m.romero.orth@gmail.com

Juan L. Reutter
jreutter@ing.puc.cl

Moshe Y. Vardi
vardi@cs.rice.edu

[1] Pontificia Universidad Católica de Chile, Santiago, Chile

[2] Universidad de Chile, Santiago, Chile

[3] Rice University, Houston, TX 77005, USA

Published online: 16 April 2016

Together with the fact that evaluating regular queries is not harder than evaluating UCN2RPQs, our results show that regular queries achieve a good balance between expressiveness and complexity, and constitute a well-behaved class that deserves further investigation.

Keywords Graph databases · Conjunctive regular path queries · Regular queries · Containment

1 Introduction

Graph databases have become a popular data model over the last decade. Important applications include the Semantic Web [3, 4], social network analysis [31], biological networks [38], and several others. The simplest model of a graph database is as an edge-labeled directed graph [15, 34]: nodes represent objects and a labeled edge between nodes represents the fact that a particular type of relationship holds between these two objects. For a survey of graph databases, see [2, 8].

Conceptually, graph databases differ from relational databases in that the topology of the data is as important as the data itself. Thus, typical graph database queries are *navigational*, asking whether some nodes are connected by paths satisfying some specific properties. The most basic query language for graph databases is that of *regular-path queries* (RPQs) [28], which selects pairs of nodes that are connected by a path conforming to a regular expression. A natural extension of RPQs is the class of *two-way regular-path queries* (2RPQs), which enables navigation of inverse relationships [17, 18]. In analogy to *conjunctive queries* (CQs) and *union of* CQs (UCQs), the class of *union of conjunctive two-way regular path queries* (UC2RPQs) enable us to perform unions, joins and projections over 2RPQs [17]. The navigational features present in these languages are considered essential in any reasonable graph query language [8].

More expressive languages have been studied, for example, in the context of knowledge bases and description logics [9, 10, 12, 14]. The class of *nested two-way regular path queries* (N2RPQs) and the corresponding class of *union of conjunctive* N2RPQs (UCN2RPQs), extend C2RPQs with an *existential test operator*, inspired in the language XPath [6, 41]. Previous results show that CN2RPQs is a well-behaved class, as it increases the expressive power of C2RPQs without increasing the complexity of evaluation or static analysis problems such as containment [9, 12, 14]. Yet the regular patterns detected by 2RPQs and N2RPQs are still quite simple: one can only check paths or for a very limited form of trees between two nodes. Thus, these languages cannot express queries involving more complex regular patterns.

One key property that the query classes of UC2RPQs and UCN2RPQs fail to have is that of *algebraic closure*. Indeed, note that the relational algebra is defined as the closure of a set of relational operators [1]. Also, the class of CQs is closed under project and join, while UCQs are also closed under union [1]. Similarly, the class of 2RPQs is closed under concatenation, union, and transitive closure. In contrast, UC2RPQs and UCN2RPQs are not closed under transitive closure, because even

the transitive closure of a binary UC2RPQ query is not a UC2RPQ query. Thus, UC2RPQs and UCN2RPQs are not natural classes of graph database queries in this sense.

In this paper we study the language of *Regular Queries* (RQs), which result from closing the class of UC2RPQs also under transitive closure. We believe that RQs fully capture regular patterns over graph databases. We define RQs as *binary nonrecursive Datalog* programs extended with the *transitive closure* of binary predicates (either extensional or intensional predicates). The class of RQs is a natural extension of UC2RPQs and UCN2RPQs and can express many interesting properties that UCN2RPQs cannot (see e.g. [12, 40, 43]), but its algorithmic properties until now have not been well understood.

It is easy to see that the complexity of evaluation of regular queries is the same as for UC2RPQs: NP-complete in combined complexity and NLOGSPACE-complete in data complexity. This is a direct consequence of the fact that RQs are subsumed by binary *linear* Datalog [23, 25]. Nevertheless, the precise complexity of checking the containment of RQs has been open so far. This is the focus of this paper.

The *containment problem* for queries asks, given two queries Ω and Ω', whether the answer of Ω is contained in the answer of Ω', over *all* graph databases. Checking query containment is crucial in several contexts, such as query optimization, view-based query answering, querying incomplete databases, and implications of dependencies [16, 22, 30, 32, 35, 36]. A non-elementary upper bound for RQ containment follows from [26, 27]. But, is the complexity of the containment problem elementary or non-elementary? Given the importance of the query-containment problem, a non-elementary lower bound for containment of RQs would suggest that the class may be too powerful to be useful in practice.

Our main technical contribution is to show elementary tight bounds for the containment problem of RQs. We attack this problem by considering an equivalent query language, called *nested* UC2RPQs (nUC2RPQs), which is of independent interest. A nUC2RPQ is basically a 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 [17, 19, 23] 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 conclude our investigation by studying some interesting restrictions and extensions of RQs. First we consider RQs of bounded *treewidth* [24, 29]. Intuitively, for

$k \geq 1$, a 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 C2RPQs [5]. 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 a RQ. By definition, all intensional predicates in a 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, even if 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 arbitrary databases instead of graph databases.

There are several other languages that are either more expressive or incomparable to regular queries. One of the oldest is *GraphLog* [25], which is equivalent to *first-order logic with transitive closure*. More recent languages include *extended* CRPQs [5], which extends CRPQs with *path variables*, XPath for graph databases [37, 39], and algebraic languages such as [33, 40]. 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. In fact, regular queries were investigated in [12, 14] (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 regular queries [12–14, 43]. Interestingly, the containment problem is non-elementary even for *positive* first-order logic with *unary* transitive closure [12, 14], which is a slight generalization of regular queries. Thus, regular queries seems to be the most expressive fragment of first-order logic with transitive closure that is known to have an elementary containment problem.

Part of this work has been previously presented in IDCT'15 [42]. However, this version contains several new material. To begin with, the definitions of RQs, nUC2RPQs and flat nUC2RPQs have been streamlined and are now much cleaner, and we have also added some new examples. Moreover, the main 2EXPSPACE bound was only sketched in the conference version, and proofs of correctness were not included; we only gave a simpler version of the construction (that was not enough for the proof) and just explained how it could be modified into a construction that was correct. This version contains the full construction, as well as proofs for correctness. Finally, all other results in Sections 7 and 8 are new to this version, as they were not included in the conference paper.

Organization We present preliminaries in Section 2. In Section 3 we introduce RQs and nUC2RPQs. The containment problem of RQs is analyzed in Sections 4–7: Section 4 presents and studies flat nUC2RPQs, Section 5 shows how to reduce from containment of flat nUC2RPQs to containment of a single-atom C2RPQ in a flat nUC2RPQ, Section 6 shows that the latter problem is in EXPSPACE, and Section 7 finishes the upper bound and provides the lower bound. In Section 8, we study restrictions and extensions of RQs and their impact in the complexity of evaluation and containment. Finally, in Section 9 we present some conclusions and future work.

2 Preliminaries

Graph Databases Let Σ be a finite alphabet. A *graph database* $\mathcal{G}$ over Σ is a pair (V, E), where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$. Thus, each edge in $\mathcal{G}$ is a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a-labeled edge from v to v' in $\mathcal{G}$. We define the finite alphabet $\Sigma^{\pm}$ as the extension of Σ with the *inverse* of each symbol, that is, $\Sigma^{\pm} = \Sigma \cup \{a^- \mid a \in \Sigma\}$. The *completion* $\mathcal{G}^{\pm}$ of a graph database $\mathcal{G}$ over Σ, is a graph database over $\Sigma^{\pm}$ that is obtained from $\mathcal{G}$ by adding the edge (v', a^-, v) for each edge (v, a, v') in $\mathcal{G}$.

Conjunctive Queries We assume familiarity with relational schemas and relational databases. A *conjunctive query* (CQ) is a formula in the $\exists, \wedge$-fragment of first-order logic. We adopt a rule-based notation: A CQ $\theta(x_1, \ldots, x_n)$ over the relational schema σ is a rule of the form $\theta(\bar{x}) \leftarrow R_1(\bar{y}_1), \ldots, R_m(\bar{y}_m)$, where R_i is a predicate symbol in σ, for each $1 \leq i \leq m$, $\bar{x}$ are the free variables, and the variables in some $\bar{y}_i$ that are not mentioned in $\bar{x}$ are the existential quantified variables. The *answer* of a CQ $\theta(x_1, \ldots, x_n)$ over a relational database $\mathcal{D}$ is the set $\theta(\mathcal{D}) = \{\bar{a} \mid \mathcal{D} \models \theta(\bar{a})\}$, of tuples that satisfies θ in $\mathcal{D}$. As usual, if θ is a *Boolean* CQ, that is, it has no free variables, we identify the answer false with the empty relation, and true with the relation containing the 0-ary tuple.

We want to use CQs for querying graph databases over a finite alphabet Σ. In order to do this, given an alphabet Σ, we define the schema $\sigma(\Sigma)$ that consists of one binary predicate symbol E_a, for each symbol $a \in \Sigma$. For readability purposes, we identify E_a with a, for each symbol $a \in \Sigma$. Each graph database $\mathcal{G} = (V, E)$ over Σ can be represented as a relational database $\mathcal{D}(\mathcal{G})$ over the schema $\sigma(\Sigma)$: The database $\mathcal{D}(\mathcal{G})$ consists of all facts of the form $E_a(v, v')$ such that (v, a, v') is an edge in $\mathcal{G}$.

A conjunctive query over Σ is simply a conjunctive query over $\sigma(\Sigma^{\pm})$. The answer $\theta(\mathcal{G})$ of a CQ θ over $\mathcal{G}$ is $\theta(\mathcal{D}(\mathcal{G}^{\pm}))$. A union of CQs (UCQ) Θ over Σ is a set $\{\theta_1(\bar{x}), \ldots, \theta_k(\bar{x})\}$ of CQs over Σ with the same free variables. The answer $\Theta(\mathcal{G})$ is $\bigcup_{1 \leq j \leq k} \theta_j(\mathcal{G})$, for each graph database $\mathcal{G}$.

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 $\mathsf{neq}(\varphi)$.

C2RPQs The basic mechanism for querying graph databases is the class of *two-way regular path queries*, or 2RPQs [18]. A 2RPQ E over Σ is a regular expression over $\Sigma^{\pm}$. Intuitively, E computes the pairs of nodes connected by a path whose label is in the language $L(E)$ defined by E. Formally, the *answer* $E(\mathcal{G})$ of a 2RPQ E over a graph database $\mathcal{G} = (V, E)$ is the set of pairs (v, v') of nodes in V for which there is a word $r_1 \cdots r_p \in L(E)$ such that (v, v') is in the answer of the CQ $\theta(x, y) \leftarrow r_1(x, z_1), \ldots, r_p(z_{p-1}, y)$ over $\mathcal{G}$. Note that when p is 0, the word $r_1 \cdots r_p$ corresponds to ε and then $\theta(x, y)$ becomes $x = y$.

The analogue of CQs in the context of graph databases is the class of *conjunctive* 2RPQs, or C2RPQs [17]. A C2RPQ $\gamma(\bar{x})$ over Σ is a rule of the form

$\gamma(\bar{x}) \leftarrow E_1(y_1, y_1'), \ldots, E_m(y_m, y_m')$, where $\bar{x}$ are the free variables, the variables in $\{y_1, y_1', \ldots, y_m, y_m'\}$ not mentioned in $\bar{x}$ are the existential quantified variables and E_i is a 2RPQ over Σ, for each $1 \leq i \leq m$. The *answer* $\gamma(\mathcal{G})$ of γ over a graph database $\mathcal{G}$ is defined in the obvious way. A union of C2RPQs (UC2RPQ) Γ over Σ is a finite set $\{\gamma_1(\bar{x}), \ldots, \gamma_k(\bar{x})\}$ of C2RPQs over Σ with the same free variables. We define $\Gamma(\mathcal{G})$ as $\bigcup_{1 \leq j \leq k} \gamma_j(\mathcal{G})$, for each graph database $\mathcal{G}$.

Datalog While UC2RPQs extends 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 $i \geq 0$, $P^i_\Pi(\mathcal{D})$ denote 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 $\Pi(\mathcal{D}(\mathcal{G}^\pm))$.

3 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, and that is better for defining the automata techniques that we use throughout the paper.

3.1 Regular Queries

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 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 (v, v') 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 1 Suppose we have a graph database of persons and its relationships. We have relations *knows* and *helps*, abbreviated k and h, respectively. Thus our alphabet is $\Sigma = \{k, h\}$. Lets say that a person p is a *friend* of person 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.

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

Example 2 Suppose now 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 UCN2RPQs (the extension of UC2RPQs with an existential test operator [6, 41]), as RQs are actually strictly more expressive than these other classes of queries. In particular, the transitive closure of a binary UC2RPQ cannot be expressed as a UC(N)2RPQ, and indeed using ideas from [12, 14] one can show that the queries in Examples 1 and 2 cannot be expressed by any UCN2RPQ, whereas these where easily expressed as RQs. Moreover, the class of RQs is not only closed under join and union, but also under transitive closure: the transitive closure of a binary RQ is always a RQ. This makes RQs a natural graph query language.

We start our investigation of RQs by establishing some basic results regarding the complexity of evaluation. Recall that the *evaluation problem* asks, given a query Ω, a graph database $\mathcal{G}$ and a tuple $\bar{t}$, whether $\bar{t} \in \Omega(\mathcal{G})$. Interestingly, we show that RQs are not harder to evaluate than UCN2RPQs.

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 ocurr 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):

$$P^+(x, y) \leftarrow P(x, y).$$
$$P^+(x, y) \leftarrow P^+(x, z), P(z, y).$$

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 Regular Queries 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 2 is translated into the following linear program:

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

As a consequence of this translation we can derive tight complexity bounds for the evaluation problem [23, 25].

Theorem 1 *The evaluation problem for RQs is* NP*-complete in combined complexity and* NLOGSPACE*-complete in data complexity.*

3.2 Nested UC2RPQs

Most of the proofs in this paper assume that regular queries are given in an equivalent definition called *nested* UC2RPQs. Nested UC2RPQs can be seen as a more intuitive extension of C2RPQs as they allow regular expression 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

[1]These programs are sometimes referred to as *stratified linear* programs, or *piecewise linear Programs* [45].

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 2 (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 1

- *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.

4 Containment of Regular Queries

Now we turn to the task of checking containment. Recall that the *containment problem* asks, given two queries Ω and Ω', whether Ω is *contained* in Ω', that is, whether $\Omega(\mathcal{G}) \subseteq \Omega'(\mathcal{G})$ for each graph database $\mathcal{G}$. Our main technical result is the following.

Theorem 2 *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 2 is a direct consequence of the following theorem.

Theorem 3 *The containment problem for nUC2RPQs is* 2EXPSPACE*-complete.*

We thus focus solely on Theorem 3. 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 $E() \leftarrow R(y, y')$, where R 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 4.1 we show how to go from nUC2RPQs to flat nUC2RPQs (with a possible exponential blowup). Then Section 5 shows how to reduce from containment of flat nUC2RPQs to containment of a *single-atom* C2RPQ and a flat nUC2RPQ, and Section 6 shows that the latter problem can be done in EXPSPACE. Finally, in Section 7 we put all the ingredients together and provide a proof for Theorems 3 and 2. The latter section also includes the proof for the 2EXPSPACE lower bound.

4.1 From nUC2RPQs to Flat nUC2RPQs

We start by introducing the class of flat nUC2RPQs.

Definition 3 (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.

By definition, a 2RPQ over Σ is a regular expression over $\Sigma^{\pm}$. However, since regular expressions and *nondeterministic finite automata* (NFAs) are equivalent formalisms to express regular languages, we can also represent 2RPQs as NFAs over $\Sigma^{\pm}$. By slightly abusing notation, we say that a NFA over $\Sigma^{\pm}$ is also a 2RPQ over $\Sigma^{\pm}$. We adopt the convention that all 2RPQs appearing in a flat nUC2RPQ are represented as NFAs. This is particularly important when we reduce containment of flat nUC2RPQs to containment of a single-atom C2RPQ in a flat nUC2RPQ, since the 2RPQs of the flat nUC2RPQ produced in this reduction are directly defined as NFAs,

and a translation of these to regular expressions may involve an exponential blow-up in size. For the sake of readability though, all the examples of flat nUC2RPQs presented in the paper are specified using regular expressions.

Example 3 The following flat nUC2RPQ is equivalent to the query from Example 2. 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 3 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 1 and 2 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 2 *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 $O(g)$. 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 $l \leq d$ such that $\mathcal{R}^l_{Ans} = \mathcal{R}^{l+1}_{Ans}$. Note that each rule in $\mathcal{R}^l_{Ans}$ satisfies condition (1) in Definition 3, 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 $|\mathcal{R}^l_{Ans}| \leq h^{1+w+\cdots+w^d}$. 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 $\mathcal{R}^l_{Ans}$. We define $\mathcal{S}_{Ans}$ as the set of rules obtained from $\mathcal{R}^l_{Ans}$ by replacing each 2RPQ by an equivalent NFA (recall that 2RPQs in a nUC2RPQ and thus in $\mathcal{R}^l_{Ans}$ are given as regular expressions, whereas 2RPQs in a flat nUC2RPQ are given as NFAs), and 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 3. 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 $\mathcal{R}^l_{Ans}$ 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 $O(g)$, as the translation from regular expressions to NFAs is linear. 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 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 4 (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 suc that

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

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

where, $p \geq 0$, $r_1 \cdots r_p \in L(E)$, 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 [19, 23]. 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 [21, 44] and the fact that each flat nUC2RPQ is equivalent to the union of its expansions.

Proposition 3 *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 [21], due to the presence of inverses:

Definition 5 (CQ Containment Mappings) If θ and θ' are two Boolean CQs over Σ, then 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.

Given flat nUC2RPQs Γ and Γ' over Σ, we construct a single-atom C2RPQ $\tilde{\gamma}() \leftarrow \tilde{E}(y, y')$, where $\tilde{E}$ is 2RPQ (represented as NFA), 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 strings over a larger alphabet Δ. More importantly, serialized expansions constitute a regular language. Thus, we can construct a NFA $\tilde{E}$ such that $L(\tilde{E})$ is precisely the set of serialized expansions of Γ. This technique has been already used before in [17, 18].
2. Next we need to serialize Γ'. Proposition 3 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.

In what follows, we fix flat nUC2RPQs Γ and Γ' and focus on the construction of $\tilde{E}$ and $\tilde{\Gamma}$.

5.1 Construction of $\tilde{E}$

Let M be 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 string U in Δ, we define U^{-1} as the string 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 strings over Δ. This is a natural extension of the representation given in [17] 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 4).

Definition 6 (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 $\varepsilon(x_1, x_2)$).

The *serialized expansion* W_π associated with π is the string 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 $R_i = E$ 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 $r_1 \cdots r_p \in L(E)$, 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 string over Δ is a *serialized expansion* of Γ if it is the serialized expansion associated with some expansion of Γ.

Example 4 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 $(h_i^l, \exists)$, $(h_i^l, 1)$, $(h_i^l, 2)$, respectively.

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

Proposition 4 *Let Γ be a flat nUC2RPQ over Σ. There exists a NFA $\tilde{E}$ over alphabet Δ, such that $W \in L(\tilde{E})$ if and only if W is a serialized expansion of Γ. Moreover, the size of $\tilde{E}$ 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 string could have any level.
2. $\mathcal{A}_R^+$ accepts all the strings 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 $S(x, y) \leftarrow E_1(y_1, y'_1), \ldots, E_m(y_m, y'_m)$, where each E_i is a 2RPQ (recall that E_i is specified as a NFA). 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 E_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 $\tilde{E}$ 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, $\tilde{E}$ 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 $\tilde{E}$ also is. □

5.2 Construction of $\tilde{\Gamma}$

Let $\tilde{\gamma}$ be the single-atom C2RPQ $\tilde{\gamma}() \leftarrow \tilde{E}(y, y')$. We can associate to each string $W = r_1 \cdots r_p \in L(\tilde{E})$, 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 $\varepsilon \notin L(\tilde{E})$, 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 3, 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 string 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 4. The rest of the variables in φ are *external* variables. For example, the internal and external variables of expansion φ from Example 4 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 string. 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 substring 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 4, positions 5,7,10,16 represent y in φ, positions 19,22 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 4 and that this string $r_1 \cdots r_p$ is the substring $W_\varphi[k+1, k+p]$ 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 4, position 3 represents z_1 and position 11 represents z_1'. Positions 4, 12 and 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 $1 \leq i, j \leq |W_\varphi|$ such that $W_\varphi[i], W_\varphi[j] \in \mathcal{V}$. There are several cases. Suppose first that $W_\varphi[i] = W_\varphi[j] = (h_k^l, s)$, for some $1 \leq k \leq 2M$, $0 \leq l \leq d$, 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 6. In Example 4, 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 4, the pairs $(5, 7)$, $(15, 18)$ and $(27, 29)$ are horizontally equivalent.

Suppose now that $t\ 1^l\ W'\ 2^l\ t'$ is a substring of W_φ and that the levels of symbols t, t' is $l \in \{1, \ldots, d\}$. Assume that t appears at position i in W_φ, and that $W_\varphi[j]$ is a symbol in W' of the form $(h_k^{l-1}, 1)$. Then we have that $\mathsf{var}(i) = \mathsf{var}(j)$ and thus i and j are equivalent. Indeed, this substring appears in W_φ precisely when we unfold an expression Q^+ (item (2), Definition 6) and W' represents an expansion ϕ of Q. By definition, $(h_k^{l-1}, 1)$ 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 $(h_k^{l-1}, 2)$. In either case we say that i and j are *vertically* equivalent. In Example 4, the pairs $(7, 10)$, $(19, 22)$ 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 $W_\varphi[i] = (h_k^l, s)$ and $W_\varphi[j] = (h_{k'}^l, s')$, for some $1 \leq k, k' \leq 2M$, $0 \leq l \leq d$ 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 6). 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 4, 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 *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 $k_1, \cdots k_p$ 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.*

Now we characterize equivalence in terms of regular languages.

Definition 7 (Foldings) Let w and u be strings 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 string $w = \$y_1 b^- a y_2\$$. Then, 3,2,1,2,3,4,3 is a $(3, 4)$-folding from $b y_1^- y_1 b^- a a^-$ 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 strings*. Equality strings are strings 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 string into W_φ. Equality strings are concatenations of *basic* equality strings. We have one type of basic equality string for each type of equivalence. For a level $0 \leq l \leq d$, G_l denotes the alphabet $\Delta^{\pm}_{\leq l}$, where $\Delta_{\leq l}$ 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 string α is a string that satisfies the regular expression $t\ G_l^*\ t + t^-\ G_l^*\ t^-$, where, $0 \leq l \leq d$ 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 string α and (i, j)-folding of α into W_φ.
2. A *vertical* equality string α is a string that satisfies the regular expression $e_1 + e_2$, where $e_1 = t_1\ 1^l\ G_{l-1}^*\ t_2 + t_2^-\ G_{l-1}^*\ (1^l)^-\ t_1^-$, for $t_1 \in \mathcal{V}$ with level l and t_2 of the form $(h_k^{l-1}, 1)$. Similarly, $e_2 = t_2\ G_{l-1}^*\ 2^l\ t_1 + t_1^-\ (2^l)^-\ G_{l-1}^*\ t_2^-$, for $t_1 \in \mathcal{V}$ with level l and t_2 of the form $(h_k^{l-1}, 2)$. It is easy to see that i and j are vertically equivalent iff there is a vertical equality string α and (i, j)-folding of α into W_φ.
3. A *semantic* equality string α is a string 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 $(h_{k'}^l, s')$. Clearly, i and j are semantically equivalent iff there is a semantic equality string α and (i, j)-folding of α into W_φ.

An equality string is a string 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 string. Note that the empty string ε is an equality string. The expression δ allows us to concatenate

foldings of basic equality string. The following proposition is immediate from the definition of equality string:

Proposition 6 *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 string 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* $\mathsf{last}(\mathcal{F}) = k'$.

In Example 4, 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. Equivalence of 5 and 33 is then witnessed by the equality string (now we indicate levels; bold symbols are due the expression δ; $[\cdot]$ indicates basic equality strings)

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

Now we define $\tilde{\Gamma}$. Let $w = w_1 \cdots w_p$ be a string over $\Sigma^\pm$. We define $\mathsf{serial}(w)$ as the language that contains all the strings 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 $0 \le i \le p$, α_i is an equality string. 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 1 *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 strings. Then the NFA $\mathcal{A}'$ guesses, on input $s_1 \cdots s_n$, positions $i_1 < \cdots < i_l$ such that $s_{i_1} s_{i_2} \cdots s_{i_l} \in L(\mathcal{A})$ and it checks that each intermediate substring $s_1 \cdots s_{i_1-1}, s_{i_1+1} \cdots s_{i_2-1}, \ldots, s_{i_l+1} \cdots s_n$ is an equality string, 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. □

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

Proposition 7 *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 4, we expand a 2RPQ E by choosing a string in $L(E)$. 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 string $S_\psi(i) \in L(A_\psi(i))$ 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 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 strings 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, $S_{\varrho'}(i) = \epsilon \in L(A_{\varrho'}(i))$, 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 $o_1 \ldots o_{n-1}$ 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 $\mu(o_0), \mu(o_1), \ldots, \mu(o_{n-1}), \mu(o_n)$. Let $j_1 < j_2 < \cdots < j_m$ be all the positions j in $\{0, \ldots, n\}$, 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 substring $s_{j_{k-1}+1} \cdots s_{j_k}$ 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 string $S_\varphi(i')$, 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 $q \in \{1, \ldots, p\}$ 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 $q' \in \{1, \ldots, p\}$ such that $W_\varphi[q'] \in \mathcal{V}$, and q and q' are equivalent (since μ is a containment mapping). By Proposition 6, there is an equality string 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 $S_{\varrho'}(i) \in \mathsf{serial}(s_1 \ldots s_n) \subseteq L(\mathsf{serial}(A_{\varphi'}(i))) = L(A_{\varrho'}(i))$.

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 strings $\alpha_2, \ldots, \alpha_m$ 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 $q \in \{1, \ldots, p\}$ such that $W_\varphi[q] \in \mathcal{V}$, and $\mathsf{pos}(\nu(o_0))$ and q are equivalent. By Proposition 6, there is an equality string 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 string. 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 $\xi_{\varphi'}^{-1}(z)$. 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 $y, y' \in \xi_{\varphi'}^{-1}(z)$. Then there exists a sequence $t_0 = t_1, t_1 = t_2, \cdots t_{l-1} = t_l$ of equality atoms in φ', where $l \geq 1$, $t_0 = y$ and $t_l = y'$. Since the t_ks belongs to U^{ext}, they are also external variables of ϱ'. As we mentioned above, for each $0 \leq k \leq l-1$, $t_l = t_{l+1}$ is either an equality atom in ϱ' or there is a basic expansion of the form $a_1(t_l, y_1), a_2(y_1, y_2), \cdots, a_n(y_{n-1}, t_{l+1})$ where $a_1 \ldots a_n$ is a nonempty equality string. 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 $a_1^k(v_k, y_1), a_2(y_1, y_2), \cdots, a_n^k(y_{n-1}, v_{k+1})$ with $a_1^k \cdots a_n^k$ a nonempty equality string. 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 strings 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 $a_1^k \cdots a_n^k$ 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 string 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 $y_0 \in \xi_{\varphi'}^{-1}(z)$ and $y_n \in \xi_{\varphi'}^{-1}(z')$, respectively. □

As a corollary of Proposition 3 and 7 we have that:

Corollary 1 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 4 *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.*

6 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 [17, 19, 23]. 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 [17, 19] to handle nUC2RPQs. In [17], 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 (string 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 E, this can be done easily with a 2NFA by guessing a folding of a word $w \in L(E)$ 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 next.

6.1 Cuts

To decide whether a single-atom C2RPQ $\gamma() \leftarrow E(y, y')$ 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 3). We can assume w.l.o.g. that $\varepsilon \notin L(E)$ and replace $\mathsf{neq}(\theta)$ by θ (see the beginning of Section 5.2). 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 $w_0 \cdots w_{k-1} \in L(E)$ and each z_j is distinct. Thus we can directly identify expansions of γ with strings in $L(E)$. 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 strings) 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 6.4.

Definition 8 (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 (recall P_j is given as a NFA),
  - 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 $C = (\rho, Inc, Mark,)$ 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 2 *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 string 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 $|\Gamma|$. □

6.2 Transition System Based on Cuts

We represent each linear CQ $\theta() \leftarrow w_0(z_0, z_1), \ldots, w_{k-1}(z_{k-1}, z_k)$ with the string $w = w_0, \ldots, w_{k-1}$, 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 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 string 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 string 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 string $w = w_0, \ldots, w_{k-1}$. 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

$$\mathit{Ans}(x_1, \ldots, x_n) \leftarrow P_1(u_1, v_1), \ldots, P_m(u_m, v_m)$$

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 7) from Section 5.2. 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 5.2, 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 $\tau^- = \tau_l^-, \ldots, \tau_1^-$, if $\tau = \tau_1, \ldots, \tau_l$.

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 string 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 $0 \leq i \leq l$,
   - 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 $C = (\rho, Inc, Mark,)$ and $C' = (\rho, Inc', Mark', States')$ in $Cuts(\Gamma)$, with $Inc \subseteq Inc'$, $Mark' = Mark \cup Inc' \setminus Inc$, $States = (s_1, \ldots, s_m)$ and $States' = (s'_1, \ldots, s'_m)$, 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 3 *Let $\Gamma(x, y)$ be a flat nUC2RPQ, $w = w_0, \ldots, w_{k-1}$ a string 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 $C = (\rho, Inc, Mark, States)$ and $C' = (\rho, Inc', Mark', States')$ 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 $1 \leq j \leq m$. 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 $p_0 \leq p_\ell$. 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 $\mathrm{C}uts(\Gamma)$, 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

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

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 $(c_I, 0) \Rightarrow^*_{(\Gamma_P, w)} c \Rightarrow^*_{(\Gamma_P, w)} (s^\ell_j, p_\ell)$, for some initial cut c_I, and $c[1]$ marks u_j. By condition D.5, we have that (s^ℓ_j, 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 $c^y_0, c^x_0, c^y_1, c^x_1, , \ldots, c^y_f, c^x_f, c^y$ witnessing the fact that (s^ℓ_j, p_ℓ) is final. In this case, we finish the construction of $c_1, \ldots, c_{2n}$ with the pairs c, (c^y_0, r_0); (c^x_0, r_0), (c^y_1, r_1); $\cdots$; (c^x_f, r_f), (c^y, p_ℓ).

($\Longleftarrow$) 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 substring. 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 $(c_i^x, \eta(z'_{i-1}))$, $(c_i^y, \eta(z'_i))$, for each $1 \leq i \leq q$, such that $(c_i^x, \eta(z'_{i-1}))$ and $(c_i^y, \eta(z'_i))$ 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_1^x, p_0) and (c_1^y, 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_1^x, p_0) and $(c_1^y, \eta(z'_1))$. In this case, we let $(c_1, q_1) = (c_1^x, p_0)$. 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 $(c_{e_0+1}^x, \eta(z'_{e_0}))$ and $(c_{e_0+1}^y, \eta(z'_{e_0+1}))$, 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 $(c_{e_1+1}^x, g_1)$ define a transitive run for $\Gamma_P(u_j, v_j)$. Thus we can define $(c_{e_1+1}^x, g_1)$ as the next element in our sequence.

Let $\mathcal{S}_1$ be the witness of the accepting run of $(c_{e_1+1}^x, g_1)$ and $(c_{e_1+1}^y, \eta(z'_{e_1+1}))$. 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 $(c_{e_1+1}^x, g_1)$, 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. □

6.3 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 strings 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 R be a 2RPQ (given as an NFA), and assume that Q is the set of states of R. Then each symbol of $\Sigma(R)$ consists a set of pairs of states of R. In other words, $\Sigma(R) = 2^{Q\times Q}$.
- 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 $Cuts(\Gamma)^2 = Cuts(\Gamma) \times Cuts(\Gamma)$. We define $\mathsf{Same}(\Gamma) = 2^{Cuts(\Gamma)^2}$, $\mathsf{Initial}(\Gamma) = \mathsf{Final}(\Gamma) = 2^{Cuts(\Gamma)} \times 2^{Cuts(\Gamma)}$, $\mathsf{Transitive}(\Gamma) = 2^{Cuts(\Gamma)^2} \times 2^{Cuts(\Gamma)^2}$. 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 $R \in \mathcal{R}_{\text{ans}}(\Gamma)$, a symbol from $\Sigma(R)$ 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,

$$\Sigma(\Gamma) = \mathsf{Same}(\Gamma) \times \mathsf{Initial}(\Gamma) \times \mathsf{Final}(\Gamma) \times \mathsf{Transitive}(\Gamma) \times \bigtimes_{R\in\mathcal{R}_{\text{ans}}(\Gamma)} \Sigma(R) \times \bigtimes_{P\in\mathcal{I}_{\text{ans}}(\Gamma)} \Sigma(\Gamma_P)$$

The following is shown directly from Lemma 2.

Lemma 4 *Let Γ be a binary flat nU2CRPQ. Then the number of different symbols $|\Sigma(\Gamma)|$ of $\Sigma(\Gamma)$ is exponential in $|Cuts(\Gamma)|$. In particular, $|\Sigma(\Gamma)|$ is double exponential in $|\Gamma|$.*

We now give intuition regarding this construction. Let w be a string 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 $Cuts(\Gamma) \times Cuts(\Gamma)$, 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 $Cuts(\Gamma)$. 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 $R \in \mathcal{R}_{\text{ans}}(\Gamma)$ 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 R, 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 string 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 *Let $\Gamma(x, y)$ be a flat nU2CRPQ over Σ. Then there is a NFA $\mathcal{A}^{va}_{\Gamma}$ that accepts a string 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 $R \in \mathcal{R}_{\text{ans}}(\Gamma)$, we build an NFA A_R that check that the symbols from $\Sigma(R)$ appearing in $u_0, \ldots, u_k$ are correct. At each position p, the automaton A_R stores two subsets $\mathcal{Q}_p^{\leftarrow}$ and $\mathcal{Q}_p^{\rightarrow}$ of $Q \times Q$:

- $\mathcal{Q}_p^{\leftarrow}$ contains all the pairs (q, q') such that there is a word w' that can be read from q to q' in R, and that can be $[p, p]$-folded into $\tau(w)$ using only positions smaller or equal to p.
- $\mathcal{Q}_p^{\rightarrow}$ contains all the pairs (q, q') such that there is a word w' that can be read from q to q' in R, and that can be $[p, p]$-folded into $\tau(w)$ using only positions greater or equal to p.

Then A_R checks that u_p is consistent with $\mathcal{Q}_p^{\leftarrow}$ and $\mathcal{Q}_p^{\rightarrow}$, that is, the symbol $\mathcal{Q}_p \in \Sigma(R)$ appearing in u_p is exactly the transitive closure of $\mathcal{Q}_p^{\leftarrow} \cup \mathcal{Q}_p^{\rightarrow}$. Note that $\mathcal{Q}_0^{\leftarrow} = \{(q, q) \mid q \in Q\}$ and that $\mathcal{Q}_{p+1}^{\leftarrow}$ is determined by $\mathcal{Q}_p^{\leftarrow}$ and a_p. Indeed, $\mathcal{Q}_{p+1}^{\leftarrow}$ is the reflexive and transitive closure of all the pairs (q, q') such that there is a pair $(\hat{q}, \hat{q}') \in \mathcal{Q}_p^{\leftarrow}$ with $\hat{q} \in \delta_R(q, a_p^-)$ and $q' \in \delta_R(\hat{q}', a_p)$, where δ_R is the transition function of R. Similarly, $\mathcal{Q}_k^{\rightarrow} = \{(q, q) \mid q \in Q\}$ and $\mathcal{Q}_p^{\rightarrow}$ is determined by $\mathcal{Q}_{p+1}^{\rightarrow}$ and a_p. Then $\mathcal{Q}_p^{\leftarrow}$ and $\mathcal{Q}_p^{\rightarrow}$ can be easily computed by an NFA. Note that the number of states of A_R is exponential with respect to $|Q|$, and thus w.r.t $|Cuts(\Gamma)|$.

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 $R_1, \ldots, R_n$ 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 $\Sigma(R_1), \ldots, \Sigma(R_n), \Sigma(\Gamma_{P_1}), \ldots, \Sigma(\Gamma_{P_r})$. Thus A^{same} can check directly that in each u_p, the symbol from $\mathsf{Same}(\Gamma)$ is consistent with the symbols from $\Sigma(R_1), \ldots, \Sigma(R_n), \Sigma(\Gamma_{P_1}), \ldots, \Sigma(\Gamma_{P_r})$.

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 $\mathcal{E}_p^{\leftarrow}$ 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 $\mathcal{E}_p^{\rightarrow}$ 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 $\mathcal{F}R_p$ 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 $C_0^y, C_0^x, C_1^y, C_1^x, \ldots, C_\ell^y, C_\ell^x$ such that
  - All the p_is are distinct and $p \leq p_i$, 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 $0 \leq i \leq \ell - 1$, 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 a final cut C_F of Γ such that

$$(C', p) \Rightarrow^*_{(\Gamma,w)} (C_\ell^x, p_\ell) \Rightarrow^*_{(\Gamma,w)} (C_F, k)$$

$A^{\text{final}}_{x,y}$ computes $\mathcal{E}_p^{\leftarrow}$, $\mathcal{E}_p^{\rightarrow}$ and $\mathcal{F}R_p$ as follows:

- Note that $C \in \mathcal{E}_p^{\leftarrow}$ iff there are cuts C', C'' such that $C' \in \mathcal{E}_{p-1}^{\leftarrow}$, $(C', p - 1) \Rightarrow_{(\Gamma,\tau(w))} (C'', p)$ and $(C'', p) \Rightarrow^*_{(\Gamma,\tau(w))} (C, p)$. Thus $\mathcal{E}_p^{\leftarrow}$ is determined by $\mathcal{E}_{p-1}^{\leftarrow}$, u_p and a_{p-1}. Thus $A^{\text{final}}_{x,y}$ can guess the $\mathcal{E}_p^{\leftarrow}$s, starting from $\mathcal{E}_0^{\leftarrow}$, 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 $\mathcal{E}_p^{\leftarrow}$, we have that $\mathcal{E}_p^{\rightarrow}$ is determined by $\mathcal{E}_{p+1}^{\rightarrow}$, u_p and a_p. Thus we can guess the $\mathcal{E}_p^{\rightarrow}$s and check that $\mathcal{E}_k^{\rightarrow}$ 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 $\mathcal{F}R_p$ is determined by $\mathcal{F}R_{p+1}$, u_p, a_p, $\mathcal{E}_p^{\leftarrow}$ and $\mathcal{E}_p^{\rightarrow}$. Thus $A^{\text{final}}_{x,y}$ guesses the $\mathcal{F}R_p$s and checks that $\mathcal{F}R_k$ 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 $D_y, D'_x \in \mathcal{E}_k^{\rightarrow}$.

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 $C^y \in \mathcal{E}_p^{\rightarrow}$. This corresponds to condition (1) in the definition of final cut.
2. there are pairs (D, D_y) and $(\hat{C}, \hat{C})' \in \mathcal{F}R_{p+1}$ such that $(C, p) \Rightarrow^*_{(\Gamma,\tau(w))} (D, p) \Rightarrow_{(\Gamma,\tau(w))} (\hat{C}, p+1)$, D_y marks y, $D_y \in \mathcal{E}_p^{\leftarrow}$ 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 $\mathcal{F}R_{p+1}, a_p, u_p, \mathcal{E}_p^{\leftarrow}$ and $\mathcal{E}_p^{\rightarrow}$, thus $A^{\text{final}}_{x,y}$ can check whether the $\mathcal{F}_p$s are correct. Note that the number of states of $A^{\text{final}}_{x,y}$ 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 $A^{\text{initial}}_{x,y}$ 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 $\mathcal{B}R_p$ 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 $\mathcal{F}R_{p+1}$ and pairs $(D_1, D_2), (D_3, D_4), \ldots, (D_{2h-1}, D_{2h})$ from $\mathcal{B}R_{p-1}$, 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 $\mathcal{F}R_{p+1}$ and pairs $(D_1, D_2), (D_3, D_4), \ldots, (D_{2h+1}, D_{2h+2})$ from $\mathcal{B}R_{p-1}$, 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 R in Γ, an NFA of the form A_R, 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 $O(|\Gamma|) = O(|\mathrm{C}uts(\Gamma)|)$ NFAs, each with a number of states exponential in $|\mathrm{C}uts(\Gamma)|$. It follows that the number of states of $\mathcal{A}_\Gamma^{\text{va}}$ is also exponential in $|\mathrm{C}uts(\Gamma)|$. □

We can finally proceed to build the desired NFA $\mathcal{A}_\Gamma$ that gives the strings corresponding to the linearizations of a flat nUC2RPQ Γ. This NFA needs to simulate the system $T_{(\Gamma,w)}$, from an initial cut of a 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 strings, as explained above.

Lemma 6 *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 string 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 $\mathrm{C}uts(\Gamma)$. 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 string 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. □

6.4 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.

Theorem 5 *The problem of checking whether a single atom C2RPQ is contained in a flat nUC2RPQ is in* EXPSPACE.

Proof Let $\gamma() \leftarrow E(y, y')$ be the single-atom C2RPQ, where E is 2RPQ given as NFA, 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. The number of states of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ is exponential in $|\mathit{Cuts}(\hat{\Gamma})|$.
- 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 $a_0 \cdots a_{k-1} \in L(E)$. Clearly the number of states of $\mathcal{A}_\gamma$ is polynomial in γ.
- We build the NFA $\mathcal{A}_{\hat{\Gamma}}$, as in Lemma 6. The number of states of $\mathcal{A}_{\hat{\Gamma}}$ is $|\mathit{Cuts}(\hat{\Gamma})|$.

From Lemma 6 and 3, it follows that the language of $(\mathcal{A}_{\hat{\Gamma}})^C$ intersected with the language of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ is precisely those strings 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 $(\mathcal{A}_{\hat{\Gamma}})^C$ is not empty. The number of states of $\mathcal{A}^{\text{va}}_{\hat{\Gamma}}$ and $(\mathcal{A}_{\hat{\Gamma}})^C$ is single exponential in $|\mathit{Cuts}(\hat{\Gamma})|$, and thus double exponential in $|\Gamma|$. Then we can check non emptiness in EXPSPACE using a standard on-the-fly implementation. □

7 Putting it all Together: Upper and Lower Bounds for Containment of nUC2RPQs

Theorems 5 and 4 provide an immediate proof that the containment problem for flat nUC2RPQs is in EXPSPACE. However, we cannot use this result 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 2 and 3). Section 7.1 shows how to refine the constructions in the previous sections to arrive at the desired 2EXPSPACE algorithm, and Section 7.2 contains the matching lower bound.

7.1 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 2). 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 7 *Let Γ be a binary flat nUC2RPQ with depth d, height h, width w and weight g. Then the size of $Cuts(\Gamma)$ 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 containment of nUC2RPQs (or RQs).

1. Given nUC2RPQs Ω and Ω', we unfold these queries to construct flat nUC2RPQs Γ and Γ'. By Proposition 2, 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. 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 7 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 $(\mathcal{A}_{\hat{\Gamma}})^C$ from Theorem 5 is triple-exponential in $|\Omega|$ and $|\Omega'|$. To decide whether $\Omega \subseteq \Omega'$, we check the intersection of these automata for emptiness, which can be done in 2EXPSPACE.

7.2 Lower Bound

By combining techniques from [17, 23], we can show a matching lower bound for containment of nUC2RPQs, and conclude Theorems 3 and 2.

Theorem 6 *The containment problem for nUC2RPQs is* 2EXPSPACE*-hard.*

Proof The proof is based on ideas from [23]. 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 string of length 2^{2^n}. The *symbols* of the string 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 [23], 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

$$\begin{aligned}
\mathit{Bit}(x, y) &\leftarrow \mathtt{IsAddress}(x, x), \mathtt{Zero}(x, x), \mathtt{Carry0}(x, x), \mathtt{E}(x, y).\\
\mathit{Bit}(x, y) &\leftarrow \mathtt{IsAddress}(x, x), \mathtt{Zero}(x, x), \mathtt{Carry1}(x, x), \mathtt{E}(x, y).\\
\mathit{Bit}(x, y) &\leftarrow \mathtt{IsAddress}(x, x), \mathtt{One}(x, x), \mathtt{Carry0}(x, x), \mathtt{E}(x, y).\\
\mathit{Bit}(x, y) &\leftarrow \mathtt{IsAddress}(x, x), \mathtt{One}(x, x), \mathtt{Carry1}(x, x), \mathtt{E}(x, y).\\
\mathit{ConfAddress}(x, y) &\leftarrow \mathit{Bit}^+(x, y).
\end{aligned}$$

Above, the variable x represents an address position. We consider the four possible values for the address bit and carry bit. The atom $\mathtt{E}(x, y)$ connects adjacent positions. We also have rules

$$\begin{aligned}
\mathit{ConfSymbol}(x, y) &\leftarrow \mathit{ConfAddress}(x, z), \mathtt{IsSymbol}(z, z), \mathtt{Q}_a(z, z), \mathtt{E}(z, y).\\
\mathit{ConfSymbol}(x, y) &\leftarrow \mathit{ConfAddress}(x, z), \mathtt{IsSymbol}(z, z), \mathtt{Q}_a(z, z), \$(z, y).\\
&\quad \text{(for each symbol } a\text{)}\\
\mathit{Comp}(x, y) &\leftarrow \mathit{ConfSymbol}^+(x, y).
\end{aligned}$$

The predicate *ConfSymbol* describes an address, followed by a symbol a. The atom $\mathtt{E}(z, y)$ 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

$$\mathit{Final}(x, y) \leftarrow \mathit{ConfAddress}(x, y), \mathtt{IsSymbol}(y, y), \mathtt{Q}_a(y, y).$$

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

$$\mathit{Ans}() \leftarrow \mathtt{Start}(x, x), \mathit{Comp}(x, z), \mathit{Final}(z, y).$$

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, $\mathit{dist}_{\leq i}$, $\mathit{dist}_{<i}$ and equal_i. For each $0 < i \leq n$, we have a rule $\mathit{dist}_i(x, y) \leftarrow \mathit{dist}_{i-1}(x, z), \mathit{dist}_{i-1}(z, y)$. We also have rules $\mathit{dist}_0(x, y) \leftarrow \mathtt{E}(x, y)$ and $\mathit{dist}_0(x, y) \leftarrow \$(x, y)$. Clearly, the predicate $\mathit{dist}_i(x, y)$ holds precisely when there is a path of length 2^i from x to y, consisting of $\mathtt{E}$-labeled or $\$$-labeled edges.

For each $0 < i \leq n$, we have two rules

$$\begin{aligned}
\mathit{dist}_{\leq i}(x, y) &\leftarrow \mathit{dist}_{\leq i-1}(x, z), \mathit{dist}_{\leq i-1}(z, y).\\
\mathit{dist}_{<i}(x, y) &\leftarrow \mathit{dist}_{<i-1}(x, z), \mathit{dist}_{\leq i-1}(z, y).
\end{aligned}$$

We also have rules $\mathit{dist}_{\leq 0}(x, y) \leftarrow \mathtt{E}(x, y)$, $\mathit{dist}_{\leq 0}(x, y) \leftarrow \$(x, y)$, $\mathit{dist}_{\leq 0}(x, x) \leftarrow$ true and $\mathit{dist}_{<0}(x, x) \leftarrow$ true. Here, $\mathit{dist}_{\leq i}$ holds precisely when there is a path of length at most 2^i from x to y, and $\mathit{dist}_{<i}(x, y)$ 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

$$equal_i(x, y) \leftarrow equal_{i-1}(x, y), equal_{i-1}(x', y'), dist_{i-1}(x, x'), dist_{i-1}(y, y').$$
$$equal_0(x, y) \leftarrow \#_1(x, x'), \#_2(y, y'), ?(x, x), ?(y, y).$$
$$\text{(for each } \#_1, \#_2 \in \{\mathtt{E}, \$\} \text{ and } ? \in \{\mathtt{Zero}, \mathtt{One}\})$$

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 {Start, IsAddress, IsSymbol, Zero, One, Carry0, Carry1} $\cup$ {Q_a | 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 [23]. 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

$$Ans() \leftarrow \mathtt{Start}(x, x), dist_{<n}(x, y), \$(y, z).$$
$$Ans() \leftarrow \mathtt{Start}(x, x), dist_{<n}(x, y), \mathtt{IsSymbol}(y, y).$$
$$Ans() \leftarrow \mathtt{Start}(x, x), dist_{n}(x, y), \mathtt{IsAddress}(y, y).$$

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 $? \in \{\mathtt{E}, \$\}$, we add rules

$$Ans() \leftarrow \mathtt{IsSymbol}(x, x), ?(x, y), dist_{<n}(y, z), \$(z, z').$$
$$Ans() \leftarrow \mathtt{IsSymbol}(x, x), ?(x, y), dist_{<n}(y, z), \mathtt{IsSymbol}(z, z).$$
$$Ans() \leftarrow \mathtt{IsSymbol}(x, x), ?(x, y), dist_{n}(y, z), \mathtt{IsAddress}(z, z).$$

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

$$Ans() \leftarrow \mathtt{Start}(x, x), dist_{<n}(x, y), \mathtt{One}(y, y).$$

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

$$Ans() \leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), dist_{\leq n}(x,y), \texttt{IsSymbol}(y,y), \texttt{\$}(y,z).$$

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

$$AllOnes_i(x,y) \leftarrow AllOnes_{i-1}(x,z), AllOnes_{i-1}(z,y). \qquad \text{(for each } 0 < i \leq n\text{)}$$
$$AllOnes_0(x,y) \leftarrow \texttt{E}(x,y), \texttt{One}(x,x).$$

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

$$SameConf(x,y) \leftarrow \texttt{E}^+(x,y).$$
$$NextConf(x,y) \leftarrow SameConf(x,z), \texttt{\$}(z,z'), SameConf(z',y).$$

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

$$Ans() \leftarrow \texttt{Start}(x,x), dist_n(x,y), \texttt{Q}_a(y,y). \quad \text{(for each symbol } a \neq (s_0,\perp)\text{)}$$
$$Ans() \leftarrow \texttt{Start}(x,x), dist_n(x,y), SameConf(y,z), \texttt{IsSymbol}(z,z), \texttt{Q}_a(z,z).$$
(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. □

8 Restrictions and Variants of Regular Queries

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.

8.1 RQs with Bounded Treewidth

The notion of treewidth measures the "tree-likeness" of a CQ. It is well-known that CQs of *bounded treewidth* can be evaluated in polynomial time (see e.g. [24, 29]). This tractability result can be easily extended to bounded treewidth (U)C2RPQs [5] (see also [7]). 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 a RQ is the maximum treewidth over its rules.

Proposition 8 *Let $k \geq 1$. There is a* PTIME *algorithm that, given a RQ Ω of treewidth at most k, a graph database $\mathcal{G}$ and a tuple $\bar{t}$, decides whether $\bar{t} \in \Omega(\mathcal{G})$.*

Proof 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 $\bar{t} \in \Omega(\mathcal{G})$, also in polynomial time. □

Note that the queries of the reduction in Theorem 7.2 are of bounded treewidth. Thus, although evaluation is more efficient for RQs of bounded treewidth, containment is still 2EXPSPACE-hard in this case.

8.2 RQs with Bounded Depth

The *depth* of a RQ is the maximum length of a directed path from an extensional predicate to the Ans predicate in its dependence graph, minus 1.

Proposition 9 *Let $d \geq 1$. There is an* EXPSPACE *algorithm that, given RQs Ω and Ω' of depth at most d, decide whether Ω is contained in Ω'*

Proof We simply use the same algorithm for containment of RQs (Section 7.1). 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.

8.3 Intensional Predicates with Unbounded Arity: Generalized RQs

By definition, each intentional predicate in a RQ, distinct from Ans, has arity 2. If we drop this condition we arrive at what we call *generalized* RQs. More formally, a *generalized* RQ over Σ is simply 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 6 Recall our database of persons and its relationships from Examples 1 and 2, 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 regular query:

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

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. [46]).

Proposition 10 *Evaluation of generalized RQs is* PSPACE*-complete, in combined complexity.*

On the positive side, we have that checking containment of generalized RQs is not harder than the case of RQs.

Proposition 11 *The containment problem for generalized RQs is* 2EXPSPACE*-complete.*

Proof Using the same argument as in the case of nUC2RPQs, we have that Proposition 2 still applies when we start with a generalized RQ Γ. Then the proposition follows from results in Section 7.1. □

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 $k \geq 1$, a *k-generalized* RQ over Σ is a nonrecursive extended Datalog program over Σ, where all intensional predicates, except possibly for Ans, have arity at most k. Note that RQs are contained in the class of 2-generalized RQs. The following proposition is immediate.

Proposition 12 *Let $k \geq 1$ be an integer.*

- *The evaluation problem for k-generalized RQs is* NP*-complete in combined complexity.*
- *The containment problem for k-generalized RQs is* 2EXPSPACE*-complete.*

8.4 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 13 *The containment problem for generalized RQs over a relational schema 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 11.

We use an idea from [19] 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 [19], it follows that Ω_1 is contained in Ω_2 iff Ω'_1 is contained in Ω'_2. □

9 Conclusions

The results in this paper show that regular queries achieve a good balance between expressiveness and complexity: they are sufficiently expressive to subsume UC2RPQs and UCN2RPQs, and they are not harder to evaluate than UCN2RPQs. While checking containment of regular queries is harder than checking containment of UC2RPQs, it is still elementary. Moreover, all generalizations of regular queries known to date worsen the complexity of the containment problem to non-elementary

or even undecidable. Thus we believe that regular queries constitute a well-behaved class that deserves further investigation.

We also show that containment of RQs of bounded depth has the same complexity of checking containment of C2RPQs, and that in general the blowup depends mostly on the width and depth of queries. This is a much more practical bound, since the typical examples of recursive queries used in industry can generally be expressed as programs with low width and depth.

An interesting research direction is to study the containment problem of a Datalog program in a regular query. Decidability of this problem follows from [26, 27], nevertheless the precise complexity is open. Although it is not clear how to extend the techniques presented in this paper to containment of Datalog in regular queries, we conjecture that this problem is elementary.

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. [9, 10, 20]). Following this direction, it would also be interesting to understand whether the techniques introduced in this paper can be use to study the containment problem for Guarded Regular Queries [11], a language that shares a number of similarities with Regular Queries.

Acknowledgments We'd like to thank Pablo Barceló for helpful discussions on regular queries. The third author is grateful to Vincent Jugé for early discussions on containment of regular queries. First and second authors are supported by the Millennium Nucleus Center for Semantic Web Research Grant NC120004, and first author also by Fondecyt Iniciación grant 11130648.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases Addison-Wesley (1995)
2. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv **40**(1) (2008)
3. Arenas, M., Gutierrez, C., Miranker, D.P., Pérez, J., Sequeda, J.F.: Querying semantic data on the web? ACM SIGMOD Rec **41**(4), 6–17 (2013)
4. Arenas, M., Pérez, J.: Querying semantic web data with sparql. In: PODS, pp. 305–316 (2011)
5. Barcelo, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM TODS **37**(4), 31 (2012)
6. Barceló, P., Pérez, J., Reutter, J.L.: Relative expressiveness of nested regular expressions. In: AMW, pp. 180–195 (2012)
7. Barcelo, P., Romero, M., Vardi, M.Y.: Semantic acyclicity on graph databases. In: PODS, pp. 237–248 (2013)
8. Barceló Baeza, P.: Querying graph databases. In: PODS, pp. 175–188 (2013)
9. Bienvenu, M., Calvanese, D., Ortiz, M., Simkus, M.: Nested regular path queries in description logics. In: KR (2014)
10. Bienvenu, M., Ortiz, M., Šimkus, M.: Conjunctive regular path queries in lightweight description logics. In: IJCAI, pp. 761–767 (2013)
11. Bienvenu, M., Ortiz, M., Šimkus, M.: Navigational queries based on frontier-guarded datalog Preliminary results. In: Alberto Mendelzon International Workshop on Foundations of Data Management, p. 162 (2015)
12. Bourhis, P., Krötzsch, M., Rudolph, S.: How to best nest regular path queries. In: Description Logics (2014)
13. Bourhis, P., Krötzsch, M., Rudolph, S.: Query containment for highly expressive datalog fragments. preprint arXiv: 1406.7801 (2014)

14. Bourhis, P., Krötzsch, M., Rudolph, S.: Reasonable highly expressive query languages. In: IJCAI, pp. 2826–2832 (2015)
15. Buneman, P.: Semistructured data. In: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 117–121 (1997)
16. Buneman, P., Davidson, S., Hillebrand, G., Suciu, D.: A query language and optimization techniques for unstructured data. ACM SIGMOD Rec **25**(2), 505–516 (1996)
17. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.: Containment of conjunctive regular path queries with inverse. In: 7th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 176–185 (2000)
18. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.: Rewriting of regular expressions and regular path queries. J. Comput. Syst. Sci **64**(3), 443–465 (2002)
19. Calvanese, D., De Giacomo, G., Vardi, M.Y.: Decidable containment of recursive queries. Theor. Comput. Sci **336**(1), 33–56 (2005)
20. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Conjunctive query containment and answering under description logic constraints. ACM Trans. Comput. Log **9**(3) (2008)
21. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proceedings of the ninth annual ACM symposium on Theory of computing, pp. 77–90 (1977)
22. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: 2013 IEEE 29Th International Conference on Data Engineering (ICDE), p. 1995
23. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive datalog programs. In: Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 55–66 (1992)
24. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. Theor. Comput. Sci **239**(2), 211–229 (2000)
25. Consens, M., Mendelzon, A.: Graphlog: A visual forMalism for real life recursion. In: 9th ACM Symposium on Principles of Database Systems (PODS), pp. 404–416 (1990)
26. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Inf. Comput **85**(1), 12–75 (1990)
27. Courcelle, B.: Recursive queries and context-free graph grammars. Theor. Comput. Sci **78**(1), 217–244 (1991)
28. Cruz, I., Mendelzon, A., Wood, P.: A graphical query language supporting recursion. In: ACM Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD), pp. 323–330 (1987)
29. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Principles and Practice of Constraint Programming-CP 2002, pp. 310–326. Springer (2002)
30. Fagin, R., Vardi, M.Y.: The theory of data dependenciesan overview. In: Automata, Languages and Programming, pp. 1–22. Springer (1984)
31. Fan, W.: Graph pattern matching revised for social network analysis. In: Proceedings of the 15th International Conference on Database Theory, pp. 8–21 (2012)
32. Fernandez, M., Florescu, D., Levy, A., Suciu, D.: Verifying integrity constraints on web sites. In: IJCAI, vol. 99, pp. 614–619 (1999)
33. Fletcher, G.H., Gyssens, M., Leinders, D., Surinx, D., Van den Bussche, J., Van Gucht, D., Vansummeren, S., Wu, Y.: Relative expressive power of navigational querying on graphs. Inf. Sci **298**, 390–406 (2015)
34. Florescu, D., Levy, A.Y., Mendelzon, A.O.: Database techniques for the world-wide web: A survey. SIGMOD Rec **27**(3), 59–74 (1998)
35. Friedman, M., Levy, A.Y., Millstein, T.D., etal: Navigational plans for data integration. AAAI/IAAI **1999**, 67–73 (1999)
36. Imielinski, T., Lipski, W.: Incomplete information in relational databases. J. ACM **31**(4), 761–791 (1984)
37. Kostylev, E.V., Reutter, J.L., Vrgoc, D.: Containment of data graph queries. In: ICDT, pp. 131–142 (2014)
38. Lacroix, Z., Murthy, H., Naumann, F., Raschid, L.: Links and paths through life sciences data sources. In: Data Integration in the Life Sciences, pp. 203–211. Springer (2004)
39. Libkin, L., Martens, W., Vrgoč, D.: Querying graph databases with xpath. In: Proceedings of the 16th International Conference on Database Theory, pp. 129–140 (2013)
40. Libkin, L., Reutter, J., Vrgoč, D.: Trial for rdf: adapting graph query languages for rdf data. In: PODS, pp. 201–212 (2013)

41. Pérez, J., Arenas, M., Gutierrez, C.: nSPARQL: A navigational language for RDF. J. Web Semant **8**(4), 255–270 (2010)
42. Reutter, J.L., Romero, M., Vardi, M.Y.: Regular queries on graph databases. In: 18th International Conference on Database Theory (ICDT 2015), vol. 31, pp. 177–194 (2015)
43. Rudolph, S., Krötzsch, M.: Flag & check: Data access with monadically defined queries. In: Proceedings of the 32nd symposium on Principles of database systems, pp. 151–162 (2013)
44. Sagiv, Y., Yannakakis, M.: Equivalences among relational expressions with the union and difference operators. J. ACM (JACM) **27**(4), 633–655 (1980)
45. Ullman, J.D.: Principles of database and Knowledge-Base systems computer science press (1989)
46. Vorobyov, S., Voronkov, A.: Complexity of nonrecursive logic programs with complex values. In: PODS, pp. 244–253 (1998)