Regular Queries on Graph Databases

Juan Reutter[1], Miguel Romero[2], and Moshe Y. Vardi[3]

1 **Pontificia Universidad Católica de Chile**
`jreutter@ing.puc.cl`
2 **Universidad de Chile**
`mromero@dcc.uchile.cl`
3 **Rice University**
`vardi@cs.rice.edu`

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 with *transitive closure* rules. 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. 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.

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 [27], biological networks [34], and several others. The standard model for a graph database is as an edge-labeled directed graph [12, 30]: 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 [1, 5].

Conceptually, graph databases differs 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) [24], 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 enable navigation of inverse relationships [14, 15]. 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 [14]. The navigational features present in these languages are considered essential in any reasonable graph query language [5].

licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

More expressive languages have been studied, for example, in the context of knowledge bases and description logics [11, 9, 8]. The class of *nested two-way regular path queries* (N2RPQs) and the corresponding class of *union of conjunctive* N2RPQs (UCN2RPQs), extends C2RPQs with an *existential test operator*, inspired in the language XPath [37, 7]. Typical results show that CN2RPQs is a well-behaved class, as it increases the expressive power of C2RPQs without increasing the complexity of evaluation or containment [11, 8]. Yet the regular patterns detected by 2RPQs and N2RPQs are still quite simple: they speak of paths and a restricted form of trees. 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*. To see that, note that the relational algebra is defined as the closure of a set of relational operators [2]. Also, the class of CQs is closed under select, project, and join, while UCQs are also closed under union [2]. Similarly, the class of 2RPQs is closed under concatenation, union, and transitive closure. In contrast, UC2RPQs and UCN2RPQs are not closed under transitive closure. For example, 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 paper we study the language of *Regular Queries* (RQ), which result from closing the class of UC2RPQs also under transitive closure. We believe that RQ fully captures regular patterns over graph databases. We define RQ as *binary nonrecursive Datalog* programs with the addition of *transitive closure* rules of the form $S(x, y) \leftarrow R^+(x, y)$. Such a rule defines S as the transitive closure of the predicate R (which may be defined by other rules). The class of RQ queries is a natural extension of C2RPQs and CN2RPQs and can express many interesting properties that CN2RPQs can not (See e.g. [**?**, 36]), 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 regular queries are subsumed by binary *linear* Datalog [21, 19]. Nevertheless, the precise complexity of checking the containment of regular queries 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 [13, 18, 28, 31, 26, 32]. A non-elementary upper bound for regular query containment follows from [22, 23]. 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 regular queries 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 regular queries. We attack this problem by considering an equivalent query language, called *nested unions of* C2RPQs (nested UC2RPQs), which is of independent interest. The intuition is that a regular query can be unfolded to construct an equivalent nested UC2RPQ. This is remiscent of the connection between nonrecursive Datalog and UCQs: each nonrecursive program can be unfolded to construct an equivalent UCQ [2]. Unfoldings of regular queries may involve an exponential blow up in size, and thus we can think of regular queries as a succinct version of nested UC2RPQs. We also analyze the impact of this succinctness on the complexity of the containment problem.

Remarkably, despite the considerable gain in expressive power that comes with nesting UC2RPQs, we are able to show that checking containment of nested UC2RPQs is Expspace-complete, i.e, it is not harder than checking containment of UC2RPQs [14]. Our proof is based on two novel ideas:

1. We reduce containment of nested UC2RPQs, to containment of a 2RPQ in a nested UC2RPQ. The reduction is based on a *serialization* technique, where we represent expansions of nested UC2RPQs as strings, and modify the original nested UC2RPQs accordingly.
2. We show that checking containment of a 2RPQ E in a nested UC2RPQ Γ can be done in Expspace. Here, we exploit automata-theoretic techniques used before, e.g. in [14, 16, 19] 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 novel representation of the *partial mappings* of Γ to the expansions of E. This representation is robust against nesting and does not involve a non-elementary blow up in size.

This result yields a 2Expspace upper bound for containment of regular queries, and we also provide a matching lower bound. Thus, the succinctness of regular queries is inherent and the containment problem is 2Expspace-complete.

There are several other languages that are either more expressive or incomparable to regular queries. One of the oldest is *GraphLog* [21], which is equivalent to *first-order logic with transitive closure*. More recent languages include *extended* CRPQs [6], which extends CRPQs with *path variables*, XPath for graph databases [35, 33], and algebraic languages such as [29, 36]. 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 [11] (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 [38, 10, 11]. Interestingly, the containment problem is non-elementary even for *positive* first-order logic with *unary* transitive closure [11], which is a slight generalization of regular queries.

Organization. We present preliminaries in Section 2. In Section 3 we introduce regular queries. The containment problem of regular queries is analyzed in Section 4. Finally, in Section 5 we conclude the paper by discussing realistic restrictions on regular queries 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 $\Sigma^{\pm} = \Sigma \cup \{a^- \mid a \in \Sigma\}$, that is, $\Sigma^{\pm}$ is the extension of Σ with the *inverse* of each symbol. 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 $\bar{x}$ are the free variables, the variables in some $\bar{y}_i$ not mentioned in $\bar{x}$ are the existential quantified variables and R_i is a predicate symbol in σ, for each $1 \leq i \leq m$. 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 [15]. A 2RPQ E over Σ is a regular expression over $\Sigma^{\pm}$. Intuitively, E computes the pairs of nodes connected by a path that conforms to the regular 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 $p = 0$, $r_1 \cdots r_p = \varepsilon$ and $\theta(x, y)$ becomes $x = y$.

The analogue of CQs in the context of graph databases is the class of *conjunctive* 2RPQs, or C2RPQs [14]. A C2RPQ is a CQ where each atom is a 2RPQ, instead of a symbol in $\sigma(\Sigma^{\pm})$. Formally, a C2RPQ $\gamma(\bar{x})$ over Σ is 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 $\Gamma(\mathcal{G})$ to be $\bigcup_{1 \leq j \leq k} \gamma_j(\mathcal{G})$, 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. IDB(Π) and EDB(Π) denote the set of intensional and extensional predicates, respectively. 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 with schema EDB(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. It is well known that each nonrecursive program can be expressed as a UCQ, via unfolding of the program.

A (nonrecursive) Datalog program over a finite alphabet Σ is a (nonrecursive) Datalog program Π such that $\text{EDB}(\Pi) = \sigma(\Sigma^{\pm})$. The *answer* $\Pi(\mathcal{G})$ of a (nonrecursive) Datalog program Π over a graph database $\mathcal{G}$ over Σ is $\Pi(\mathcal{D}(\mathcal{G}^{\pm}))$.

▶ Remark. Typically, when we analyze a problem involving 2RPQs over Σ, we shall assume that 2RPQs are represented as *one-way nondeterministic finite automata* (1NFA) over alphabet $\Sigma^{\pm}$. This does not affect the complexity of problems as we can translate a regular expression to an equivalent automaton in polynomial time.

3 Regular Queries

We now introduce the class of *Regular Queries* (RQs) and establish some basic results regarding the complexity of evaluation.

▶ **Definition 1. (Regular query)** A *transitive closure rule* is a rule of the form $S(x, y) \leftarrow R^{+}(x, y)$, where S, R are predicate symbols and x, y are variables. We extend the notions of *intensional predicate*, *extensional predicate* and *dependence graph* to a finite set of Datalog and transitive closure rules in the natural way. A *regular query* Ω over a finite alphabet Σ is a finite set of Datalog and transitive closure rules such that:

1. The extensional predicates of Ω belongs to $\sigma(\Sigma^{\pm})$.
2. There is a distinguished intensional predicate called Ans of arbitrary arity.
3. All intensional predicates, except maybe Ans, have arity 2.
4. The dependence graph of Ω is acyclic.

The semantics of regular queries is based on the semantics of Datalog. We define a translation function λ that transforms a Datalog rule or a transitive closure rule into a set of Datalog rules. If ρ is a Datalog rule then $\lambda(\rho) = \{\rho\}$. When ρ is a transitive closure rule of the form $\rho = S(x, y) \leftarrow R^{+}(x, y)$, then $\lambda(\rho)$ contains the rules $\{S(x, y) \leftarrow R(x, y), S(x, y) \leftarrow S(x, z), R(z, y)\}$. We translate each regular query $\Omega = \{\rho_1, \dots, \rho_k\}$ into a Datalog program $\lambda(\Omega) = \lambda(\rho_1) \cup \dots \cup \lambda(\rho_k)$. Then, the *answer* $\Omega(\mathcal{G})$ of a regular query Ω over a graph database $\mathcal{G}$ is the answer of $\lambda(\Omega)$ over $\mathcal{G}$.

▶ **Example 2.** Suppose we have a graph database of persons and its relationships. We have relations *knows*, *is a friend* and *is a good friend*, abbreviated k, f and g, respectively. Thus our alphabet is $\Sigma = \{k, f, g\}$. The following query returns all the pairs of persons connected by a chain of *potential friends*: p and p' are potential friends, if either they are friends, or p just knows p', but they are connected with the same person by a chain of good friends.

$$\begin{array}{lll} G(x,y) \leftarrow g(x,y) & R(x,y) \leftarrow f(x,y) & Ans(x,y) \leftarrow R^{+}(x,y) \\ S(x,y) \leftarrow G^{+}(x,y) & R(x,y) \leftarrow k(x,y), S(x,z), S(y,z) & \end{array}$$

By using ideas from [11], it can be shown that this query cannot be expressed by any UCN2RPQ. ◻

Recall that the *evaluation problem* for regular queries asks, given a RQ Ω, a graph database $\mathcal{G}$ and a tuple $\bar{t}$, whether $\bar{t} \in \Omega(\mathcal{G})$. Each regular query can be translated to a Datalog program, and in fact, this program is *binary* (all intensional predicates have arity 2, except maybe by *Ans*) and *linear* (in the sense of [21]). As a consequence, we can derive tight complexity bound for the evaluation problem [19, 21]. Interestingly, RQs are not harder to evaluate than UCN2RPQs.

▶ **Theorem 3.** *The evaluation problem for regular queries is* NP*-complete in combined complexity and* NLogspace*-complete in data complexity.*

4 Containment of Regular Queries

Given regular queries Ω and Ω' over alphabet Σ, we say that Ω is *contained* in Ω' if $\Omega(\mathcal{G}) \subseteq \Omega'(\mathcal{G})$, for each graph database $\mathcal{G}$ over Σ. The *containment problem* for regular queries asks, given two RQs Ω and Ω', whether Ω is contained in Ω'. We devote the rest of this paper into proving the following theorem:

▶ **Theorem 4.** *The containment problem for regular queries is* 2Expspace*-complete.*

We attack this problem by considering an equivalent language, called *nested* UC2RPQs. As mentioned in the Introduction, each regular query can be unfolded to construct an equivalent exponentially-sized nested UC2RPQ. Thus by considering first nested UC2RPQs, we study the impact of succinctness in the complexity of the containment problem.

4.1 Containment of Nested UC2RPQs

To define formally the class of nested UC2RPQs we introduce a special type of rule. 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 $1 \leq i \leq m$, either R_i is a 2RPQ or R_i is of the form $R_i = P_i^+$, where P_i is a binary predicate symbol. For a finite set Γ of extended C2RPQ rules, we define *intensional predicates* and the *dependence graph* in the obvious way; now the 2RPQs mentioned in Γ play the role of extensional predicates.

▶ **Definition 5. (Nested UC2RPQ)** A *nested UC2RPQ* Γ over Σ is a finite set of extended C2RPQ rules such that:

1. All 2RPQs mentioned in Γ are defined over Σ.
2. There is a distinguished intensional predicate called *Ans* of arbitrary arity.
3. All intensional predicates, except possibly for *Ans*, have arity 2.
4. The dependence graph of Γ is acyclic.
5. For each intensional predicate S, there is a unique occurrence of S over rule bodies of Γ.

Conditions (1)-(4) describe the basic structure of a nested UC2RPQ, and are analogous to that of regular queries. The key condition is Condition (5). It disallows the use of a predicate several times in different parts of the program. If the predicate S was already defined, then S can be used in the body of only one rule, and in the body of that rule, it can be used only once. It is important to note that S can occur several times in the head of rules, that is, it can be defined by more than one rule.

The semantics of nested UC2RPQs is defined in terms of the semantics of regular queries. For each 2RPQ E, let $\delta(E)$ be an equivalent regular query. We define a translation function λ that maps an extended C2RPQ rule to a set of Datalog or transitive closure rules. Let ρ be an extended C2RPQ rule of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$, then $\lambda(\rho)$ contains the following rules:

- One rule of the form $S(x_1, \dots, x_n) \leftarrow P_1(y_1, y'_1), \dots, P_m(y_m, y'_m)$, where $P_1, \dots, P_m$ are distinct fresh predicate symbols.
- For each $1 \leq i \leq m$, if $R_i = E$ is a 2RPQ, then we add rules in $\delta(E)$, where all the predicate symbols in $\delta(E)$ expect by Ans are fresh symbols, and Ans is renamed to P_i.
- If $R_i = Q^+$, for a predicate symbol Q, then we add the rule $P_i(x, y) \leftarrow Q^+(x, y)$.

We translate each nested UC2RPQ $\Gamma = \{\rho_1, \dots, \rho_k\}$ to a regular query $\lambda(\Gamma) = \lambda(\rho_1) \cup \dots \cup \lambda(\rho_k)$. Then, the *answer* $\Gamma(\mathcal{G})$ of a nested UC2RPQ over a graph database $\mathcal{G}$, is the answer of $\lambda(\Gamma)$ over $\mathcal{G}$.

▶ **Example 6.** The following nested UC2RPQ corresponds to the unfolding of the query in Example 2. Observe how the two occurrences of S now become two occurrences of distinct predicates G_1 and G_2.

$$\begin{array}{lll} G_1(x,y) \leftarrow g(x,y) & R(x,y) \leftarrow f(x,y) & Ans(x,y) \leftarrow R^+(x,y) \\ G_2(x,y) \leftarrow g(x,y) & R(x,y) \leftarrow k(x,y), G_1^+(x,z), G_2^+(y,z) & \end{array}$$

In this section, we provide tight complexity bounds for the containment problem for nested UC2RPQs. As it turns out, checking containment of nested UC2RPQs is not harder than checking containment of UC2RPQs.

▶ **Theorem 7.** *The containment problem for nested UC2RPQs is* Expspace*-complete.*

The lower bound holds trivially as containment of UC2RPQs is already Expspace-hard. In order to prove the Expspace upper bound, we use the following approach:

1. We note that containment of nested UC2RPQs can be reduced to containment of boolean nested UC2RPQs.
2. We show that containment of boolean nested UC2RPQs can be reduced to containment of a *boolean* 2RPQ in a boolean nested UC2RPQ. The semantics of a boolean 2RPQ is defined in the obvious way: the result is `true` if the answer of the 2RPQ is nonempty.
3. We prove an Expspace upper bound for containment of a boolean 2RPQ in a boolean nested UC2RPQ.

Step (1) is straightforward (see the Appendix) and makes our subsequent arguments and definitions significantly simpler. Thus, until the end of this section, we focus on boolean queries. When it is clear from the context, we write 2RPQ and nested UC2RPQ, instead of boolean 2RPQ and boolean nested UC2RPQ.

Step (2) is based on a novel *serialization* technique, where we represent expansions of nested UC2RPQs as strings, and modify the original nested UC2RPQs accordingly. For step (3) we combine automata-theoretic techniques [14] with a robust representation of the *partial mappings* from a nested UC2RPQ to an expansion of a 2RPQ.

4.1.1 Reduction to Containment of 2RPQs in nested UC2RPQs

We now show that checking containment of two nested UC2RPQs Γ and Γ' over Σ can be reduced to checking containment of a 2RPQ $\tilde{E}$ in a nested UC2RPQ $\tilde{\Gamma}$ over a larger alphabet Δ. We start by defining the notion of *expansion*, which is central in the analysis of nested UC2RPQs.

Let Γ be a nested UC2RPQ over alphabet Σ and let S be an intensional predicate. We denote by $\mathsf{rules}(S)$ the set of rules in Γ such that S occurs in the head of the rule. An *expansion* φ of S is a CQ with equality over Σ of the form

$$\varphi(x_1, \ldots, x_n) \leftarrow \varphi_1(y_1, y_1'), \ldots, \varphi_m(y_m, y_m')$$

such that there is a rule $\rho \in \mathsf{rules}(S)$ of the form $S(x_1, \ldots, x_n) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$ and the following two conditions hold (note that $n = 0$ if $S = Ans$; otherwise, $n = 2$):

1. For each $1 \leq i \leq m$, if $R_i = E$ is a 2RPQ, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, y_i')$$
where, $p \geq 0$, $r_1 \cdots r_p \in L(E)$, and the z_j's are fresh variables. When $p = 0$, we have that $r_1 \cdots r_p = \varepsilon$, and $\varphi_i(y_i, y_i')$ becomes $y_i = y_i'$.
2. If $R_i = Q^+$ for an intensional predicate Q, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \ldots, \phi_q(w_{q-1}, w_q)$$
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 the CQ 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.

An *expansion* of a nested UC2RPQ is an expansion of its predicate Ans. In particular, any expansion of a nested UC2RPQ is a boolean query. The intuition is that an expansion of a nested UC2RPQ is simply an expansion of its associated Datalog program [19, 16]. Containment of nested UC2RPQs can be characterized in terms of containment of CQs. This is an easy consequence of the semantics of CQs [17, 39] and the fact that each nested UC2RPQ is equivalent to the union of its expansions.

▶ Proposition 8. Let Γ and Γ' be two nested UC2RPQs. 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 [17], due to the presence of inverses:

▶ **Definition 9.** 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 two nested UC2RPQs Γ and Γ' over Σ, we shall construct a 2RPQ $\tilde{E}$ and a nested UC2RPQ $\tilde{\Gamma}$ such that Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$. Our reduction is based on two ideas:

1. Expansions of Γ can be "serialized" and represented by *serialized expansions*, which are strings over a larger alphabet Δ. More importantly, serialized expansions constitute a regular language. Thus, we can construct a 2RPQ $\tilde{E}$ such that $L(\tilde{E})$ is precisely the set of serialized expansions of Γ. This technique has been already used before [14, 15].
2. Now we need to serialize Γ'. Proposition 8 basically tell us that Γ is contained in Γ' iff Γ' can be "mapped" to each expansion of Γ. We have replaced Γ by $\tilde{E}$. Thus, expansions of Γ are replaced by serialized expansions. By modifying the 2RPQs mentioned in Γ', we construct a nested UC2RPQ $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to a serialized expansion

W of Γ iff Γ' can be mapped to the expansion of Γ represented by W. This is a novel technique and constitutes the crux of the reduction.

Next we develop these two ideas. Due to space limitations, we only present the main ideas and intuitions. Details can be found in the Appendix.

Serialization of Γ

We can represent each expansion φ of Γ, by a serialized expansion, that is, a string over a suitable alphabet Δ. Suppose first that Γ is simply a C2RPQ, that is, a single rule of the form $Ans() \leftarrow E_1(y_1, y_1'), \dots, E_m(y_m, y_m')$. Then, an expansion φ corresponds to choose a string $r^i = r_1^i, \dots, r_{p_i}^i \in L(E_i)$, for each $1 \leq i \leq m$. This can be represented by the string (a similar representation is used in [14])

$$\$ y_1 r_1^1 \cdots r_{p_1}^1 y_1' \$ y_2 r_1^2 \cdots r_{p_2}^2 y_2' \$ \cdots \$ y_m r_1^m \cdots r_{p_m}^m y_m' \$$$

Thus, the alphabet Δ contains $\Sigma^{\pm}$, a separator \$, and the variable set of Γ. Assume now that Γ consists of two rules of the form

$$P(x,y) \leftarrow F_1(t_1, t_1'), \dots, F_n(t_n, t_n') \qquad Ans() \leftarrow P^+(y_1, y_1'), E_2(y_2, y_2'), \dots, E_m(y_m, y_m')$$

Suppose φ is the expansion of Γ resulting from choosing strings $r^i \in L(E_i)$, for each $2 \leq i \leq m$, and for the atom $P^+(y_1, y_1')$, choosing two intermediate variables z_1 and z_2, and three expansions $\varphi_1, \varphi_2, \varphi_3$ of P. Then, we represent φ by the string

$$\$ y_1 W_1 \star \$ \star W_2 \star \$ \star W_3 y_1' \$ y_2 r^2 y_2' \$ \cdots \$ y_m r^m y_m' \$$$

where W_1, W_2, W_3 are strings representing $\varphi_1, \varphi_2, \varphi_3$, respectively, as defined before. Now, the alphabet Δ contains additionally the separator $\star$ that represents fresh intermediate variables that appear when we expand a transitive closure atom.

Applying this idea recursively, it is easy to define serialized expansions for a general nested UC2RPQ Γ. Additionally, it can be shown that serialized expansions can be detected by a 1NFA $\tilde{E}$ over Δ. Moreover, we can construct $\tilde{E}$ such that its size is polynomial in the size of Γ. We say that $\tilde{E}$ is the *serialization* of Γ.

Serialization of Γ'

We replaced Γ by $\tilde{E}$. Thus we replaced expansions of Γ by expansions of $\tilde{E}$, which are of the form $\theta^W() \leftarrow w_1(x_0, x_1), w_2(x_1, x_2), \dots, w_n(x_{n-1}, x_n)$, for a serialized expansion $W = w_1 \cdots w_n$. Suppose W represents an expansion φ of Γ. Our goal is to construct $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to θ^W iff Γ' can be mapped to $\mathsf{neq}(\varphi)$. To construct $\tilde{\Gamma}$, we modify Γ' in order to translate mappings from Γ' to $\mathsf{neq}(\varphi)$, into mappings from $\tilde{\Gamma}$ to θ^W (and vice versa). Next we explain the main difficulties in the construction of $\tilde{\Gamma}$.

Let W be a serialized expansion representing an expansion φ of Γ. Each symbol in W (except the separator \$) represents a variable in φ. We denote by $\mathsf{var}(i)$ the variable represented by the i-th symbol of W. On the other hand, equality atoms in φ define equivalence classes over the variables of φ. We write $y \equiv_\varphi y'$ when the variables y, y' belong to the same equivalence class. Thus it could be possible that $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$, or even that $\mathsf{var}(i) = \mathsf{var}(j)$, for two distinct positions $i < j$ in W. This implies that $\mathsf{var}(i)$ and $\mathsf{var}(j)$ are renamed exactly to the same variable in $\mathsf{neq}(\varphi)$. Thus, in order to simulate mappings from Γ' to $\mathsf{neq}(\varphi)$, we have to consider positions i and j as equivalent, that is, we must be able to "jump" between positions i and j, whenever necessary.

To overcome this problem, we introduce the notion of *equality string*. Equality strings are strings over $\Delta^{\pm}$ with the following key property. For positions $1 \leq i < j \leq |W|$, $\mathsf{var}(i) \equiv_{\varphi} \mathsf{var}(j)$ iff there is an equality string that can be "folded" into W from i to j. Intuitively, a string α can be folded into W if α 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. For instance, consider the string $W = \$y_1b^-ay_2\$$. Then, $by_1^-y_1b^-aa^-$ can be folded into W from 3 to 4, and $b^-aa^-ay_2\$$ can be folded into W from 3 to 6.

Next we give some intuition about equality strings (the precise definition is given in the appendix). Equality strings are concatenations of *basic* equality strings. There are several types of basic equality strings, each one detecting a particular type of connection between variables. For example consider a serialized expansion of the form

$$\$x \cdot a \cdot y\$x \cdot z\$w \cdot \$t \cdot b \cdot s\$ \cdot z\$w \cdot aab \cdot u\$$$

where the alphabet is $\{a, b\}$. The substring $\$t \cdot b \cdot s\$$ is a "sub" serialized expansion generated due to a transitive closure atom. Thus the first occurrence of x is equivalent to the last occurrence of z. This is witnessed by an "horizontal" equality string $x \cdot a \cdot y\$x \cdot z\$w \cdot \$t \cdot b \cdot s\$ \cdot z$, that is, a string satisfying the regular expression $x\Delta^*xz\Delta^*z$. If t is the first variable of the serialized expansion $\$t \cdot b \cdot s\$$, then w and t are also equivalent. This is witnessed by a "downward" equality string $w\$t$. In these two examples, we need to know that x and z are in the same "level" and that the level of t is the level of w minus 1. In order to achieve this, we incorporate levels to the symbols in Δ, thus the actual definition of Δ is slightly more involved that the one presented here.

Now we are ready to serialize the nested UC2RPQ Γ'. Let $w = w_1 \cdots w_p$ be a string over $\Sigma^{\pm}$. The *serialization* of w, denoted by $\mathsf{serial}(w)$, is the set of 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 $0 \leq i \leq p$, the string α_i is either ε or an equality string. If L is a language over $\Sigma^{\pm}$, then $\mathsf{serial}(L)$ is the language over $\Delta^{\pm}$ defined by $\mathsf{serial}(L) = \{w' \mid w' \in \mathsf{serial}(w),\ \text{for some } w \in L\}$. It can be shown that if $\mathcal{A}$ is a 1NFA accepting the language L over $\Sigma^{\pm}$, then there exists a 1NFA $\mathcal{A}'$ over $\Delta^{\pm}$ accepting precisely $\mathsf{serial}(L)$. Moreover, the size of $\mathcal{A}'$ is polynomial in the size of $\mathcal{A}$ and Δ.

The *serialization* $\tilde{\Gamma}$ of Γ' is the nested UC2RPQ over Δ obtained from Γ' by replacing each 2RPQ E in Γ' by $\mathsf{serial}(E)$. It is important to note that the serialization $\tilde{\Gamma}$ of Γ' depends on both Γ and Γ'. This is because it depends on Δ (which, at the same time, depends on Γ). Observe also that the size of $\tilde{\Gamma}$ is polynomial in the size of Δ and Γ', and thus polynomial in the size of Γ and Γ'. Furthermore, $\tilde{E}$ and $\tilde{\Gamma}$ can be constructed in polynomial time from Γ and Γ'. The following proposition concludes our reduction.

▶ Proposition 10. Let $\tilde{E}$ and $\tilde{\Gamma}$ be the serialization of Γ and Γ', respectively. Then, Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$.

The idea behind Proposition 10 is as follows. Suppose Γ is contained in Γ'. Let $\theta^W() \leftarrow w_1(x_1, x_2), w_2(x_3, x_4), \ldots, w_n(x_n, x_{n+1})$ be an expansion of $\tilde{E}$, where $W = w_1 \cdots w_n$ is a serialized expansion representing φ. We know that there is a containment mapping μ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$, for some expansion φ' of Γ'. We have to find an expansion ψ of $\tilde{\Gamma}$ and a containment mapping ν from $\mathsf{neq}(\psi)$ to θ^W. The structure of $\mathsf{neq}(\psi)$ and $\mathsf{neq}(\varphi')$ is the same, except for the strings chosen when we expand 2RPQs. The *internal variables* of an expansion are the fresh variables that appears when we expand 2RPQs (the z_j's in the definition of expansion). The rest of the variables are *external variables.* Thus, the idea is that the external variables of $\mathsf{neq}(\psi)$ and $\mathsf{neq}(\varphi')$ coincide, but the internal variables differ according to the string chosen in both expansions. Now, for each external variable t in

$\mathsf{neq}(\psi)$ we define $\nu(t)$ as follows. Let $s = \mu(t)$ (we can apply μ to t as t is also an external variable of $\mathsf{neq}(\varphi')$). Let y be a variable in φ that is renamed to s in the construction of $\mathsf{neq}(\varphi)$. Then, we define $\nu(t)$ to be x_j, for some $1 \leq j \leq n$ such that $\mathsf{var}(j) = y$.

To conclude, we need to define the expansions of 2RPQs in $\mathsf{neq}(\psi)$ and extend ν to the internal variables. Suppose that in $\mathsf{neq}(\varphi')$ we expand a 2RPQ E, between external variables t and t', into the CQ $a_1(t, z_2), a_2(z_2, z_3), \ldots, a_k(z_k, t')$. We want to define an expansion $b_1(t, o_2), b_2(o_2, o_3), \ldots, b_\ell(o_k, t')$ of $\mathsf{serial}(E)$ and extend ν to $\{o_2, \ldots, o_k\}$ ($\nu(t)$ and $\nu(t')$ are already defined). This amounts to finding a folding of $B = b_1 \cdots b_\ell$ into W from i to j, where $\nu(t) = x_i$ and $\nu(t') = x_j$. We define B and this folding simultaneously. We start with $B = a_1 \cdots a_k$ and analyze B from left to right. We examine the values $\mu(t), \mu(z_2), \ldots, \mu(z_k), \mu(t')$ in that order, and according to these values we fold B into W. If all the values are internal variables, there is no problem: we can easily fold B into W. If we see an external variable, we have a problem: we need to "jump" to an equivalent position in order to continue our folding. Thus, we can interleave a suitable equality string α so we can continue our folding into W. In this way, we end up with a string $B \in L(\mathsf{serial}(E))$ (since we only interleave equality strings with $a_1 \cdots a_k \in L(E)$) and with a folding of B into W, as required. The other direction of the proposition is analogous.

4.1.2 Containment of 2RPQs in nested UC2RPQs: Upper Bound

Next we show that containment of a 2RPQ in a nested UC2RPQ is in Expspace. We exploit automata-theoretic techniques along the lines of [14, 19, 16]. The main idea is to reduce the complement of the containment problem of a 2RPQ E in a nested UC2RPQ Γ to the non-emptiness of a suitable doubly exponential-sized 1NFA $\mathcal{A}$. The crux of the construction is an intermediate automaton $\mathcal{A}_\Gamma$ that accepts all the (potential) expansions θ of E such that there is a mapping from Γ to θ.

In [14], it is shown that checking containment of two UC2RPQs Γ' and Γ is in Expspace. To construct the automaton $\mathcal{A}_\Gamma$, that accepts expansions θ of Γ' such that Γ can be mapped to θ, the authors exploit *two-way* NFAs (2NFA). Moreover, they annotate the expansions with variables of Γ, which indicate the potential mapping of Γ to the expansion. The number of possible annotations is bounded, as the number of variables is bounded. It is by no means obvious how to extend these techniques to the case when Γ is a nested UC2RPQ, as the number of variables involved in a mapping from Γ to θ is not bounded anymore. Thus we follow a different approach: We work directly with 1NFAs. We construct $\mathcal{A}_\Gamma$ directly as a 1NFA. 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.

Note that the expansions for a 2RPQ E over Σ are CQs of a very particular form, that we call *linear CQs*: they are sequences $r_1(z_1, z_2), \ldots, r_p(z_p, z_{p+1})$ where $r_1 \cdots r_p \in L(E)$ and each z_j is distinct. Thus if we want to decide whether a 2RPQ E is contained in a nested UC2RPQ Γ, we need only to look at those expansions of Γ that can be *flattened* into a linear CQ, i.e., those that can actually be mapped to some linear CQ. Formally, given an expansion φ of Γ, a *linearization* of φ is a linear CQ θ such that there is a containment mapping from φ to θ. Furthermore, the set of linearizations of Γ is the union of all linearizations of all the expansions of Γ. As we mentioned before, the idea of this proof is to show that the set of linearizations of a nested UC2RPQ Γ can be characterized by an 1NFA $\mathcal{A}_\Gamma$.

The *depth* of a nested UC2RPQ is the maximum length of a directed path from some 2RPQ to the *Ans* predicate in its dependence graph, minus 1. For instance, the query in Example 6 has depth 2.

Cuts. Let Γ be a nested UC2RPQ of depth 0, defined by the rules

$$\begin{array}{rcl} Ans(x_1,\dots,x_n) & \leftarrow & \Gamma^1_1(u^1_1,v^1_1),\dots,\Gamma^1_{m_1}(u^1_{m_1},v^1_{m_1}), \\ \vdots & & \vdots \\ Ans(x_1,\dots,x_n) & \leftarrow & \Gamma^\ell_1(u^\ell_1,v^\ell_1),\dots,\Gamma^\ell_{m_\ell}(u^\ell_{m_\ell},v^\ell_{m_\ell}), \end{array} \tag{1}$$

Let $\mathcal{A}^i_j$ be a 1NFA associated with the 2RPQ Γ^i_j, for each $1 \leq i \leq \ell$ and $1 \leq j \leq m_i$. Let $Vars(\Gamma, i)$ be the set of variables appearing in the i-th rule of query (1) above. A *cut* of Γ is an ℓ-tuple $(C^1,\dots,C^\ell)$, where each of the C^i's is either $\perp$ or a triple of form $(\text{Prev}^i, \text{Same}^i, \mathcal{S}^i)$, with $\text{Same}^i \subseteq \text{Prev}^i \subseteq Vars(\Gamma, i)$ and where $\mathcal{S}^i$ is an m_i-tuple $\mathcal{S}^i = (s^i_1,\dots,s^i_{m_i})$ containing a state of each of the A^i_js, $1 \leq j \leq m_i$.

Using the definition of cuts for UC2RPQs, we can naturally extend the definition for cuts of queries of depth > 0. Assume that the rules in Γ with the answer predicate in the head follow form (1) above (note that now some of the Γ^i_js might be predicates P^+ instead of RPQs). Then cuts for Γ are again ℓ-tuples $(C^1,\dots,C^\ell)$, the only thing that changes is the definition of $\mathcal{S}^i = (s^i_1,\dots,s^i_{m_i})$, for the cases when Γ^i_j is not a 2RPQ but a predicate of form $P^+(x,y)$. In this case, s^i_j is a cut of query P. Furthermore, *initial cuts* are those in which each Prev^i is empty, and *final cuts* are those in which at least one of the Prev^is is equal to $Vars(\Gamma, i)$. A cut *marks* a variable x if in all C^is that are not $\perp$ we have that x belongs to Same^i.

Let us now give some intuition on the notion of cuts. Suppose Γ is a C2RPQ and μ is a mapping from Γ to a linearization θ of Γ. If we look at position k in θ, then the partial mapping of μ until this position can be represented by a cut $(\text{Prev}, \text{Same}, \mathcal{S})$: Prev are the variables that are mapped to positions smaller than k and Same are the variables that are mapped precisely to position k. The intuition of $\mathcal{S}$ is as follows. We know that 2RPQs of the form $F(y,y')$ with $y,y' \in \text{Prev} \cup \text{Same}$ are satisfied. The only information that is missed is that of the 2RPQs that are "cut" by $\text{Prev} \cup \text{Same}$, that is, the 2RPQs of the form $F(y,y')$ such that $y \in \text{Prev} \cup \text{Same}$ and $y' \notin \text{Prev} \cup \text{Same}$ (or vice versa). Suppose that μ expands F in a string $r_1 \cdots r_p$ and let $s_1,\dots,s_{p+1}$ be an accepting run of F over $r_1 \cdots r_p$. Consider the mapping μ over $r_1 \cdots r_p$ and suppose r_j is the last symbol to be mapped in positions smaller that k. Then $\mathcal{S}$ contains the state s_j, for each cut 2RPQ $F(y,y')$. Note that this is the only information we need to extend this partial mapping to the one at position $k+1$, and eventually to the global mapping μ. Since we are dealing with UC2RPQs we also need to account for the case when a certain disjunct of Γ cannot be mapped to a linearization: in this case, the corresponding triple of the cut is set to $\perp$.

The notion of cut is crucial for two reasons. First, transitions between cuts can be captured by a 1NFA $\mathcal{A}_\Gamma$. Second, it is easy to see that the size of each cut is polynomial in the size of Γ (here we use the fact that Γ is a nested UC2RPQ, instead of a regular query). This implies that the set of all cuts, denoted by $Cuts(\Gamma)$, is of exponential size in the size Γ. This is important to obtain our desired EXPSPACE upper bound.

Transition system based on cuts

Looking to characterize the set of linearizations of nested UC2RPQs, our next step is to define a transition system $T_{(\Gamma,w)}$ defined over cuts of Γ and positions of a word w over $\Sigma^\pm$, i.e., over pairs from $Cuts(\Gamma) \times \{1,\dots,p+1\}$.

For space reasons, we leave the definition of the system to the appendix. Instead, 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 in Prev, that can be mapped to each position in w. For the same reason, the transition system can only move towards configurations in which Prev is not smaller that previous configurations.

▶ **Example 11.** Consider query $\Gamma(x,y) \leftarrow g^+(x,z), g^+(y,z)$, stating that there is a path labeled with g^+ between both x and z, and y and z. It is not difficult to see that the CQ $q = g(x,x'), g(x',z), g(y',z), g(y,y')$ is a linearization of Γ. The string associated to q is $w = ggg^-g^-$. A valid run for $T_{(\Gamma,w)}$ over w starts in the initial cut in position 1 . It moves to cut $(\{x\},\{x\},\mathcal{S})$, where $\mathcal{S}$ only needs to store the state where the automaton for query g^* is, after having read a g. We then advance to cut $(\{x,z\},\{z\},\mathcal{S})$ in position 3, $(\{x,z\},\{\},\mathcal{S})$ in position 4, where now $\mathcal{S}$ again needs to store a state of the automaton for g^+, and finally to cut $(\{x,z,y\},\{y\},\mathcal{S})$ in position 5. Since this last cut is final, we determine that $T_{(\Gamma,w)}$ can advance form an initial state to a final state.

There is a technicality when formally defining this intuition, as variables x and y of Γ need not be mapped right at the start or the end of the the computation of $T_{(\Gamma,w)}$. Thus to state the correctness of this system we need to define a special type of run. Formally, given a nested UC2RPQ $\Gamma(x,y)$ and a word w, then (C,p) and (C',p') define an *accepting run* for Γ over w if the following holds:

- C marks x and C' marks y
- There is an initial cut C_I and a position p_I of w such that one can go from (C_I,p_I) to (C,p) by means of $T_{(\Gamma,w)}$.
- There is a final cut C_F and a position p_F of w such that one can go from (C',p') to (C_F,p_F) by means of $T_{(\Gamma,w)}$.

The following lemma states the correctness of our system.

▶ **Lemma 12.** *Let $\Gamma(x,y)$ be a nested UC2RPQ and $w = r_1 \cdots r_p$ a string over Σ. There are pairs (C,i) and (C',i') over Cuts$(\Gamma) \times \{1,\dots,p+1\}$ that define an* accepting run *for Γ over w if and only if there is an expansion φ of Γ and a containment mapping from φ to $Q_w = r_1(z_1,z_2),\dots r_p(z_p,z_{p+1})$ that maps x and y to variables z_i and $z_{i'}$ of Q_w.*

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 an extra blowup in our algorithm. In a nutshell, given Γ and w we have to extend the symbols of w with information about the cuts reachable from configurations of the form (c,p) in $T_{(\Gamma,w)}$, where $1 \leq p \leq |w|$.

Formally, from $\Sigma^{\pm}$ we construct the extended alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$ as follows. Assume that C*uts*(Γ) contains a number N of cuts. Then

- If Γ is a nested UC2RPQ of depth 0, $\Sigma(\Gamma)$ is an $N+2$ tuple of subsets of C*uts*(Γ), i.e., $\Sigma(\Gamma) = (2^{Cuts(\Gamma)})^{N+2}$.
  In other words, $\Sigma(\Gamma)$ contains a subset of C*uts*(Γ) for each cut in C*uts*(Γ).
- Otherwise assume that Γ is of form (1), and that $\mathcal{P}$ is the set of predicates occurring in the rules of Γ that are not 2RPQs. Then $\Sigma(\Gamma) = (2^{Cuts(\Gamma)})^{N+2} \times \mathsf{X}_{P\in\mathcal{P}}\,\Sigma(P)$. In other words, it is the cartesian product of the set $(2^{Cuts(\Gamma)})^{N+2}$ with each $\Sigma(P)$, for every predicate P that is a subquery of Γ.

Let us now give some intuition regarding this construction. Let w be a string from $\Sigma(\Gamma) \times \Sigma^{\pm}$, and let $\tau(w)$ be its projection over $\Sigma^{\pm}$. Each symbol (u_p,a_p), for $u_p \in \Sigma(\Gamma)$

contains, in particular, $N+2$ subsets of $Cuts(\Gamma)$, where $N = |Cuts(\Gamma)|$. The first subset of $Cuts(\Gamma)$ represent all those cuts C such that there is a position $\hat{p}$ in w and an initial cut $\hat{C}$ of Γ such that (C,p) is reachable from $(\hat{c},\hat{p})$ using $T_{(\Gamma,\tau(w))}$. The second subset of $Cuts(\Gamma)$ contains all those cuts C such that there is a final cut $\hat{C}$ and a position $\hat{p}$ so that $(\hat{C},\hat{p})$ can be reached from (C,p) using $T_{(\Gamma,\tau(w))}$. We have N subsets remaining, namely one for each cut C of Γ. For each such cut C, its corresponding subset $\mathcal{C}$ contains all those cuts $\hat{C}$ for which the configuration $(\hat{C},p)$ is reachable from (C,p) using $T_{(\Gamma,\tau(w))}$.

If a string w from $\Sigma(\Gamma) \times \Sigma^{\pm}$ satisfies the above conditions, we say that w has *valid annotations* w.r.t. Γ. We show:

▶ **Lemma 13.** *Let Γ be a nested U2CRPQ. Then the number of different symbols in $\Sigma(\Gamma)$ is double exponential w.r.t. the size of Γ. Furthermore, each symbol in $\Sigma(\Gamma)$ is of size exponential w.r.t. Γ.*

▶ **Lemma 14.** *Let Γ be a nested U2CRPQ. Then the language of all strings over $\Sigma(\Gamma) \times \Sigma^{\pm}$ that have* valid annotations *w.r.t. Γ is regular. Furthermore, one can build an 1NFA that checks this language of size double-exponential in the size of Γ.*

We can finally proceed to build the desired 1NFA A_Γ that gives the strings corresponding to the linearizations of a nested UC2RPQ Γ. This 1NFA needs to simulate the system $T_{(\Gamma,w)}$, from an initial cut of a nested UC2RPQ Γ to a final cut in which all variables have already been mapped. Of course, we have to check, for every subquery $\Gamma^i_j(u^i_j, v^i_j)$ of Γ, whether this query is indeed satisfied when starting in the position assigned to variable u^i_j and finishing on the position assigned to v^i_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 15.** *Given a nested UC2RPQ Γ over Σ, one can construct, in exponential time, a 1NFA A_Γ over alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$ such that A_Γ accepts a string $w = r_1, \dots, r_p$ with valid annotations w.r.t. Γ if and only if there are pairs (C,i) and (C',i') over $Cuts(\Gamma) \times \{1,\dots,p+1\}$ that define an accepting run for Γ over w.*

Main Proof. To decide containment of a 2RPQ E in a nested UC2RPQ Γ, we proceed as follows:
- Build an 1NFA $\mathcal{A}_E$ for E, extended so that it works with the alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$.
- Build the 1NFA $\mathcal{A}_{\Sigma(\Gamma)}$ that checks for strings over $\Sigma(\Gamma) \times \Sigma^{\pm}$ that are valid w.r.t. Γ.
- Build the 1NFA $\mathcal{A}_\Gamma$, and complement it, obtaining the automaton $(\mathcal{A}_\Gamma)^C$.

The language of $(\mathcal{A}_\Gamma)^C$ intersected with the language of $\mathcal{A}_{\Sigma(\Gamma)}$ is precisely those strings w with valid annotations such that its projection $\tau(w)$ over $\Sigma^{\pm}$ does not correspond to any linearization of Γ. Thus, if we intersect this language with the one of $\mathcal{A}_E$, we have that the resulting intersection is nonempty if and only if there is an expansion q for E that does not correspond to any of the linearizations of Γ, i.e., if E is not contained in Γ.

Even though some of these automata can be of doubly exponential size w.r.t. E and Γ, we can perform this algorithm in EXPSPACE using a standard on-the-fly implementation.

4.2 Containment of Regular Queries: upper and lower bounds

Expressing a regular query as a nested UC2RPQ may involve an exponential blow up in size. Next we formalize this. Analogous to the case of nested UC2RPQs, 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. For instance, the query in Example 2 has depth 2. The *height* of a RQ or a nested UC2RPQ is the maximum size of $\mathsf{rules}(S)$ over all intensional predicates S. Recall that $\mathsf{rules}(S)$ is the set of rules whose heads mention the predicate S. Finally, the *width* of a RQ or a nested UC2RPQ is the maximum size of a rule body.

▶ Proposition 16. Let Ω be a regular query. Let h, w and d be the height, the width and the depth of Ω, respectively. Then, Ω is equivalent to a nested UC2RPQ Γ of height at most $h^{O(w^d)}$, width at most w^{d+1}, and depth at most d. In particular, the number of rules in Γ is double-exponential in the size of Ω.

In view of this result, we cannot use Theorem 7 directly to show a 2EXPSPACE upper bound for the containment problem of two regular queries, as unfolding a regular query Ω may result in a nested UC2RPQ Γ that is of double-exponential size with respect to Ω, and thus the number of cuts in Γ might be of triple-exponential size with respect to Ω.

However, we can further refine the construction we have shown so that the number of cuts depends exponentially only in the width and depth of Γ, but not on the height. The idea is to define cuts of nested UC2RPQs not as a tuple $(C^1, \ldots, C^\ell)$ for each of the ℓ rules in $\mathsf{rules}(Ans)$, but rather as a single triple $(\text{Prev}, \text{Same}, \mathcal{S})$. Intuitively, this corresponds to guessing a priori which disjunct or rule is to be used when witnessing linearizations of Γ. Using this modified construction and Proposition 16 we can then show that the number of cuts is again double-exponential in the size of Ω, which immediately leads to a 2EXPSPACE algorithm, following the ideas of the proof of Theorem 7.

Moreover, by combining techniques from [19, 14], we can show a matching lower bound for containment of regular queries.

▶ **Theorem 17.** *The containment problem for regular queries is* 2EXPSPACE*-hard.*

5 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 constitutes a well-behaved class that deserve further investigation.

There are several realistic restrictions on regular queries that lead to better complexity bounds. For instance, it is easy to see that RQs of *bounded treewidth* [20, 25] can be evaluated in polynomial time in the size of the query and the database. For $k \geq 1$, a RQ has treewidth at most k, if the *underlying graph* of each rule has treewidth at most k. Thus the good behavior of bounded treewidth C2RPQs [6] extends to regular queries. Another natural restriction is that of bounded depth. As a corollary of our results in Section 4, we have that containment for RQs of bounded depth is EXPSPACE-complete. This is very interesting, as in many situations it may be natural to express regular queries as nested UC2RPQs or to consider regular queries of small depth. In these cases, checking containment is EXPSPACE-complete, the same as for UC2RPQs. Note also that from Theorem 7, it follows that containment of UCN2RPQs is EXPSPACE-complete, which was not known to date.

An interesting research direction is to study the containment problem of a Datalog program in a regular query. Decidability of this problem follows from [22, 23], 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 RQs, we conjecture that this problem is elementary.

References

1 R. Angles, C. Gutierrez. Survey of graph database models. In *ACM Comput. Surv.*, 40(1):1–39, 2008.

2 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

3 M. Arenas, J. Peréz. Querying semantic web data with SPARQL. In *PODS* 2011, pages 305–316.

4 M. Arenas, C. Gutierrez, D. Miranker, J. Peréz, J. Sequeda. Querying Semantic Data on the Web? *SIGMOD Record*, 41(4): 6–17 (2012).

5 P. Barceló. Querying graph databases. In *PODS* 2013, pages 175–188.

6 P. Barceló, L. Libkin, A. Lin, P. Wood. Expressive languages for path queries over graph-structured data. In *ACM TODS* 38(4), 2012.

7 P. Barceló, J. Pérez, J. Reutter. Relative Expressiveness of Nested Regular Expressions. In *AMW 2012*, pages 180–195.

8 M. Bienvenu, D. Calvanese, M. Ortiz, M. Simkus. Nested regular path queries in description logics. In *KR* 2014.

9 M. Bienvenu, M. Ortiz, M. Simkus. Conjunctive regular path queries in lightweight description logics. In *IJCAI* 2013, pages 761–767.

10 P. Bourhis, M. Krotzsch, S. Rudolph. Query containment for highly expressive datalog fragments. CoRR abs/1406.7801 (2014).

11 P. Bourhis, M. Krotzsch, S. Rudolph. How to Best Nest Regular Path Queries. In DL 2014, Poster.

12 P. Buneman. Semistructured data. In *PODS* 1997, pages 117–121.

13 P. Buneman, S. B. Davidson, G. G. Hillebrand, D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD* 1996, pages 505-516.

14 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.

15 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

16 D. Calvanese, G. de Giacomo, M. Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.* 336(1), pages 33–56, 2005.

17 A. Chandra, Ph. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC* 1977, pp. 77–90.

18 S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. In *ICDE* 1995, pages 190-200.

19 S. Chaudhuri, M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *J. Comput. Syst. Sci.* 54(1), pages 61–78, 1997.

20 C. Chekuri, A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.* 239(2), pages 211–229, 2000.

21 M. Consens, A. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS* 1990, pages 404–416.

22 B. Courcelle. The monadic second-order theory of graphs I – Recognizable sets of finite graphs. *Inform. and Comput.,* 85 (1972), pp. 263–267.

23 B. Courcelle. Recursive queries and context-free graph grammars *Theoret. Comput. Sci.*, 78 (1991), pp. 217–244.

24 I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD Record*, 16(3):323–330, 1987.

25 V. Dalmau, P. Kolaitis, M. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP* 2002, pp. 310–326.

26 R. Fagin, M. Y. Vardi. The theory of data dependencies - An overview. In *ICALP* 1984, pages 1–22.

27 W. Fan. Graph pattern matching revised for social network analysis. In *ICDT* 2012, pages 8–21.

28 M. F. Fernández, D. Florescu, A. Y. Levy, D. Suciu. Verifying integrity constraints on web sites. In *IJCAI* 1999, pages 614–619.

29 G. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, Y. Wu. Relative expressive power of navigational querying on graphs. In *ICDT* 2011, pages 197–207.

30 D. Florescu, A. Levy, A. Mendelzon. Database techniques for the World-Wide-Web: A survey. In *SIGMOD Record*, 27(3):59–74, 1998.

31 M. Friedman, A. Y. Levy, T. D. Millstein. Navigational plans For data integration. In *AAAI/IAAI* 1999, pages 67–73.

32 T. Imielinski, W. Lipski Jr. Incomplete information in relational databases. *J. of the ACM* 31(4), pages 761–791, 1984.

33 E. Kostylev, J. Reutter, D. Vrgoc. Containment of Data Graph Queries. In *ICDT* 2014, pages 131–142.

34 Z. Lacroix, H. Murthy, F. Naumann, L. Raschid. Links and paths through life sciences data Sources. In *DILS* 2004, pages 203–211.

35 L. Libkin, W. Martens, D. Vrgoc. Querying graph databases with XPath. In *ICDT* 2013, pages 129–140.

36 L. Libkin, J. Reutter, D. Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In *PODS* 2013, pages 201–212.

37 J. Perez, M. Arenas, C. Gutierrez. nSPARQL: A navigational language for RDF. In *J. of Web Semantics* 8, 255–270 (2010).

38 S. Rudolph, M. Krotzsch. Flag & check: data access with monadically defined queries. In *PODS* 2013, pages 151–162.

39 Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operator. In *J. of the ACM* 27(4), 1980, pages 633–655.

A Proofs and Intermediate Results

A.1 Reduction to boolean nested UC2RPQs

▶ **Theorem 18.** *There is a polynomial time reduction from the containment problem for nested UC2RPQs to the containment problem for boolean nested UC2RPQs.*

Proof: Let Γ and Γ' be two nested UC2RPQs over alphabet Σ. Suppose Γ and Γ' have free variables $x_1, \dots x_n$, with $n \geq 1$. Let $\$_1, \dots, \$_n$ be n fresh symbols that are not in Σ and consider the alphabet $\Sigma' = \Sigma \cup \{\$_1, \dots, \$_n\}$. We define the boolean nested UC2RPQ $b(\Gamma)$ as follows: Start with Γ and replace each rule of the form $Ans(x_1, \dots, x_n) \leftarrow R_1(y_1, y_1'), \dots, R_m(y_m, y_m')$, by a rule $Ans() \leftarrow R_1(y_1, y_1'), \dots, R_m(y_m, y_m'), \$_1(x_1, x_1), \dots, \$_n(x_n, x_n)$. Similarly, we define $b(\Gamma')$. It is straightforward to show that Γ is contained in Γ' iff $b(\Gamma)$ is contained in $b(\Gamma')$. ◻

A.2 Reduction to Containment of 2RPQs in nested UC2RPQs

We now show that checking containment of two nested UC2RPQs Γ and Γ' over Σ can be reduced to checking containment of a 2RPQ $\tilde{E}$ in a nested UC2RPQ $\tilde{\Gamma}$ over a larger alphabet Δ. We start by defining the notion of *expansion*, which is central in the analysis of nested UC2RPQs.

Let Γ be a nested UC2RPQ over alphabet Σ and let S be an intensional predicate. We denote by $\mathsf{rules}(S)$ the set of rules in Γ such that S occurs in the head of the rule. An *expansion* φ of S is a CQ with equality over Σ of the form

$$\varphi(x_1, \dots, x_n) \leftarrow \varphi_1(y_1, y_1'), \dots, \varphi_m(y_m, y_m')$$

such that there is a rule $\rho \in \mathsf{rules}(S)$ of the form $S(x_1, \dots, x_n) \leftarrow R_1(y_1, y_1'), \dots, R_m(y_m, y_m')$ and the following two conditions hold (note that $n = 0$ if $S = Ans$; otherwise, $n = 2$):

1. For each $1 \leq i \leq m$, if $R_i = E$ is a 2RPQ, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \dots, r_p(z_{p-1}, y_i')$$
where, $p \geq 0$, $r_1 \cdots r_p \in L(E)$, and the z_j's are fresh variables. When $p = 0$, we have that $r_1 \cdots r_p = \varepsilon$, and $\varphi_i(y_i, y_i')$ becomes $y_i = y_i'$.
2. If $R_i = Q^+$ for an intensional predicate Q, then $\varphi_i(y_i, y_i')$ is a CQ with equality of the form
$$\varphi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \dots, \phi_q(w_{q-1}, w_q)$$
where $q \geq 1$, $w_0 = y_i$, $w_q = y_i'$, $w_1, \dots, w_{q-1}$ 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 the CQ 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.

An *expansion* of a nested UC2RPQ is an expansion of its predicate Ans. In particular, any expansion of a nested UC2RPQ is a boolean query. The intuition is that an expansion of a nested UC2RPQ is simply an expansion of its associated Datalog program [19, 16]. Each time we expand a 2RPQ in the case (1) of the definition of expansion, we generate new fresh variables z_j's. We call these variables the *internal* variables of the expansion. The rest of the variables are the *external* variables. Example 19 illustrates these concepts.

▶ **Example 19.** *Consider a boolean nested UC2RPQ over* $\Sigma = \{f, k, g\}$*:*

$$
\begin{aligned}
R(x,y) &\leftarrow f + \varepsilon(x,y)\\
R(x,y) &\leftarrow k(x,y), gg^*g^-(g^-)^*(x,y)\\
Ans() &\leftarrow R^+(x,y)
\end{aligned}
$$

A possible expansion of this query is given by a chain x, w_1, w_2, w_3, y *from* x *to* y*:*

$f(x,w_1)$,
$k(w_1,w_2), g(w_1,z_1), g^-(z_1,z_2), g^-(z_2,w_2)$,
$w_2 = w_3$,
$k(w_3,y), g(w_3,t_1), g(t_1,t_2), g(t_2,t_3), g^-(t_3,y)$.

Here, the internal variables are $\{z_1, z_2, t_1, t_2, t_3\}$ *and the external variables are* $\{x, y, w_1, w_2, w_3\}$*.* ◻

Containment of nested UC2RPQs can be characterized in terms of containment of CQs. This is an easy consequence of the semantics of CQs [17, 39] and the fact that each nested UC2RPQ is equivalent to the union of its expansions.

▶ Proposition 20. Let Γ and Γ' be two nested UC2RPQs. 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 [17], due to the presence of inverses:

▶ **Definition 21.** 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 two nested UC2RPQs Γ and Γ' over Σ, we shall construct a 2RPQ $\tilde{E}$ and a nested UC2RPQ $\tilde{\Gamma}$ such that Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$. Our reduction is based on two ideas:

1. Expansions of Γ can be "serialized" and represented by *serialized expansions*, which are strings over a larger alphabet Δ. More importantly, serialized expansions constitute a regular language. Thus, we can construct a 2RPQ $\tilde{E}$ such that $L(\tilde{E})$ is precisely the set of serialized expansions of Γ. This technique has been already used before [14, 15].
2. Now we need to serialize Γ'. Proposition 20 basically tell us that Γ is contained in Γ' iff Γ' can be "mapped" to each expansion of Γ. We have replaced Γ by $\tilde{E}$. Thus, expansions of Γ are replaced by serialized expansions. By modifying the 2RPQs mentioned in Γ', we construct a nested UC2RPQ $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to a serialized expansion W of Γ iff Γ' can be mapped to the expansion of Γ represented by W. This is a novel technique and constitutes the crux of the reduction.

Serialization of Γ

Let M be the maximum number of atoms in the body of a rule in Γ. Let d be the depth of Γ. Recall that the depth of Γ is the maximum length of a directed path from some 2RPQ to the Ans predicate in its dependence graph, minus 1. For each $0 \leq i \leq d$, we define the set $\mathcal{V}^i = \{h_1^i, \ldots, h_{2M}^i\} \times \{1, 2, \exists\}$ and $\mathcal{S}^i = \{\$^i, \star^i, 1^i, 2^i\}$. 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 Δ as $\Delta = \Sigma^{\pm} \cup \mathcal{V} \cup \mathcal{S}$. 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 and $k \geq 0$, we define U^{+k} as the string over Δ (if well-defined) obtained from U by replacing $\$^j, \star^j, 1^j, 2^j$ by $\$^{j+k}, \star^{j+k}, 1^{j+k}, 2^{j+k}$, respectively, and (h_i^j, s) by (h_i^{j+k}, s), for $1 \leq i \leq 2M$ and $s \in \{1, 2, \exists\}$.

Intuitively, each (h_i^j, f) represent a variable: the index j indicates the level of nesting of the variable and f indicates whether is the first free variable ($f = 1$), the second free variable ($f = 2$) or an existential quantified variable ($f = \exists$). $\mathcal{S}$ contains auxiliary symbols and symbols in $\Sigma^{\pm}$ appears in the base case, when we expand 2RPQs.

Let φ be an expansion of an intensional predicate S of Γ of the form

$$\varphi(x_1, \ldots, x_n) \leftarrow \varphi_1(y_1, y_1'), \ldots, \varphi_m(y_m, y_m')$$

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

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

The depth of the expansion φ is the number of unfoldings that we need to reach the base case, when all atoms are 2RPQs. Formally, the *depth* of φ can be defined recursively as follows. Let $\mathcal{I} \subseteq \{1, \ldots, m\}$ be the set of indices i such that $R_i = Q^+$ for some predicate Q. If $\mathcal{I} = \emptyset$, then the depth of φ is 0. If $\mathcal{I} \neq \emptyset$, consider an index $i \in \mathcal{I}$ and assume that $R_i = Q^+$ and that $\varphi_i(y_i, y_i')$ is of the form $\varphi_i(y_i, y_i') \leftarrow \phi_1(w_0, w_1), \phi_2(w_1, w_2), \ldots, \phi_q(w_{q-1}, w_q)$, for $q \geq 1$, $w_0 = y_i$ and $w_q = y_i'$. For each $1 \leq j \leq q$, let $\zeta_j(t_1^j, t_2^j)$ be the expansion of Q defining $\phi_j(w_{j-1}, w_j)$. We denote by d_i the maximum depth of $\zeta_j(t_1^j, t_2^j)$ over all $j \in \{1, \ldots, q\}$. The depth of φ is defined as $1 + \max\{d_i \mid i \in \mathcal{I}\}$. Note that the depth of any expansion is at most the depth of Γ.

Now we define the notion of serialized expansion. Again, let φ be an expansion of predicate S of the form

$$\varphi(x_1, \ldots, x_n) \leftarrow \varphi_1(y_1, y_1'), \ldots, \varphi_m(y_m, y_m')$$

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

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

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 d_φ be the depth of φ. Let Φ be a function from $\{v_1, \ldots, v_N\}$ to $\mathcal{V}$ such that, for each $1 \leq i \leq N$, $\Phi(v_i) = (h_i^{d_\varphi}, 1)$ if v_i is the first free variable of φ; $\Phi(v_i) = (h_i^{d_\varphi}, 2)$ if v_i is the second free variable; or $\Phi(v_i) = (h_i^{d_\varphi}, \exists)$ if v_i is not a free variable in φ. Note that, when $S = Ans$, then the latter case holds for each v_i. Observe also that Φ is well-defined as $d_\varphi \leq d$ and $N \leq 2M$, where d is the depth of Γ and M is the maximum number of atoms in a rule body.

The *serialized expansion* W associated with φ is a string over Δ of the form

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

where the level of $\$$ is d_φ and, for each $1 \leq i \leq m$, W_i is defined as follows:

1. If $R_i = E$ is a 2RPQ, and $\varphi_i(y_i, y_i)$ is of the form $\varphi_i(y_i, y_i') \leftarrow r_1(y_i, z_1), r_2(z_1, z_2), \ldots, r_p(z_{p-1}, y_i')$, 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 $\varphi_i(y_i, y_i')$ is of the form

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

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

$$1 \cdot W_1' \cdot 2 \star \$ \star 1 \cdot W_2' \cdot 2 \star \$ \cdots \$ \star 1 \cdot W_q' \cdot 2$$

where the level of $1, 2, \star, \$$ is d_φ and $W_j' = (W_j'')^{+(d_\varphi - d_j - 1)}$, where W_j'' is the serialized expansion associated with $\zeta_j(t_1^j, t_2^j)$ and d_j is the depth of ζ_j, for each $1 \leq j \leq q$.

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. Note that the mapping from expansions to its associated serialized expansion is a bijection: each serialized expansion W represents a unique (modulo renaming) expansion φ of Γ.

▶ **Example 22.** *This is the serialized expansion associated with the expansion of Example 19:*

$$\begin{array}{l}
\$^1(h_1^1,\exists)1^1 \quad \$^0(h_1^0,1)f(h_2^0,2)\$^0 \quad 2^1\star^1 \\
\$^1\star^1 1^1 \quad \$^0(h_1^0,1)k(h_2^0,2)\$^0(h_1^0,1)gg^-g^-(h_2^0,2)\$^0 \quad 2^1\star^1 \\
\$^1\star^1 1^1 \quad \$^0(h_1^0,1)(h_2^0,2)\$^0 \quad 2^1\star^1 \\
\$^1\star^1 1^1 \quad \$^0(h_1^0,1)k(h_2^0,2)\$^0(h_1^0,1)gggg^-(h_2^0,2)\$^0 \quad 2^1(h_2^1,\exists)\$^1
\end{array}$$

□

Intuitively, a serialized expansion represents an expansion by reusing variables in $\mathcal{V}$. Intermediate variables that appears when we expand a predicate Q^+ are represented by the symbol $\star$. The level of symbols indicates the level of nesting. This is why we have to normalize the levels of W_j'' in the case (2) of the definition of serialized expansion.

Let W be a serialized expansion representing an expansion φ. It is clear from the definition, that each occurrence of a variable in $\mathcal{V}$ represents a variable in φ. We formalize this as a partial mapping var from $\{1, \ldots, |W|\}$ to $Var(\varphi)$, where $Var(\varphi)$ is the set of variables in φ. Thus, if an occurrence in W of a variable in $\mathcal{V}$ represents the variable y in φ, and this occurrence corresponds to the symbol at position $i \in \{1, \ldots, |W|\}$ in W, then we define $\mathsf{var}(i) = y$. Note that distinct positions in W can represent the same variable in φ. For instance, in Example 22, the occurrences of $(h_2^0, 2)$ and $\star^1$ in the first row, and the occurrence of $\star^1$ with the two occurrences $(h_1^0, 1)$ in the second row, represent the same variable w_1 in the expansion of Example 19.

At this point, only external variables are being represented by positions in W. To represent internal variables we do the following. Each time we expand a 2RPQ to a string $r_1 \cdots r_p$ in the case (1) of the definition of serialized expansion, then a variable z_j is represented by the occurrence of r_{j+1}, for each $j \in \{1, \ldots, p-1\}$. If this occurrence of r_{j+1} appears at position i in W, then we define $\mathsf{var}(i) = z_j$. Observe that each internal variable is represented by a unique position. For instance, in Example 22, the two occurrence of g^- represent z_1 and z_2, respectively (the first occurrence of g do not represent any variable).

Interestingly, serialized expansions of Γ constitute a regular language, and then, they can be represented by a 2RPQ $\tilde{E}$. We say that $\tilde{E}$ is the *serialization* of Γ.

▶ Proposition 23. Let Γ be a nested UC2RPQ over Σ. There exists a 2RPQ $\tilde{E}$ 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 the size of Γ.

Proof: We can construct $\tilde{E}$ bottom-up in the program Γ. We start with intensional predicates S such that S does not depend on any other predicate (formally, the in-degree is 0 in the dependence graph). We consider a rule ρ in $\mathsf{rules}(S)$. The rule is 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. We define a 1NFA $\mathcal{A}_\rho$ that accepts the serialized expansions of S that corresponds to rule ρ, ignoring the levels of the symbols, that is, any occurrence of a symbol could have any level. The construction of $\mathcal{A}_\rho$ is straightforward (see [14] for example). Moreover, the size of $\mathcal{A}_\rho$ is polynomial in ρ and Δ. We do this for all $\rho \in \mathsf{rules}(S)$. Let $\{\rho_1, \ldots, \rho_k\} = \mathsf{rules}(S)$. Then we define the union 1NFA $\mathcal{A}_S = \mathcal{A}_{\rho_1} \cup \cdots \cup \mathcal{A}_{\rho_k}$. Thus $\mathcal{A}_S$ accepts all the serialized expansions of S, ignoring levels. Note that the size of $\mathcal{A}_S$ is polynomial in Γ. Now we define a 1NFA $(\mathcal{A}_S)^+$ that accepts all the string of the form

$$1 \cdot W_1' \cdot 2 \star \$ \star 1 \cdot W_2' \cdot 2 \star \$ \star \cdots \star 1 \cdot W_q' \cdot 2$$

where $1, 2, \star, \$$ are of any level, and each W_i' is a serialized expansion of S modulo levels, that is, it is accepted by $\mathcal{A}_S$. It is straightforward to construct $(\mathcal{A}_S)^+$ and, moreover, its size is polynomial in Γ.

We continue in a bottom-up fashion in the dependence graph of Γ. Consider an intensional predicate R such that only depends on predicates S, whose automaton $(\mathcal{A}_S)^+$ is already constructed. Consider a rule $\rho \in \mathsf{rules}(R)$. Suppose ρ is of the form $R(x, y) \leftarrow R_1(y_1, y_1'), \ldots, R_m(y_m, y_m')$. It is clear that we can construct a 1NFA $\mathcal{A}_\rho$ that accepts all serialized expansions of R associated with rule ρ, ignoring levels. Again, the construction is similar of that in [14], but each time $R_i = S^+$ we use our automaton $(\mathcal{A}_S)^+$. The automaton $\mathcal{A}_\rho$ is of polynomial size in the size of Γ. As before, we define $\mathcal{A}_R = \mathcal{A}_{\rho_1} \cup \cdots \cup \mathcal{A}_{\rho_k}$, where $\{\rho_1, \ldots, \rho_k\} = \mathsf{rules}(R)$ and we define $(\mathcal{A}_R)^+$. Again, the automaton $(\mathcal{A}_R)^+$ is of polynomial size in the size of Γ.

We continue this process until we construct $\mathcal{A}_{Ans}$. It is easy to see, that we can construct a 1NFA $\mathcal{A}^\ell$ of polynomial size in Γ, that accepts serialized expansions such that the levels of the symbols are correct. The final 1NFA $\tilde{E}$ is the product automaton $\mathcal{A}_{Ans} \times \mathcal{A}^\ell$. Clearly, $\tilde{E}$ accepts all serialized expansion of Γ and the size of $\tilde{E}$ is polynomial in the size of Γ.

◻

Serialization of Γ'

We need to introduce some notation and concepts. Let $w = w_1 w_2 \cdots w_k$ and $u = u_1 u_2 \cdots u_\ell$ be strings over $\Delta^\pm = \{a^- \mid a \in \Delta\}$. A *folding* $\mathcal{F}$ from u into w is a sequence $\mathcal{F} = i_0, i_1, \ldots, i_\ell$ of positions in the set $\{0, \ldots, k\}$ such that, for each $1 \leq j \leq \ell$, 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 $u_j = w^-_{i_{j-1}}$. The notion of folding has been very useful to analyze 2RPQs [15].

The first and last position of the folding are denoted by $first(\mathcal{F})$ and $last(\mathcal{F})$, 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 $j_1 \in \{1, \ldots, k\}$, and similarly, the last symbol that is read is the j_2-symbol, with $j_2 \in \{1, \ldots, k\}$, 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 $first(\mathcal{F}) \in \{j_1 - 1, j_1\}$ and $last(\mathcal{F}) \in \{j_2 - 1, j_2\}$. For instance, consider the string $w = \$y_1b^-ay_2\$$. Then, $3, 2, 1, 2, 3, 4, 3$ is a $(3, 4)$-folding from $by_1^-y_1b^-aa^-$ into w, and $2, 3, 4, 3, 4, 5, 6$ is a $(3, 6)$-folding of $b^-aa^-ay_2\$$ into w.

▶ Observation 1. A convenient way to think of a folding is the following. Suppose we have strings $w = w_1 \cdots w_k$ and $u = u_1 \cdots u_\ell$ over $\Delta^\pm$. Let θ^w and θ^u be boolean CQs over Δ of the form

$$\theta^w() \leftarrow w_1(x_0, x_1), w_2(x_1, x_2), \ldots, w_k(x_{k-1}, x_k)$$
$$\theta^u() \leftarrow u_1(y_1, y_2), u_2(y_2, y_3), \ldots, u_\ell(y_{\ell-1}, y_\ell),$$

representing the strings w and u, respectively. Then, a folding $\mathcal{F}$ from u into w represents a containment mapping from θ^u to θ^w (and viceversa). This connection between foldings and containment mappings will be useful later.

Let φ be an expansion of the nested UC2RPQ Γ. Equality atoms in φ define equivalent classes over the variables of φ. We write $y \equiv_\varphi y'$ when two variables y and y' in φ belong to the same equivalent class. Recall that $\mathsf{neq}(\varphi)$ denotes the CQ without equality associated to φ. The query $\mathsf{neq}(\varphi)$ is obtained from φ by eliminating, iteratively, an equality atom $y_1 = y_2$ and renaming y_1 and y_2 by a fresh variable z. This defines a renaming function Φ_φ from the variables of φ to the variables of $\mathsf{neq}(\varphi)$. It is easy to see that $y \equiv_\varphi y'$ iff $\Phi_\varphi(y) = \Phi_\varphi(y')$, where y, y' are variables in φ.

We replaced Γ by $\tilde{E}$. Thus we replaced expansions of Γ by expansions of $\tilde{E}$, which are of the form $\theta^W() \leftarrow w_1(x_0, x_1), w_2(x_1, x_2), \ldots, w_n(x_{n-1}, x_n)$, for a serialized expansion $W = w_1 \cdots w_n$. Suppose W represents an expansion φ of Γ. Our goal is to construct $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to θ^W iff Γ' can be mapped to $\mathsf{neq}(\varphi)$. To construct $\tilde{\Gamma}$, we modify Γ' in order to translate mappings from Γ' to $\mathsf{neq}(\varphi)$, into mappings from $\tilde{\Gamma}$ to θ^W (and viceversa). The main difficulty is the following: It could be possible that $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$ for two distinct positions $i < j$ in W. This implies that $\Phi_\varphi(\mathsf{var}(i)) = \Phi_\varphi(\mathsf{var}(j))$, and thus positions i and j corresponds to the same variable in $\mathsf{neq}(\varphi)$. Hence, in order to simulate mappings from Γ' to $\mathsf{neq}(\varphi)$, we have to consider positions i and j as equivalent, that is, we must be able to "jump" between positions i and j, whenever necessary.

To overcome this problem, we introduce the notion of *equality string*. Equality strings are string over $\Delta^\pm$ with the following key property: For positions $1 \le i < j \le |W|$, $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$ iff there is a (i, j)-folding of some equality string into W. There are four types of basic equality strings:

1. An *horizontal equality string* α is a string over $\Delta^\pm$ for which there is a sequence of variables $\vartheta_1, \ldots, \vartheta_\ell \in \mathcal{V}$, for $\ell \ge 1$, of the same level j such that α satisfies the regular expression

$$\begin{aligned} & V_1G_j^*V_1V_2G_j^* \cdots V_{\ell-1}V_\ell G_j^*V_\ell \\ + \;& V_1V_2G_j^*V_2V_3G_j^* \cdots V_{\ell-1}V_\ell G_j^*V_\ell \\ + \;& V_1G_j^*V_1V_2G_j^* \cdots V_{\ell-2}V_{\ell-1}G_j^*V_{\ell-1}V_\ell \\ + \;& V_1V_2G_j^*V_2V_3G_j^* \cdots V_{\ell-2}V_{\ell-1}G_j^*V_{\ell-1}V_\ell \end{aligned}$$

where V_i is the regular expression $\vartheta_i + \vartheta_i^-$, for each $1 \le i \le \ell$, and G_j is the alphabet $\Delta^\pm_{\le j}$, where $\Delta_{\le j} = \Sigma^\pm \cup \bigcup_{k=0}^j \mathcal{V}^k \cup \bigcup_{k=0}^j \mathcal{S}^k$. Note that, when $\ell = 1$, then the expression becomes simply $V_1 G_j^* V_1 + V_1$. Horizontal equality strings detect equalities between variables in $\mathcal{V}$, in the higher level j of a "sub" serialized expansion W'. This equalities can be produced either by the repetition of the same variable in different atoms, or by equality atoms. We have four terms in the regular expression, one for each pattern we could see. The use of G_j^* forces the equality string to be folded inside W': if we go outside W' we must see a symbol of level $j+1$ (specifically, we must see 1^{j+1} or 2^{j+1}).

2. A *downward equality string* α is a string over $\Delta^{\pm}$ for which there exists variables $\vartheta \in \mathcal{V} \cup \{\star^i \mid 0 \leq i \leq d\}$ and $\xi \in \mathcal{V}$ of level j and $j-1$, respectively, such that, either
   - ξ is of the form $(h_i^{j-1}, 1)$, and α satisfies the regular expression $\vartheta \cdot 1^j \cdot G_{j-1}^* \cdot \xi$, or
   - ξ is of the form $(h_i^{j-1}, 2)$ and α satisfies the regular expression $\vartheta^- \cdot (2^j)^- \cdot G_{j-1}^* \cdot \xi$.

   Each time we decrease a level in a serialized expansion, that is, each time we see a substring of the form $\$y1W'2y'\$$, where $y, y' \in \mathcal{V} \cup \{\star^i \mid 0 \leq i \leq d\}$ and W' is a sub serialized expansion, then y and y' corresponds to the first and second free variables in W', respectively. These connections are detected using downward equality strings.
3. An *upward equality string* α is a string over $\Delta^{\pm}$ for which there exists variables $\vartheta \in \mathcal{V} \cup \{\star^i \mid 0 \leq i \leq d\}$ and $\xi \in \mathcal{V}$ of level j and $j-1$, respectively, such that, either
   - ξ is of the form $(h_i^{j-1}, 1)$, and α satisfies the regular expression $\xi \cdot G_{j-1}^* \cdot (1^j)^- \cdot \vartheta^-$, or
   - ξ is of the form $(h_i^{j-1}, 2)$ and α satisfies the regular expression $\xi \cdot G_{j-1}^* \cdot 2^j \cdot \vartheta$.

   The intuition is analogous to that of downward equality strings, but now we increase the level of variables.
4. An *star equality string* α is a string over $\Delta^{\pm}$, either of the form $\star^j$, $(\star^j)^-$, $\star^j\$^j\star^j$ or $(\star^j)^-(\$^j)^-(\star^j)^-$, for some $j \in \{0, \ldots, d\}$. These strings connect equivalent occurrences of the $\star$ symbol.

An *equality string* is a string α of the form $\alpha = \alpha_1 \cdot \alpha_2 \cdots \alpha_k$, where α_i is either an horizontal, downward, upward or star equality string, for each $1 \leq i \leq k$.

▶ **Example 24.** *Consider the expansion in Example 19 and its serialized expansion in Example 22. Consider the first occurrence of $(h_1^0, 2)$ in the second row, and the last occurrence of $(h_1^0, 1)$ in the last row. These two occurrence represents the variables w_2 and w_3 in the expansion, respectively, which are equivalent. This is witnessed, for example, by the following equality string:*

$$\begin{array}{rl}
(\textit{Upward}) & (h_2^0, 2)\$^0(h_1^0, 1)gg^-g^-(h_2^0, 2)\$^0 2^1 \star^1 \cdot \\
(\textit{Star} \times 3) & (\star^1)^- \cdot \star^1\$^1 \star^1 \cdot (\star^1)^- \cdot \\
(\textit{Downward}) & \star^1\, 1^1\$^0(h_1^0, 1) \cdot \\
(\textit{Horizontal}) & (h_1^0, 1)^-(h_1^0, 1)(h_2^0, 2)(h_2^0, 2)^- \cdot \\
(\textit{Upward}) & (h_2^0, 2)\$^0 2^1 \star^1 \cdot \\
(\textit{Star} \times 3) & (\star^1)^- \cdot \star^1\$^1 \star^1 \cdot (\star^1)^- \cdot \\
(\textit{Downward}) & \star^1\, 1^1\$^0(h_1^0, 1)k(h_2^0, 2)\$^0(h_1^0, 1)
\end{array}$$

◻

▶ **Lemma 25.** *Let W be a serialized expansion representing an expansion φ. Let i, j be two positions in $\{1, \ldots, |W|\}$. Then, the following are equivalent:*

1. $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$.
2. *there is an equality string α and a (i, j)-folding from α into W.*

Moreover, if (2) holds, then for any choice $(k, k') \in \{i-1, i\} \times \{j-1, j\}$*, we can choose* α *and the* (i, j)*-folding* $\mathcal{F}$ *such that* $first(\mathcal{F}) = k$ *and* $last(\mathcal{F}) = k'$*.*

Proof: Suppose $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$. It is easy to see that we can find a sequence of positions $k_0, \dots, k_\ell$ in $\{1, \dots, |W|\}$, with $k_0 = i$ and $k_\ell = j$ such that, for each $0 \leq p \leq \ell - 1$, $\mathsf{var}(k_p) \equiv_\varphi \mathsf{var}(k_{p+1})$, and either

1. if $\mathsf{var}(k_p) = y$ and $\mathsf{var}(k_{p+1}) = y'$, then $y, y' \in \{y_1, y_1', \dots, y_m, y_m'\}$ for a sub expansion $\theta = \theta_1(y_1, y_1'), \dots, \theta_m(y_m, y_m')$ of φ and y and y' are equivalent due to repetitions of variables or equality atoms between variables in $\{y_1, y_1', \dots, y_m, y_m'\}$. In particular the symbols at positions k_p and k_{p+1} have the same level.
2. There is a substring in W of the form $\$h1W'2h'\$$, and either h is at position k_p and some occurrence representing the first free variable in W' is in position k_{p+1}, or h' is at position k_p and some occurrence representing the second free variable in W' is in position k_{p+1},
3. There is a substring in W of the form $\$h1W'2h'\$$, and either some occurrence representing the first free variable in W' is in position k_p and h is at position k_{p+1}, or some occurrence representing the second free variable in W' is in position k_p and h' is at position k_{p+1},
4. There is a substring $\star^j \$^j \star^j$ in W, for some $0 \leq j \leq d$, and k_p and k_{p+1} are the positions of the first $\star^j$ and the second $\star^j$, or the positions of the second $\star^j$ and the first $\star^j$.

For each case, it is easy to find an equality string α_p and a (k_p, k_{p+1})-folding of α_p to W. For example, for case (1), as y and y' are equivalent, there must exist a sequence of variables $x_1, \dots, x_t \in \{y_1, y_1', \dots, y_m, y_m'\}$, with $x_1 = y$ and $x_t = y'$, where $x_i = x_{i+1}$ is an equality atom in θ, for each $1 \leq i \leq t-1$. This translates directly in an horizontal equality string α_p, and a (k_p, k_{p+1})-folding from α_p to W. All the other cases translate to a downward, upward and star equality string, respectively.

We can define $\alpha = \alpha_0 \cdots \alpha_{\ell-1}$. By definition α is an equality string. Observe also that there is a (i, j)-folding from α to W. Indeed, the only problem is that for some $0 \leq p \leq \ell - 1$, the folding $\mathcal{F}_p$ associated with α_p and the folding $\mathcal{F}_{p+1}$ associated with α_{p+1} satisfies $last(\mathcal{F}_p) \neq first(\mathcal{F}_{p+1})$. This can be easily fixed by adding a suitable equality string between α_p and α_{p+1} of the form ϑ or ϑ^- for $\vartheta \in \mathcal{V} \cup \{\star^j : 0 \leq j \leq d\}$.

Now assume that there is an equality string $\alpha = \alpha_0 \cdots \alpha_{\ell-1}$ and a (i, j)-folding from α to W. In particular, there exists a sequence of positions $k_0, \dots, k_\ell$ with $k_0 = i$ and $k_\ell = j$ such that there is a (k_p, k_{p+1})-folding from α_p to W, for each $0 \leq p \leq \ell - 1$. By the definition of the basic equality strings it is clear that $\mathsf{var}(k_p) \equiv_\varphi \mathsf{var}(k_{p+1})$, for each $0 \leq p \leq \ell - 1$. This implies that $\mathsf{var}(i) \equiv_\varphi \mathsf{var}(j)$.

Finally, note that we can choose where this (i, j)-folding starts and ends, since we can add suitable equality strings at the beginning or at the end of α, of the form ϑ or ϑ^- for $\vartheta \in \mathcal{V} \cup \{\star^j : 0 \leq j \leq d\}$.

□

Now we are ready to serialize the nested UC2RPQ Γ'. Let $w = w_1 \cdots w_p$ be a string over $\Sigma^\pm$. The *serialization* of w, denoted by $\mathsf{serial}(w)$, is the set of 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 $0 \leq i \leq p$, the string α_i is either ε or an equality string. If L is a language over $\Sigma^\pm$, then $\mathsf{serial}(L)$ is the language over $\Delta^\pm$ defined by $\mathsf{serial}(L) = \{w' \mid w' \in \mathsf{serial}(w),\ \text{for some } w \in L\}$. As it turns out, if L is regular, then $\mathsf{serial}(L)$ is also regular.

▶ **Lemma 26.** *For each 1NFA $\mathcal{A}$ over $\Sigma^{\pm}$, there is an 1NFA $\mathcal{A}'$ 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: For each of the four types of basic equality string, we can easily construct a 1NFA $\mathcal{A}_i$ ($i \in \{1, 2, 3, 4\}$) such that $\mathcal{A}_i$ accepts precisely the basic equality strings of type i. It is not hard to see that we only need to remember a finite number of variables in $\mathcal{V}$, thus the size of $\mathcal{A}_i$ is polynomial in Δ. Let $\mathcal{A}_{\cup}$ be the union automaton $\mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$. Let $\mathcal{A}_{eq}$ be the 1NFA that accept string of the form $\alpha_1 \cdots \alpha_k$, where $k \geq 1$ and each $\alpha_i \in L(\mathcal{A}_{\cup})$. The automaton $\mathcal{A}_{eq}$ accepts precisely the set of equality strings and its size is polynomial in Δ. The automaton $\mathcal{A}'$ can be now easily defined. The intuition is that $\mathcal{A}'$ on an input $S = s_1 \cdots s_n$ guesses the positions $i_1 < \cdots < i_\ell$ that corresponds to the string $s_{i_1} s_{i_2} \cdots s_{i_\ell}$ that belongs to $L(\mathcal{A})$ and it checks that each intermediate substring $s_1 \cdots s_{i_1-1}, s_{i_1+1} \cdots s_{i_2-1}, \ldots, s_{i_\ell+1} \cdots s_n$ is an equality string, that is, it is accepted by $\mathcal{A}_{eq}$. Note that 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.

◻

Consider our initial nested UC2RPQs Γ and Γ'. The *serialization* $\tilde{\Gamma}$ of Γ' is the nested UC2RPQ over Δ obtained from Γ' by replacing each 2RPQ E in Γ' by $\mathsf{serial}(E)$. It is important to note that the serialization $\tilde{\Gamma}$ of Γ' depends on both Γ and Γ'. This is because it depends on Δ (which, at the same time, depends on Γ). Observe also that the size of $\tilde{\Gamma}$ is polynomial in the size of Δ and Γ', and thus, polynomial in the size of Γ and Γ'.

▶ Proposition 27. Let Γ and Γ' be two nested UC2RPQs over Σ. Let $\tilde{E}$ and $\tilde{\Gamma}$ be the serialization of Γ and Γ', respectively. Then, Γ is contained in Γ' if and only if $\tilde{E}$ is contained in $\tilde{\Gamma}$.

Proof: Consider an expansion φ of a nested UC2RPQ. In the construction of φ, when we reach the base case (1) in the definition of expansion, 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 $\mathsf{BE}_\varphi = \{1, \ldots, e(\varphi)\}$, where $e(\varphi)$ is number of times we have to expand a 2RPQ in the whole construction of φ. Associated with the expansion φ, we have two functions A_φ and B_φ. The function A_φ maps a basic expansion $i \in \mathsf{BE}_\varphi$ to the associated 2RPQ $A_\varphi(i)$ that we are expanding. The function B_φ maps $i \in \mathsf{BE}_\varphi$ to the string $B_\varphi(i) \in L(A_\varphi(i))$ that we choose in the expansion. Note that the internal variables are those that appear exactly when we apply a basic expansion.

Recall that $\mathsf{neq}(\varphi)$ denotes the CQ without equality associated to φ. The query $\mathsf{neq}(\varphi)$ is obtained from φ by eliminating, iteratively, an equality atom $y = y'$ and renaming y and y' by a fresh variable z. This defines a renaming function Φ_φ from the variables of φ to the variables of $\mathsf{neq}(\varphi)$. Let V^{ext} and V^{int} be the external and internal variables of φ, respectively. Then, we define the *external* and *internal* variables of $\mathsf{neq}(\varphi)$ as $\Phi_\varphi(V^{ext})$ and $\Phi_\varphi(V^{int})$, respectively. Note that internal variables of φ and $\mathsf{neq}(\varphi)$ coincide, since Φ_φ is the identity over V^{int}. Now we are ready to prove the proposition.
($\Rightarrow$) Suppose that Γ is contained in Γ'. Let ϱ be an expansion of $\tilde{E}$ of the form

$$\varrho() \leftarrow w_1(z_1, z_2), \ldots, w_p(z_p, z_{p+1})$$

where $p \geq 0$ and $W = w_1 \cdots w_p \in L(\tilde{E})$. We shall prove that there is an expansion ϱ' of $\tilde{\Gamma}$ and a containment mapping ν from $\mathsf{neq}(\varrho')$ to $\mathsf{neq}(\varrho)$. By Proposition 20, this implies that $\tilde{E}$ is contained in $\tilde{\Gamma}$, as required.

Let φ be the expansion of Γ associated with W. Since Γ is contained in Γ', there is an expansion φ' of Γ' and a containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$. To construct the expansion ϱ' of $\tilde{\Gamma}$ we start with the expansion φ'. Thus, we have $\mathsf{BE}_{\varrho'} = \mathsf{BE}_{\varphi'}$. Note that, by definition, $A_{\varrho'}(i) = \mathsf{serial}(A_{\varphi'}(i))$, for each $i \in \mathsf{BE}_{\varrho'}$. The important step in the construction of ϱ' is to choose appropriately the function $B_{\varrho'}$. We construct $B_{\varrho'}$ as follows. If $B_{\varphi'}(i) = \varepsilon$, for some $i \in \mathsf{BE}_{\varrho'}$, then $B_{\varrho'}(i) = \varepsilon$. Note that $B_{\varrho'}(i) \in A_{\varrho'}(i)$, since $\varepsilon \in A_{\varphi'}(i)$ implies $\varepsilon \in \mathsf{serial}(A_{\varphi'}(i)) = A_{\varrho'}(i)$. If $B_{\varphi'}(i) \neq \varepsilon$, for some $i \in \mathsf{BE}_{\varrho'}$, then $B_{\varrho'}(i) \neq \varepsilon$. We shall define $B_{\varrho'}(i)$ later, together with the containment mapping ν. Note that the external variables in φ' and ϱ' coincide. We denote by V^{ext} this set of external variables. Moreover, if $\Phi_{\varphi'}$ and $\Phi_{\varrho'}$ are the renaming from φ' to $\mathsf{neq}(\varphi')$, and from ϱ' to $\mathsf{neq}(\varrho')$, respectively, then, by construction, we can choose $\Phi_{\varphi'}$ and $\Phi_{\varrho'}$ to coincide over V^{ext}. Thus, the external variables of $\mathsf{neq}(\varphi')$ and $\mathsf{neq}(\varrho')$ also coincide. We denote by U^{ext} this set of external variables.

Now we define the function $B_{\varrho'}$ and the containment mapping ν from $\mathsf{neq}(\varrho')$ to $\mathsf{neq}(\varrho)$. We start by defining ν over U^{ext}, that is, the external variables of $\mathsf{neq}(\varrho')$. Let μ be the containment mapping from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$, and Φ_φ the renaming from φ to $\mathsf{neq}(\varphi)$. Let t be a variable in U^{ext}. Suppose $\mu(t)$ is an external variable y in $\mathsf{neq}(\varphi)$. Then, we choose $j \in \{1, \dots, p\}$ such that $\Phi_\varphi(\mathsf{var}(j)) = y$. In other words, j is a position in W representing y. Then, we define $\nu(t) = z_j$. Suppose now that $\mu(t)$ is an internal variable z in $\mathsf{neq}(\varphi)$. Then, we pick $j \in \{1, \dots, p\}$ such that $\Phi_\varphi(\mathsf{var}(j)) = z = \mathsf{var}(j)$. Note that there is a unique such j. We define $\nu(t) = z_j$. At this point, ν is well-defined over U^{ext}.

Now we define simultaneously $B_{\varrho'}(i)$ and the extension of ν to $B_{\varrho'}(i)$, for each $i \in \mathsf{BE}_{\varrho'}$. Let $i \in \mathsf{BE}_{\varrho'}$. Suppose i is a basic expansion between external variables t and t'. Note that $\nu(t), \nu(t')$ are already defined. Observe also that, by Observation 1, the extension of ν to $B_{\varrho'}(i)$ is simply a folding $\mathcal{F}$ of $B_{\varrho'}(i)$ into W such that $first(\mathcal{F}) = \nu(t)$ and $last(\mathcal{F}) = \nu(t')$. Let $B_{\varphi'}(i) = s_1 \cdots s_n$ and let $o_2 \dots o_n$ be the internal variables associated with the basic expansion i in φ', and let $o_1 = t$ and $o_{n+1} = t'$. We examine the values $\mu(o_1), \mu(o_2), \dots, \mu(o_n), \mu(o_{n+1})$. Let $j_1 < j_2 < \cdots < j_m$ be all the positions j in $\{1, \dots, n+1\}$, such that $\mu(o_j)$ is an external variable in $\mathsf{neq}(\varphi)$. We define $B_{\varrho'}(i)$ as the string of the form

$$s_1 \cdots s_{j_1-1} \alpha_1 s_{j_1} \cdots s_{j_2-1} \alpha_2 \cdots s_{j_{m-1}} \cdots s_{j_m-1} \alpha_m s_{j_m} \cdots s_n$$

where the α_k's are equality strings defined next, together with the folding $\mathcal{F}$ from $B_{\varrho'}(i)$ to W. The folding $\mathcal{F}$ starts at $\nu(o_1) = \nu(t)$. We iterate from $q = 1$ to $q = n+1$. While $q \leq j_1 - 1$, we can extend $\mathcal{F}$ simulating the mapping μ. When $q = j_1 - 1$, since μ is a containment mapping, there exists two positions p_1, p_2 such that $\mathsf{var}(p_1) \equiv_\varphi \mathsf{var}(p_2)$, $\Phi_\varphi(\mathsf{var}(p_1)) = \Phi_\varphi(\mathsf{var}(p_2)) = \mu(o_{j_1})$. By Lemma 25 we can choose a suitable equality string α_1 and extend our folding $\mathcal{F}$ appropriately. We continue extending our folding $\mathcal{F}$ and defining our equality strings until $q = n+1$. Observe that at the end, $last(\mathcal{F}) = \nu(o_{n+1}) = \nu(t')$, and $B_{\varrho'}(i) \in L(\mathsf{serial}(A_{\varphi'}(i)))$, as required.

($\Leftarrow$) Suppose $\tilde{E}$ is contained in $\tilde{\Gamma}$. Let φ be an expansion of Γ. We want to show that there is an expansion φ' of Γ' and a containment mapping μ from $\mathsf{neq}(\varphi)$ to $\mathsf{neq}(\varphi')$. Let $W = w_1 \cdots w_p$ be the serialized expansion associated with φ and $\varrho() \leftarrow w_1(z_1, z_2), \dots, w_p(z_p, z_{p+1})$ be the expansion of $\tilde{E}$ associated with W. There is an expansion ϱ' of $\tilde{\Gamma}$ and a containment mapping from $\mathsf{neq}(\varrho')$ to $\mathsf{neq}(\varrho)$. Without loss of generality we can choose ϱ' to satisfy the following property: Let $i \in \mathsf{BE}_{\varrho'}$. If $B_{\varrho'}(i) \neq \varepsilon$, then we can choose the corresponding string $u \in A_{\varphi'}(i)$ with $B_{\varrho'}(i) \in \mathsf{serial}(w)$ to be also not empty. This avoid $B_{\varrho'}(i)$ to be an equality string. We impose $B_{\varrho'}(i)$ to be ε instead, in this case. It is easy to see that by choosing suitable equality string in the other basic expansions, we can modify ϱ' such that there still is a containment mapping from $\mathsf{neq}(\varrho')$ to $\mathsf{neq}(\varrho)$. We define φ' to be exactly like ϱ' but for each $i \in \mathsf{BE}_{\varphi'} = \mathsf{BE}_{\varrho'}$, we choose $B_{\varphi'}(i)$ to be the associated string with $B_{\varrho'}(i)$, that is, a string such that $B_{\varrho'}(i) \in \mathsf{serial}(B_{\varphi'}(i))$. By the previous property, $B_{\varrho'}(i) = \varepsilon$ iff $B_{\varphi'}(i) = \varepsilon$. This implies that the external variables of $\mathsf{neq}(\varphi')$ and $\mathsf{neq}(\varrho')$ coincide.

It remains to define the containment mapping μ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$. Let ν be the containment mapping from $\mathsf{neq}(\varrho')$ to $\mathsf{neq}(\varrho)$. Let t be an external variable of $\mathsf{neq}(\varphi')$. We define $\mu(t)$ to be the variable $\Phi_\varphi(\mathsf{var}(\nu(t)))$. Now we need to define μ over the internal variables of $\mathsf{neq}(\varphi')$. If $B_{\varphi'}(i) = s_1 \cdots s_n$, this amounts to define a containment mapping τ from the CQ $s_1(t, o_2), \cdots s_n(o_n, t')$ to $\mathsf{neq}(\varphi)$ such that $\tau(t) = \mu(t)$ and $\tau(t') = \mu(t')$. We know that there is a folding $\mathcal{F}$ from $B_{\varrho'}(i)$ to W such that $first(\mathcal{F}) = \nu(t)$ and $last(\mathcal{F}) = \nu(t')$. By following this folding and ignoring the equality strings in $B_{\varrho'}(i)$, we can easily define τ, for each $i \in \mathsf{BE}_{\varphi'}$. This implies that Γ is contained in Γ'.

$\square$

Since the construction of $\tilde{E}$ and $\tilde{\Gamma}$ can be carried out in polynomial time from Γ and Γ', we have shown the following theorem.

▶ **Theorem 28.** *There is a polynomial time reduction from the containment problem of nested UC2RPQs to the containment problem of a 2RPQ in a nested UC2RPQ.*

A.3 Containment of 2RPQs in nested UC2RPQs: Upper Bound

Cuts. Let γ be a nested UC2RPQ of nesting depth 0, defined by the rules

$$\begin{array}{rcl} \gamma(x_1, \ldots, x_n) & \leftarrow & \gamma_1^1(y_1^1, {y^1}'_1), \ldots, \gamma_{m_1}^1(y_{m_1}^1, {y^1}'_{m_1}), \\ \vdots & \vdots & \\ \gamma(x_1, \ldots, x_n) & \leftarrow & \gamma_1^\ell(y_1^\ell, {y^\ell}'_1), \ldots, \gamma_{m_\ell}^\ell(y_{m_\ell}^\ell, {y_{m_\ell}^\ell}'), \end{array} \tag{2}$$

and assume that each 2RPQ γ_j^i is given by an automaton $A_j^i = (Q_j^i, \Sigma, {q_0}_j^i, F_j^i, \delta_j^i)$. For clarity, hereon we refer to the answer predicate of nested UC2RPQs using the same name as the query, in this case γ, instead of the usual Ans. We always denote nested UC2RPQs by the name of their answer predicate.

A *cut* of γ is an ℓ-tuple $(C^1, \ldots, C^\ell)$, where each of the C^i's is either $\bot$ or a triple of form $(\text{Prev}^i, \text{Same}^i, \mathcal{S}^i)$, where for each $1 \le i \le \ell$, Same^i and Prev^i are subsets of the set of variables appearing in the i-th rule of query (2) above such that $\text{Same}^i \subseteq \text{Prev}^i$, and $\mathcal{S}^i$ is an m_i-tuple $\mathcal{S}^i = (s_1^i, \ldots, s_{m_i}^i)$ of states from $Q_1^i \times \cdots \times Q_{m_i}^i$, i,e, containing one state of each of the A_j^is, $1 \le j \le m_i$.

Before continuing, let us now give some intuition behind the notion of cuts for the simplest case of C2RPQs. Consider for a moment a C2RPQ Q, and expansion q of Q and a linearization $q' = r_1(z_1, z_2), \ldots, r_p(z_p, z_{p+1})$ of q. If we concentrate on the string $s = r_1 \cdots r_p$, the flattening can be seen as a mapping from the variables of q to different positions of s: variable y in q is assigned position i in s iff the image of y in q' is z_i. Note that in this way, variables in Q are therefore given a particular position in s as well. Furthermore, it is easy to see that the following holds: for each 2RPQ $\phi(y, y')$ in Q, if y and y' are mapped to positions n and m, respectively, of s, then there is a string u and a folding of u into s that starts in position n and ends in m, or in other words, if the automaton corresponding to ϕ must accept s, when starting in position n and ending in position m.

Given a particular position n in s, a cut $(\text{Prev}, \text{Same}, \mathcal{S})$ of a C2RPQ Q contains all the information that we need to show that q' is a linearization of Q: which variables of Q have been mapped to previous positions in s (this is the set Prev), which ones are mapped to the position n (the set Same), and, for those cases in which only one variable of an 2RPQ ϕ in Q has been mapped to a position $n' \leq n$ in λ, the set $\mathcal{S}$ contains a state of the automaton for ϕ, representing a run of this automaton that starts in position n' of s and ends in n. Once this automaton reaches a final state, say in a position n'', the remaining variable of ϕ can be included in the set Prev.

▶ **Example 29.** Consider query $Q \leftarrow g^+(x,z), g^+(z,y)$ over alphabet $\{g,f\}$, stating that there is a path labeled with g^+ between both x and y, and y and z. It is not difficult to see that the CQ $q = g(x,x'), g(x',z), g(z,y'), g(y',y)$ is a linearization of Q. The string associated to q is $s = ggg^-g^-$. The cut for Q in position 3 of s that represents the expansion q contains variables x and z in Prev, and variable z in Same, since z is mapped to position 3 of s. The cut on position 4 of s contains again x and z in Prev, no variable in position Same. Furthermore, since y is to be mapped to position 5, it is also important to store a state of the automaton representing the 2RPQ g^+, to check that it accepts the string ggg^-g^- when starting in position 3 and ending in position 5.

Since we are dealing with unions of C2RPQs we also need to account for the case when a certain disjunct of γ cannot be mapped to a certain linearization: in this case, the corresponding triple of the cut is set to $\perp$. Using the definition of cuts for UC2RPQs, we can now define cuts for queries of nesting depth > 0:

▶ **Definition 30.** **(Cuts of nested UC2RPQs)** Let γ be a nested UC2RPQ, and assume that the rules in γ with the answer predicate in the head follow form (2) above (note that now some of the γ_j^is might be predicates P^+ instead of RPQs).

Then a cut of γ is an ℓ-tuple $(C^1,\dots,C^\ell)$, where each C^i is either $\perp$, or a triple $(\text{Prev}^i, \text{Same}^i, \mathcal{S}^i)$, where for each $1 \leq i \leq \ell$, Same^i and Prev^i are subsets of the set of variables appearing in the i-th rule of query (2) above such that $\text{Same}^i \subseteq \text{Prev}^i$, and $\mathcal{S}^i$ is an m_i-tuple $\mathcal{S}^i = (s_1^i,\dots,s_{m_i}^i)$ defined as follows:

- If γ_j^i is an RPQ given by automaton A, then s_j^i is a state of A.
- Otherwise γ_j is the the transitive closure P^+ of a nested UC2RPQ whose answer predicate is P and with nesting depth strictly lower than γ. Then s_j^i is a cut of P.

Furthermore, *initial cuts* are those in which each Prev^i is empty, and *final cuts* are those in which at least one of the O^is is empty. A cut *marks* a variable x if in all C^is that are not $\perp$ x belongs to Same^i.

The same intuition in the previous example can be now extended to nested C2RPQ, and to nested UC2RPQs, by taking the cuts of the subqueries under transitive closure (instead of the state of the automata for the 2RPQs), and by considering tuples of cuts for each union of queries.

▶ **Example 31.** Consider again query

$$\begin{array}{rcl} G_1(a,b) & \leftarrow & g(a,b) \\ G_2(c,d) & \leftarrow & g(c,d) \\ R(p,q) & \leftarrow & f(p,q) \\ R(u,v) & \leftarrow & k(u,v), G_1^+(u,z), G_2^+(v,z) \\ Ans(x,y) & \leftarrow & R^+(x,y) \end{array}$$

The cuts from Q are of form $(\text{Prev}, \text{Same}, \mathcal{S})$, where Prev and Same are subsets of $\{x,y\}$, and $\mathcal{S}$ is itself a cut of R. In turn, those are of form (C_1, C_2), where cuts C_1 are of form $(\text{Prev}^1, \text{Same}^1, \mathcal{S}^1)$, where Prev^1 and Same^1 are subsets of $\{p, q\}$ and $\mathcal{S}^1$ contains a state of the NFA that checks for foldings of the 2RPQ $f(p,q)$; and cuts C_2 are of form $(\text{Prev}^2, \text{Same}^2, \mathcal{S}^2)$, where Prev^2 and Same^2 are subsets of $\{u, v, z\}$ and $\mathcal{S}^1$ is the tuple (s^2_1, s^2_2, s^2_3), with s^2_1 a state of the NFA that checks for foldings of the 2RPQ $f(p,q)$, s^2_2 a cut of G_1 and s^2_3 a cut of G_2

Let $Cuts(\gamma)$ be the set of all cuts of a query γ of form (2). We first note the following size bound:

▶ **Lemma 32.** *For every $k \geq 0$ and γ a nested UC2RPQ, the size of $Cuts(\gamma)$ belongs to $O(2^{|\gamma|})$. Furthermore, the size of each cut in $Cuts(\gamma)$ is polynomial in the size of γ.*

Proof: Let us first count the number of cuts of a nested UC2RPQs γ of depth 0. As usual, assume that γ is of form

$$\begin{array}{rcl} \gamma(x_1,\dots,x_n) & \leftarrow & \gamma^1_1(y^1_1, {y^1}'_1), \dots, \gamma^1_{m_1}(y^1_{m_1}, {y^1}'_{m_1}), \\ \vdots & & \vdots \\ \gamma(x_1,\dots,x_n) & \leftarrow & \gamma^\ell_1(y^\ell_1, {y^\ell}'_1), \dots, \gamma^\ell_{m_\ell}(y^\ell_{m_\ell}, {y^\ell_{m_\ell}}'), \end{array} \tag{3}$$

where for now each γ^i_j is a 2RPQ. Let $Vars(\gamma)$ be the set of variables used in γ

For each $1 \leq i \leq \ell$, the number of different sets $\text{Prev}^i, \text{Same}^i \subseteq Vars(\gamma)$ is obviously of order $2^{|Vars(\gamma)|}$. Furthermore, for each different choice of $\text{Prev}^i, \text{Same}^i$, one could have potentially any combination of states from each of the automata defining γ^i_j. Thus, the number of cuts is of order $2^{|Vars(\gamma)|} \cdot |\gamma^i_1| \cdot \cdots \cdot \gamma^i_{m_i}$. Combining for all $1 \leq i \leq m$, we easily obtain the $O(2^{|vars(\gamma)| \cdot 2^{|}\gamma|})$, or $O(2^{|}\gamma|)$ upper bound. Moreover, Prev^i and Same^i are clearly linear in γ, and so is each state of γ^i_j. Thus the size of each cut is of order $|\gamma| + m_i \cdot |\gamma|$, which is clearly polynomial.

If γ is a nested UC2RPQ of depth k, then assume that all the rules mentioning γ are of form (3). Assume also for simplicity that all such γ^i_j is itself a nested UC2RPQ of lower depth. Using the same argument as above, we obtain that the number of different choices for Prev^i and Same^i is of order $2^{|Vars(\gamma|}$. the difference now is the choice of the number of cuts. If $|Cuts(\gamma^i_j)|$ denotes the number of cuts of γ^i_j, we have that the number of cuts of γ corresponds roughly to

$$\prod_{1 \leq i \leq \ell} \left(2^{|Vars(\gamma|} \cdot |Cuts(\gamma^i_1)| \cdot \cdots \cdot |Cuts(\gamma^i_{m_i})|\right).$$

Now if we apply an inductive argument, and assume that $|Cuts(\gamma^i_1)|$ is itself of order $2^{|\gamma|^i_j|}$, we have that the above equation is of order

$$2^{\ell \cdot |Vars(\gamma|} \cdot 2^{\ell \cdot (|\gamma|^i_1 + \cdots + |\gamma|^i_{m_i}|)},$$

which is roughly equivalent to

$$2^{\ell \cdot |Vars(\gamma|} \cdot 2^{\ell \cdot |\gamma|}.$$

Note that for this argument only works for the case of nested UC2RPQs, in which we assume all γ^i_j to be different. When speaking about regular queries this argument is lost, and thus the bound on number of cuts is exponentially bigger. That each cut is polynomial in size can be shown using a similar argument. ◻

Transition system based on cuts

Looking to characterize the set of linearizations of nested UC2RPQs, it is best if we start by defining a transition system $T_{(\gamma,w)}$ defined over cuts of γ and positions of a word w over $\Sigma^{\pm}$.

Given a nested UC2RPQ and a string $w = w_1, \dots, w_k$ of size k from $\Sigma^{\pm}$, a *configuration* of $T_{(\gamma,w)}$ is just a pair from $\mathcal{C}uts(\gamma) \times \{1, \dots, k\}$. Intuitively, this means that a certain cut is assigned to a certain position of w. The system $T_{(\gamma,w)}$ relates configurations according to a transition relation $\Rightarrow_{(\gamma,w)}$ ranging over

$$\big(\mathcal{C}uts(\gamma) \times \{1, \dots, k\}\big) \times \big(\mathcal{C}uts(\gamma) \times \{1, \dots, k\}\big)$$

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 in Prev, that can be mapped to each position in w. For the same reason, the transition system can only move towards configurations in which Prev is not smaller that previous configurations.

We now proceed to define this relation. We begin with the set of transitions that relates configurations in subsequent positions. Assume that γ is of form (2), and that function $\Rightarrow_{(\gamma',w)}$ is defined for all nested UC2RPQ of nesting depth lower than γ. Then for a cut $C = (C^1, \dots, C^\ell)$ of γ and a position p of w, we let $(C,p) \Rightarrow_{(\gamma,w)} (C', p+1)$ for all cuts $C' = (C^{1\prime}, \dots, C^{\ell\prime})$ of γ such that the following holds:

Let L contain all indexes from $\{1, \dots, \ell\}$ such that C^i is not $\bot$ and L' contain all indexes from $\{1, \dots, \ell\}$ such that $C^{i\prime}$ is not $\bot$.

Furthermore, assume that each $C^i, i \in L$ is of form $\big(\mathrm{Prev}^i, \mathrm{Same}^i, (s^i_1, \dots, s^i_{m_i})\big)$ and each $C^{i\prime}, i \in L'$ is of form $\big(\mathrm{Prev}^{i\prime}, \mathrm{Same}^{i\prime}, (s^{i\prime}_1, \dots, s^{i\prime}_{m_i})\big)$. Then

1. $L' \subseteq L$ and $\mathrm{Prev}^i = \mathrm{Prev}^{i\prime}$ for each $i \in L'$;
2. $\mathrm{Same}^{i\prime} = \emptyset$ for each $i \in L'$.
3. For every $i \in L'$ and $1 \leq j \leq m_i$ such that both y^i_j and $y^{i\prime}_j$ are in Prev^i, or both are not in Prev^i, it holds that $s^i_j = s^{i\prime}_j$;
4. For every $i \in L'$ and $1 \leq j \leq m_i$ such that $y^i_j \in \mathrm{Prev}^i$ and $y^{i\prime}_j \notin \mathrm{Prev}^i$, then
   - If γ^i_j is a 2RPQ, then $s^{i\prime}_j \in \delta^i_j(s^i_j, w_p)$, or
   - If γ^i_j is itself a nested UC2RPQ, then $(s^i_j, p) \Rightarrow_{(\gamma^i_j,w)} (s^{i\prime}_j, p+1)$
5. For every $i \in L'$ and $1 \leq j \leq m_i$ such that $y^i_j \notin \mathrm{Prev}^i$ and $y^{i\prime}_j \in \mathrm{Prev}^i$, then
   - If γ^i_j is a 2RPQ, then $s^{i\prime}_j \in \delta^i_j(s^i_j, {w_p}^-)$, or
   - If γ^i_j is itself a nested UC2RPQ, then $(s^i_j, p) \Rightarrow_{(\gamma^i_j,w)} (s^{i\prime}_j, p+1)$

The first and second condition specifies that by reading symbols there are two possibilities: either C^i becomes $\bot$, or if it does not, one can not add extra variables to Prev^i so, for now, no new variables are added when advancing to a greater position. The third condition specifies that the condition of $\mathcal{S}^i$ is not altered if both variables are in Prev^i or none are. The last two conditions state that one can advance to cut C' only if the information in each $s^{i\prime}_j$ corresponds to a valid transition of the atom γ^i_j, be it a 2RPQ or a nested UC2RPQ with lower nesting depth.

Thus, $(C,p) \Rightarrow_{(\gamma,w)} (C', p+1)$ is defined only for those cuts C and C' where $\mathrm{Prev}^i = \mathrm{Prev}^{i\prime}$ for each $i \in L'$. Thus, no new variables can be added to cuts when *advancing* in the position of w. Instead, we just need to make sure that C' contains the correct transitions of all subqueries, with respect to C and w. For the same reason, note that we do not need the whole information of w to compute all cuts related to (C,p) via $\Rightarrow_{(\gamma,w)}$, but rather only the information of the p-th symbol of w. We use this important property later on in this proof.

So far we have not described how to change to cuts where more variables are added to Prev. Furthermore, we also need to define transitions that do not advance when reading the symbol, to allow for synchronization between the different components of γ. It is best if we start defining those for the case of nested UC2RPQs of depth 0. Assume then that γ has nesting depth 0.

For a cut $C = (C^1,\dots,C^\ell)$ of γ, define $(C,p) \Rightarrow_{(\gamma,w)} (C',p)$ for all cuts $C' = (C^{1'},\dots,C^{\ell'})$ of γ such that the following holds.

Assume that each C^i is of form $(\text{Prev}^i,\text{Same}^i,(s^i_1,\dots,s^i_{m_i})$ and each $C^{i'}$ is of form $(\text{Prev}^{i'},\text{Same}^{i'},({s^i_1}',\dots,{s^i_{m_i}}')$. Then for every $1 \le i \le \ell$ and $1 \le j \le m_i$, either $C^{i'}$ is $\perp$ or else one of the following holds:

- The sets Prev^i and $\text{Prev}^{i'}$ contain both variables in $\{y^i_j, {y^i_j}'\}$
- The sets Prev^i and $\text{Prev}^{i'}$ do not contain any of $\{y^i_j, {y^i_j}'\}$
- The set Prev^i does not contain any of $\{y^i_j, {y^i_j}'\}$, and exactly one of $\{y^i_j, {y^i_j}'\}$ is in both $\text{Same}^{i'}$ and $\text{Prev}^{i'}$, and $s^i_j = {s^i_j}'$ is an initial state.
- The set Prev^i contains exactly one variable in $\{y^i_j, {y^i_j}'\}$, the remaining variable belong to the set $\text{Same}^{i'}$, both of them are in $\text{Prev}^{i'}$, and $s^i_j = {s^i_j}'$ is a final state.
- Both Prev^i and to $\text{Prev}^{i'}$ contain y^i_j but not ${y^i_j}'$, and there is a computation of the automata for γ^i_j that starts in state s^i_j and ends in state ${s^i_j}'$, with the head pointed at the same position p of the string w.
- Both Prev^i and to $\text{Prev}^{i'}$ contain ${y^i_j}'$ but not y^i_j, and there is a computation of the automata for the inverse of γ^i_j that starts in state s^i_j and ends in state ${s^i_j}'$, with the head pointed at the same position p of the string w.

The first conditions states that one can always add the first variable of a given atom $\gamma^i_j(y^i_j, {y^i_j}')$ to Prev^i. The second states that the remaining variable can be added as long as s^i_j is a final state. Furthermore, the last three condition state that, for the remaining atoms, either both variables are in our out of the cut, or s^i_j represents a *loop* in the computation of the automata for γ^i_j (or its inverse).

Intuitively, if we move from (C,p) to (C',p'), it must be the case that, for each cut C^i of C that is not $\perp$ and each subquery $\gamma^i_j(y^i_j, {y^i_j}')$ of C^i, one of the following holds:

- this subquery is not important (because none or both of $\{y^i_j, {y^i_j}'\}$ are in Prev^i and $\text{Prev}^{i'}$, formalized in the first two conditions
- we can add the first variable of $\{y^i_j, {y^i_j}'\}$ to Prev^i, as long as the information about $\gamma^i_j(y^i_j, {y^i_j}')$ in $\mathcal{S}^i$ is an initial state (third condition),
- we can add the remaining variable in $\{y^i_j, {y^i_j}'\}$, as long as the information about $\gamma^i_j(y^i_j, {y^i_j}')$ in $\mathcal{S}^i$ is a final state (fourth condition),
- we can change the information about $\gamma^i_j(y^i_j, {y^i_j}')$ in $\mathcal{S}^i$ as long as there is a computation in the automaton for $\gamma^i_j(y^i_j, {y^i_j}')$ that starts and ends in the same position p.

▶ **Example 33.** Consider the CRPQ γ defined as $a^+(x,y), aa^-a(x,y)$, and the word $w = a$. When defining $\Rightarrow_{(\gamma,w)}$, consider the cut C in which variable x is in Prev but y is not. Clearly, we can advance for $(C,1)$ to $(C',2)$, where C' has the same information about x and y, but where the states of the automata in the tuple $\mathcal{S}'$ of C' are updated according to their transitions using symbol a. However. For the subquery $a^+(x,y)$ we have that this automata is already in a final state. But we cannot add variable y to Prev yet because the automaton corresponding to $aa^-a(x,y)$ is not in a final state. So first we need to advance from $(C',2)$ to another configuration $(C'',2)$ where both of the automaton in $\mathcal{S}''$ are in a final state, and afterwards we shall be able to move to a final cut where both x and y are in Prev. This reflects, of course, that there is a containment mapping from γ to the CQ (x,a,y).

When defining $\Rightarrow_{(\gamma,w)}$ for queries with higher nesting depth, there is an additional detail that needs to be taken care of: For symbols of form $P^+(x,y)$, the intuition is that one should be able to move from a final cut of P to an initial cut of P, reflecting the computation of the transitive closure of P. However, the important cut is not the final or the initial one, but rather the cut (and the position) where x and / or y was mapped: we can move from a cut and a position where y was just mapped, to a cut and a position where x has just been mapped (and y is still not in Prev).

We now finish the definition of $\Rightarrow_{(\gamma,w)}$ for queries of higher depth. For a cut $C = (C^1,\dots,C^\ell)$ of γ, define $(C,p) \Rightarrow_{(\gamma,w)} (C',p)$ for all cuts $C' = (C^{1\prime},\dots,C^{\ell\prime})$ of γ such that the following holds. Assume that each C^i is of form $(\text{Prev}^i, \text{Same}^i, (s^i_1,\dots,s^i_{m_i})$ and each $C^{i\prime}$ is of form $(\text{Prev}^{i\prime}, \text{Same}^{i\prime}, (s^{i\prime}_1,\dots,s^{i\prime}_{m_i})$. Then for every $1 \leq i \leq \ell$ and $1 \leq j \leq m_i$, where γ^i_j is a 2RPQ, either $C^{i\prime}$ is $\perp$ or else one of the following holds:

- The sets Prev^i and $\text{Prev}^{i\prime}$ contain both variables in $\{y^i_j, y^{i\prime}_j\}$
- The sets Prev^i and $\text{Prev}^{i\prime}$ do not contain any of $\{y^i_j, y^{i\prime}_j\}$
- The set Prev^i does not contain any of $\{y^i_j, y^{i\prime}_j\}$, and exactly one of $\{y^i_j, y^{i\prime}_j\}$ is in both $\text{Same}^{i\prime}$ and $\text{Prev}^{i\prime}$, and $s^i_j = s^{i\prime}_j$ is an initial state.
- The set Prev^i contains exactly one variable in $\{y^i_j, y^{i\prime}_j\}$, the remaining variable belong to the set $\text{Same}^{i\prime}$, both of them are in $\text{Prev}^{i\prime}$, and $s^i_j = s^{i\prime}_j$ is a final state.
- Both Prev^i and to $\text{Prev}^{i\prime}$ contain y^i_j but not $y^{i\prime}_j$, and there is a computation of the automata for γ^i_j that starts in state s^i_j and ends in state $s^{i\prime}_j$, with the head pointed at the same position p of the string w.
- Both Prev^i and to $\text{Prev}^{i\prime}$ contain $y^{i\prime}_j$ but not y^i_j, and there is a computation of the automata for the inverse of γ^i_j that starts in state s^i_j and ends in state $s^{i\prime}_j$, with the head pointed at the same position p of the string w.

This is, of course, the same as before. Furthermore, for every $1 \leq i \leq \ell$ and $1 \leq j \leq m_i$, where $C^{i\prime}$ is not $\perp$ and γ^i_j is a nested UC2RPQ, assume that each s^i_j is a cut of form $(\text{Prev}^i_j, \text{Same}^i_j, \mathcal{S}^i_j)$ and $s^{i\prime}_j$ is a cut of form $(\text{Prev}^{i\prime}_j, \text{Same}^{i\prime}_j, \mathcal{S}^i_j)'$. Then one of the following holds:

- The sets Prev^i and $\text{Prev}^{i\prime}$ contain both variables in $\{y^i_j, y^{i\prime}_j\}$
- The sets Prev^i and $\text{Prev}^{i\prime}$ do not contain any of $\{y^i_j, y^{i\prime}_j\}$
- The set Prev^i does not contain any of $\{y^i_j, y^{i\prime}_j\}$, exactly one variable $y \in \{y^i_j, y^{i\prime}_j\}$ is in both $\text{Same}^{i\prime}$ and $\text{Prev}^{i\prime}$, and (1) Variable y also belongs to the set $\text{Same}^{i\prime}_j$, (2) there is an initial cut $\hat{c}$ and a position $\hat{p}$ of w such that one can advance from $(\hat{c},\hat{p})$ to (s^i_j,p) using $\Rightarrow_{(\gamma^i_j,w)}$; and (3) $(s^i_j,p) \Rightarrow_{(\gamma^i_j,w)} (s^{i\prime}_j,p)$.
- The set Prev^i contains exactly one variable $y \in \{y^i_j, y^{i\prime}_j\}$, the remaining variable y' belongs to the set $\text{Same}^{i\prime}$, both of them are in $\text{Prev}^{i\prime}$, and (1) $\text{Same}^{i\prime}_j$ contains y', (2) $(s^i_j,p) \Rightarrow_{(\gamma^i_j,w)} (s^{i\prime}_j,p)$, and (3) there is a final cut c of γ^i_j and a position p' of w such that one can advance from $(s^{i\prime}_j,p)$ to (c,p') using $\Rightarrow_{(\gamma^i_j,w)}$.

- Prev^i and $\text{Prev}^{i'}$ both contain one of y^i_j or $y^{i'}_j$, and one can advance from (s^i_j, p) to $(s^{i'}_j, p)$ using $\Rightarrow_{(\gamma^i_j, w)}$.
- Prev^i and $\text{Prev}^{i'}$ both contain y^i_j, and (1) Same^i_j contains $y^{i'}_j$, (2) there is a position $\hat{p}$ and a final cut $\hat{c}$ of γ^i_j such that $(s^i_j, p) \Rightarrow (\gamma^i_j, w)(\hat{c}, \hat{p})$, (3) $\text{Same}^{i'}_j$ contains y^i_j and (4) there is a position $\hat{p}'$ and an initial cut $\hat{c}'$ of γ^i_j such that $(\hat{c}', \hat{p}') \Rightarrow (\gamma^i_j, w)(s^{i'}_j, p)$.
- Symmetrical to the previous condition where instead Prev^i and $\text{Prev}^{i'}$ both contain $y^{i'}_j$.

The first two conditions are the same as before, i.e., this subquery is of not importance for now. The intuition for the remaining conditions is as follows. If we move from (C, p) to (C', p'), it must be the case that, for each cut C^i of C and each subquery $\gamma^i_j(y^i_j, y^{i'}_j)$ of C^i, one of the following holds:

- we can add the first variable of $\{y^i_j, y^{i'}_j\}$ to Prev^i. In this case it must be true that there is a previous initial cut of γ^i_j and a path from this initial cut to another cut in which the variable to be added is mapped precisely to position p.
- we can add the remaining variable in $\{y^i_j, y^{i'}_j\}$ to Prev^i. In this case it must be true that there is a path from the current cut of γ^i_j in $\mathcal{S}^i$ to a final cut of γ^i_j, and that the remaining variable is added to the cut of γ^i_j in $\mathcal{S}^i$.
- we can change the information about $\gamma^i_j(y^i_j, y^{i'}_j)$ in $\mathcal{S}^i$ as long as it is dictated by $\Rightarrow_{(\gamma^i_j, w)}$
- The last two possibilities account for the computation of the transitive closure operator, going from a final cut of γ^i_j to an initial cut of γ^i_j. In this cae we can also change the information about $\gamma^i_j(y^i_j, y^{i'}_j)$ in $\mathcal{S}^i$, to reflect that we are taking a new recursive step while computing the transitive closure of this predicate. We need to be able to move from the previous cut in $\mathcal{S}^i_j$ to a final state, and also from an initial state to the state in $\mathcal{S}^i_j$, satisfying similar conditions as when one adds one of the variables $\{y^i_j, y^{i'}_j\}$ to Prev.

To state the correctness of this system, we define a special type of run. Formally, given a nested UC2RPQ $\gamma(x, y)$ and a word w, then (C, p) and (C', p') define an *accepting run* for γ over w if the following holds:

- C marks x and C' marks y
- There is an initial cut C_I and a position p_I such that one can go from (C_I, p_I) to (C, p) by means of $\Rightarrow_{(\gamma, w)}$.
- There is a final cut C_F and a position p_F such that one can go from (C', p') to (C_F, p_F) by means of $\Rightarrow_{(\gamma, w)}$.

▶ **Lemma 34.** *Let $\gamma(x, y)$ be a nested UC2RPQ and w a string over Σ with valid annotations w.r.t. γ. There are pairs (C, p) and (C', p') over $Cuts(\gamma) \times \{1, \dots, |w| + 1\}$ such that (C, p) and (C', p') define an* accepting run *for γ over w if and only if there is an expansion q of γ and a containment mapping from q to the linear CQ $Q_w = A_1(z_1, z_2), \dots A_k(z_k, z_k)$ given by the projection $\tau(w)$ of w over Σ that maps x and y to variables z_{p_x} and z_{p_y} of Q_w.*

Proof: The proof is by induction. We begin with the **If** direction. Let γ be a nested UC2RPQ and w a string over $\Sigma(\gamma) \times \Sigma^{\pm}$ with valid annotations w.r.t. γ. Furthermore, assume that there is an expansion q of γ and a containment mapping from q to the linear CQ Q_w given by the projection $\tau(w)$ of w over Σ.

Assume also for the sake of simplicity that all rules mentioning γ are of the following form, in which all subrules are itself the transitive closure of a nested UC2RPQ.

$$\begin{array}{rcl} \gamma(x_1,\dots,x_n) & \leftarrow & \gamma_1^1(y_1^1,{y^1}'_1),\dots,\gamma_{m_1}^1(y_{m_1}^1,{y^1}'_{m_1}), \\ \vdots & & \vdots \\ \gamma(x_1,\dots,x_n) & \leftarrow & \gamma_1^\ell(y_1^\ell,{y^\ell}'_1),\dots,\gamma_{m_\ell}^\ell(y_{m_\ell}^\ell,{y_{m_\ell}^\ell}'), \end{array} \tag{4}$$

Let $Vars(\gamma)$ be the variables in γ. It is clear from the definition of expansion that there must be an expansion for each subquery $\gamma_j^i(y_j^i,{y_j^i}')$ that can be mapped via a containment mapping to Q_w, sending y_j^i and ${y_j^i}'$ to positions, say p_j^i and $\hat{p_j^i}'$, and thus that the hypothesis holds for $\gamma_j^i(y_j^i,{y_j^i}')$. Using the information from the containment mapping and the information of all the cuts from the hypothesis of each $\gamma_j^i(y_j^i,{y_j^i}')$, it is straightforward to show that there are cuts $C_I = (\text{Prev}_I,\text{Same}_I,\mathcal{S}_I)$, $C_F = (\text{Prev}_F,\text{Same}_F,\mathcal{S}_F)$, $C_x = (\text{Prev}_x,\text{Same}_x,\mathcal{S}_x)$, $C_y = (\text{Prev}_y,\text{Same}_y,\mathcal{S}_y)$ be cuts of γ, where C_I is an initial cut, C_F is a final cut, C_x is a cut where $x \in \text{Same}_x$ and $y \notin \text{Same}_x$ and C_y is a cut with $y \in \text{Same}_y$ and $x \in \text{Prev}_y$, with $(C_I,p_I) \Rightarrow_{(\gamma,\tau(w))} (C_x,p_x) \Rightarrow_{(\gamma,\tau(w))} (C_y,p_y) \Rightarrow_{(\gamma,\tau(w))} (C_F,p_F)$. The construction of these cuts depends of how the variables are mapped according to the containment mapping.

For the **only if** direction, assume that there are cuts $C_I = (\text{Prev}_I,\text{Same}_I,\mathcal{S}_I)$, $C_F = (\text{Prev}_F,\text{Same}_F,\mathcal{S}_F)$, $C_x = (\text{Prev}_x,\text{Same}_x,\mathcal{S}_x)$, $C_y = (\text{Prev}_y,\text{Same}_y,\mathcal{S}_y)$ be cuts of γ, where C_I is an initial cut, C_F is a final cut, C_x is a cut where $x \in \text{Same}_x$ and $y \notin \text{Same}_x$ and C_y is a cut with $y \in \text{Same}_y$ and $x \in \text{Prev}_y$, with $(C_I,p_I) \Rightarrow_{(\gamma,\tau(w))} (C_x,p_x) \Rightarrow_{(\gamma,\tau(w))} (C_y,p_y) \Rightarrow_{(\gamma,\tau(w))} (C_F,p_F)$.

Now considering a run that witness the fact that $(C_I,p_I) \Rightarrow_{(\gamma,\tau(w))} (C_x,p_x) \Rightarrow_{(\gamma,\tau(w))} (C_y,p_y) \Rightarrow_{(\gamma,\tau(w))} (C_F,p_F)$, for each position p in w we let C^p be the greatest cut (in terms of the size of Prev and Same) that was used in such run. Moreover, consider the sequence $C^1,\dots,C^{|w|+1}$ of cuts, and choose an $1 \leq i \leq \ell$ such that the i-th component of $C^{|w|+1}$ is not $\perp$.

We focus on the triples $(\text{Prev}^i,\text{Same}^i,\mathcal{S}^i)$ of each such cut C^p. For simplicity let us abuse notation and denote the i-th component of each C^p by $(\text{Prev}^p,\text{Same}^p,\mathcal{S}^p)$

From the definition of $\Rightarrow_{(\gamma,\tau(w))}$ we have that each variable in γ must be contained to only one of $\text{Same}^1,\dots,\text{Same}^{|w|+1}$. Let then σ be the containment mapping that sends each x in γ to the position p such that $x \in \text{Same}^p$. Using an inductive argument and the fact that $(C_I,p_I) \Rightarrow_{(\gamma,\tau(w))} (C_x,p_x) \Rightarrow_{(\gamma,\tau(w))} (C_y,p_y) \Rightarrow_{(\gamma,\tau(w))} (C_F,p_F)$, it is not not difficult, although cumbersome, to show that there is an expansion q of γ and a containment mapping from q to the linear CQ $Q_w = A_1(z_1,z_2),\dots A_k(z_k,z_k)$ given by the projection $\tau(w)$ of w over Σ that maps x and y to variables z_{p_x} and z_{p_y} of Q_w. ☐

Extending the alphabet to account for computations of $T_{(\gamma,w)}$

A straightforward idea to continue with the proof is to use the system $T_{(\gamma,w)}$ to create an automaton that accepts all strings that represent the set of flinearizations 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.

In a nutshell, given γ and w we have to extend the symbols of w with information about the cuts reachable from configurations of form (c,p) in $T_{(\gamma,w)}$, where $1 \leq p \leq |w|$.

Formally, from $\Sigma^{\pm}$ we construct the extended alphabet $\Sigma(\gamma) \times \Sigma^{\pm}$ as follows. Assume that $Cuts(\gamma)$ contains a number N of cuts. Then

- If γ is a nested UC2RPQ of depth 0, $\Sigma(\gamma)$ is an $N+2$ tuple of subsets of $Cuts(\gamma)$, i.e., $\Sigma(\gamma) = (2^{Cuts(\gamma)})^{N+2}$.

In other words, $\Sigma^{\pm}$ contains a subset of $Cuts(\gamma)$ for each cut in $Cuts(\gamma)$.

- Otherwise assume that γ is of form (2), and that $\mathcal{P}$ is the set of predicates occurring in the rules of γ that are not 2RPQs. Then

$$\Sigma(\gamma) = (2^{Cuts(\gamma)})^{N+2} \times \bigtimes_{P \in \mathcal{P}} \Sigma(P),$$

In other words, it is the cartesian product of the set $(2^{Cuts(\gamma)})^{N+2}$ with each $\Sigma(P)$, for every predicate P that is a subquery of γ.

Let us now give the intuition of this extension to Σ. We first need some notation. From $\Sigma(\gamma)$ we construct a function f that assigns to each symbol $u \in \Sigma(\gamma)$ and each cut of γ, the subset $\mathcal{C}$ of cuts assigned to C, i.e., if C is the M-th cut of γ, then $f(u, C)$ corresponds to the subset given by the $M+2$-th set in the first tuple in the symbol u of $\Sigma(\gamma)$. The function is extended to those cuts of subqueris in γ in a natural way.

Now, let w be a string from $\Sigma(\gamma) \times \Sigma^{\pm}$, and let $\tau(w)$ be its projection over $\Sigma^{\pm}$. Then note that each symbol (u_p, a_p), for $u_p \in \Sigma(\gamma)$ contains, in particular, $N+2$ subsets of $Cuts(\gamma)$, where $N = |Cuts(\gamma)|$. The first subset of $Cuts(\gamma)$ represent all those cuts C such that there is a position $\hat{p}$ in w and an initial cut $\hat{C}$ of γ such that (C, p) is reachable from $(\hat{c}, \hat{p})$ using $\Rightarrow_{(\gamma,\tau(w))}$. Moreover, the second subset of $Cuts(\gamma)$ contains all those cuts C such that there is a final cut $\hat{C}$ and a position $\hat{p}$ so that $(\hat{C}, \hat{p})$ can be reached from (C, p) using $\Rightarrow_{(\gamma,\tau(w))}$. For the remaining N subsets, note that there is one subset of $Cuts(\gamma)$ remaining for each cut C of γ, namely the subset $\mathcal{C}$ returned by $f(u_p, C)$. For each such cut C, its corresponding subset $\mathcal{C}$ contains all those cuts $\hat{C}$ for which the configuration $(\hat{C}, p)$ is reachable from (C, p) using $\Rightarrow_{(\gamma,\tau(w))}$.

If a string w from $\Sigma(\gamma) \times \Sigma^{\pm}$ satisfies the above conditions, we say that w has *valid annotations* w.r.t. γ.

▶ **Example 35.** Consider again query

$$\begin{array}{rcl}
G_1(a,b) & \leftarrow & g(a,b) \\
G_2(c,d) & \leftarrow & g(c,d) \\
R(p,q) & \leftarrow & f(p,q) \\
R(u,v) & \leftarrow & k(u,v), G_1^+(u,z), G_2^+(v,z) \\
Ans(x,y) & \leftarrow & R^+(x,y)
\end{array}$$

over alphabet $\Sigma = \{g, f, k\}$. Then, assuming $N_{G_1} = |Cuts(G_1)|$,

$$\Sigma(G_1) = \{(a_1, \ldots, a_{N_{G_1}+2}) \mid a_i \subseteq Cuts(G_1)\},$$

and likewise for $\Sigma(G_2)$. If $N_R = |Cuts(R)|$, then

$$\Sigma(R) = \{(a_1, \ldots, a_{N_R+2}, c_1, \ldots, b_{N_{G_1}+2}, c_1, \ldots, c_{N_{G_2}+2}) \mid a_i \subseteq Cuts(R), b_j \subseteq Cuts(G_1), c_k \subseteq Cuts(G_2)\}.$$

Finally, $\Sigma(\gamma)$ can be defined in a similar way.

The following follows directly from Lemma 32.

▶ **Lemma 36.** *Let γ be a nested U2CRPQ. Then the number of different symbols in $\Sigma(\gamma)$ is double exponential w.r.t. the size of γ. Furthermore, each symbol in $\Sigma(\gamma)$ is of size exponential w.r.t. γ*

Furthermore, we show

▶ **Lemma 37.** *Let γ be a nested U2CRPQ. Then the language of all strings over $\Sigma(\gamma) \times \Sigma^{\pm}$ that have* valid annotations *w.r.t. γ is regular. Furthermore, one can build an NFA that checks this language of size double-exponential in the size of γ.*

Proof: Let γ be a nested U2CRPQ. Then the language of all strings over $\Sigma(\gamma) \times \Sigma^{\pm}$ that have *valid annotations* w.r.t. γ is regular. Furthermore, one can build an NFA that checks this language of size double-exponential in the size of γ.

It suffices to build the desired NFA. As usual, we define it in an inductive fashion.

Assume first γ is a UC2RPQ of depth 0, following the form

$$\begin{array}{rcl} \gamma(x_1,\dots,x_n) & \leftarrow & \gamma_1^1(y_1^1,{y^1}'_1),\dots,\gamma_{m_1}^1(y_{m_1}^1,{y^1}'_{m_1}), \\ \vdots & & \vdots \\ \gamma(x_1,\dots,x_n) & \leftarrow & \gamma_1^\ell(y_1^\ell,{y^\ell}'_1),\dots,\gamma_{m_\ell}^\ell(y_{m_\ell}^\ell,{y_{m_\ell}^\ell}'), \end{array} \tag{5}$$

Then notice that $\Sigma(\gamma)$ just relates the automaton that checks for for γ_j^i with all the states they could reach from a given point. The desired automaton can be obtained using standard tools to transform two way automata into deterministic one way automata (c.f. J.C. Shepherdson, *The reduction of two-way automata to one-way automata*, IBM J Res., 3(1959), pp. 198-200). To be more precise, for each 2RPQ γ_j^i that appears in γ, one needs to construct three NFAs: $A^{\rightarrow}_{\gamma_j^i}$, $A^{\leftarrow}_{\gamma_j^i}$ and $A^{\leftrightarrow}_{\gamma_j^i}$. Intuitively, $A^{\rightarrow}_{\gamma_j^i}$ checks all pairs of states q, q' for which there is a computation of γ_j^i that goes to the left and comes back to the same point, starting in q and ending in q'. $A^{\leftarrow}_{\gamma_j^i}$ checks for computation that only go backwards and then return. Finally, $A^{\leftrightarrow}_{\gamma_j^i}$ integrates both the information of both automata to compute precisely those computations in which we can start in state q, end in state q', but by going into any possible direction. Note that these automaton are of exponential size w.r.t. γ.

Next, if γ is a UC2RPQ of depth $k > 0$. Assume that for all UC2RPQs γ' of depth $< k$ that appear in γ, automata $A^{\leftrightarrow}$ represents the product of all such $A^{\leftrightarrow}_{\gamma}$. For simplicity, assume that all subqueries are UC2RPQs (so that none is a 2RPQ).

Then the NFA that checks if a string from $\Sigma(\gamma) \times \Sigma$ has valid annotations w.r.t. γ is defined as $A^*_{\Sigma(\gamma)} = (q, \Sigma(\gamma) \times \Sigma, q, \{q\}, \delta)$ is an automaton with a single state, that just *filters* the admitted symbols in $\Sigma(\gamma) \times \Sigma$, assuming they are already valid w.r.t. queries of lower depth.

More precisely, let (u, a) be a symbol from $\Sigma(\gamma) \times \Sigma$. Then a symbol (u, a) is *valid* if and only if the following holds:

For every cut $C = (C^1, \dots, C^\ell)$ of γ, where each C^i is of form $(\text{Prev}^i, \text{Same}^i, (s_1^i, \dots, s_{m_i}^i)$, a cut $C' = ({C^1}', \dots, {C^\ell}')$, each ${C^i}'$ is of form $({\text{Prev}^i}', {\text{Same}^i}', ({s_1^i}', \dots, {s_{m_i}^i}')$, is in $f(u, C)$ if and only if the following holds

For every $1 \leq i \leq \ell$ and $1 \leq j \leq m_i$, assume that γ_j^i is a nested UC2RPQ and each s_j^i is a cut of form $(\text{Prev}_j^i, \text{Same}_j^i, \mathcal{S}_j^i)$ and ${s_j^i}'$ is a cut of form $({\text{Prev}_j^i}', {\text{Same}_j^i}', \mathcal{S}_j^i)'$ (the case when γ is a nested UC2RPQ is analogous). Then one of the following holds:

- The sets Prev^i and ${\text{Prev}^i}'$ contain both variables in $\{y_j^i, {y_j^i}'\}$
- The sets Prev^i and ${\text{Prev}^i}'$ do not contain any of $\{y_j^i, {y_j^i}'\}$
- The set Prev^i does not contain any of $\{y_j^i, {y_j^i}'\}$, exactly one variable $y \in \{y_j^i, {y_j^i}'\}$ is in both ${\text{Same}^i}'$ and ${\text{Prev}^i}'$, and (1) Variable y also belongs to the set ${\text{Same}_j^i}'$, (2) $f_{\gamma_j^i}(u, \mathit{initial})$ contains s_j^i; and (3) $f_{\gamma_j^i}(u, s_j^i)$ contains ${s_j^i}'$.

- The set Prev^i contains exactly one variable $y \in \{y^i_j, y^{i'}_j\}$, the remaining variable y' belongs to the set $\text{Same}^{i'}$, both of them are in $\text{Prev}^{i'}$, and (1) $\text{Same}^{i'}_j$ contains y', (2) $f_{\gamma^i_j}(u, s^i_j)$ contains $s^{i'}_j$, and (3) $f_{\gamma^i_j}(u, \mathit{final})$ contains $s^{i'}_j$.
- Prev^i and $\text{Prev}^{i'}$ both contain one of y^i_j or $y^{i'}_j$, and $f_{\gamma^i_j}(u, s^i_j)$ contains $s^{i'}_j$.
- Prev^i and $\text{Prev}^{i'}$ both contain y^i_j, and (1) Same^i_j contains $y^{i'}_j$, (2) $f_{\gamma^i_j}(u, \mathit{final})$ contains s^i_j, (3) $\text{Same}^{i'}_j$ contains y^i_j and (4) $f_{\gamma^i_j}(u, \mathit{initial})$ contains $s^{i'}_j$.
- Symmetrical to the previous condition where instead Prev^i and $\text{Prev}^{i'}$ both contain $y^{i'}_j$.

In other words, valid symbols contain all the information regarding $\Rightarrow_{(\gamma,w)}$. Then δ just ensure that all read symbols are valid. Note that we can check for validity without actually checking the word since everything has already been coded into the cuts of the γ^i_j's that are queries of depth 0.

Thus, by defining the desired automaton $A_{\Sigma(\gamma)}$ as the product of $A^{\leftrightarrow}_\gamma$ and $A^*_{\Sigma(\gamma)}$, we get the desired information.

Regarding the size of $A^*_{\Sigma(\gamma)}$, observe that δ is essentially a list of which symbols are allowed and which aren't, and the number of symbols is double exponential size w.r.t. γ, clearly $A_{\Sigma(\gamma)}$ is doubly-exponential. Furthermore, it is clear from the definition that it can be computed in exponential space. ◻

Cut Automata. Finally, we proceed to build the desired NFA that gives the strings corresponding to the linearizations of nested UC2RPQs. This NFA needs to simulate the relation $\Rightarrow_\gamma$, from an initial cut of a nested UC2RPQ γ to a final cut in which all variables have already been mapped. Of course, we have to check, for every subquery $\gamma^i_j(y^i_j, y^{i'}_j)$ of γ, whether this query is indeed satisfied when starting in the position assigned to variable y^i_j and finishing on the position assigned to $y^{i'}_j$. Since we might need to check for more than one such query at any given point, synchronizing all these checks is non-trivial. To overcome this problem, we use the information stored in strings of the extended alphabet that have valid annotations.

Let then γ be a nested UC2RPQ for form 2, and consider alphabet $\Sigma(\gamma)$ as above. For simplicity, and without incidence in our results, we assume for now that NFAs can decide not to move to the following symbol of the string they are reading, i.e., we assume that the transition function for an NFA $A = (Q, \Sigma, q_0, F, \delta)$ is defined from $Q \times \Sigma$ to subsets of pairs of form $Q \times \{1, 0\}$, where 1 stands for *move* the head and 0 stand for *stay*.

The cut automata for γ is an NFA $A_\gamma = (Q, \Sigma(\gamma) \times \Sigma^\pm, q_0, q_f, \delta)$, where $Q = \{q_0, q_f\} \cup \{q_C \mid C \in \mathit{Cuts}(\gamma)\}$ is the set of states, F contains all final cuts of Γ, and δ is defined as follows:

- For each $C \in \mathit{Cuts}(\gamma)$ and $(u, a) \in \Gamma \times \Sigma^\pm$, $\delta(q_C, (u, a))$ contains all pairs $(q_{C'}, 1)$ such that $(C, 1) \Rightarrow_{(\gamma,a)} (C', 2)$
- For each $C \in \mathit{Cuts}(\gamma)$ and $(u, a) \in \Gamma \times \Sigma^\pm$, $\delta(q_0, (u, a)) = \{(q_0, 1)\}$ and $\delta(q_f, (u, a)) = \{(q_f, 1)\}$.
- For each final cut C of γ, and $(u, a) \in \Gamma \times \Sigma^\pm$, $\delta(q_C, (u, a))$ contains $(q_f, 1)$
- For each $C \in \mathit{Cuts}(\gamma)$ and $(u, a) \in \Gamma \times \Sigma^\pm$, $\delta(q_C, (u, a))$ contains all pairs $(q_{C'}, 0)$ such that cut C' is in the set $f(u, C)$, i.e., in the subset corresponding to C in u.

It is clear that A_γ is of size at most exponential w.r.t. γ. The following proposition shows its correctness. It follows from Lemma 34 and the fact that A_γ accepts w iff there are pairs (C, p) and (C', p') over $Cuts(\gamma) \times \{1, \ldots, |w| + 1\}$ such that (C, p) and (C', p') define an *accepting run* for γ over w.

▶ Proposition 38. Let γ be a nested UC2RPQ and w a string over $\Sigma(\gamma) \times \Sigma^{\pm}$ with valid annotations w.r.t. γ. Then A_γ accepts w if and only if there is an expansion q of γ and a containment mapping from q to the linear CQ $Q_w = A_1(z_1, z_2), \ldots A_k(z_k, z_k)$ given by the projection $\tau(w)$ of w over Σ that maps x and y to variables z_{p_x} and z_{p_y} of Q_w.

Main Proof. We now show how to decide containment of a 2RPQ in a nested UC2RPQ. Let ϕ and γ be boolean RPQ and nested UC2RPQ, respectively.

In light of this result, we can decide whether ϕ is contained in γ in the following way:

- Build an NFA A_ϕ for ϕ, extended so that it works with the alphabet $\Sigma(\gamma) \times \Sigma^{\pm}$.
- Build the NFA $A_{\Sigma(\gamma)}$ that checks for strings over $\Sigma(\gamma) \times \Sigma^{\pm}$ that are valid w.r.t. γ.
- Build the NFA A_γ, and complement it, obtaining the automaton $(A_\gamma)^C$.

The language of $(A_\gamma)^C$ intersected with the language of $A_{\Sigma(\gamma)}$ is precisely those strings w with valid annotations such that its projection $\tau(w)$ over Σ does not correspond to any linearization of γ. Thus, if we intersect this language with the one of (A_E), we have that the resulting intersection is nonempty if and only if there is an expansion q for E that does not correspond to any of the linearizations of γ can be mapped into q, i.e., if E is not contained in γ.

Even though some of these automata can be of double exponential size w.r.t. ϕ and γ, We can perform this algorithm in EXPSPACE using a standard on-the-fly implementation.

A.4 Proof of Proposition 16

Let Ω be a regular query and let d, h, w be its depth, height and width, respectively. We translate Ω to a nested UC2RPQ in the natural way: we unfold Ω and if it is necessary we rename intensional predicates. At the beginning, our equivalent nested UC2RPQ Γ is the empty set. We start with rules of the form $Ans(x_1, \ldots, x_n) \leftarrow \rho$. Pick a rule in Ω of this form. For each atom in ρ of the form $P(x, y)$, with P an intensional predicate, we choose any rule $P(x, y) \leftarrow \rho' \in \mathsf{rules}(P)$, rename all the variables in ρ', except by x, y, by new fresh variables, and substitute $P(x, y)$ by ρ' in the body ρ. After we do this for each atom in ρ, we end up with a rule of the form $Ans(x_1, \ldots, x_n) \leftarrow \rho_1$. The size of ρ_1 is at most w^2. We continue the process with $Ans(x_1, \ldots, x_n) \leftarrow \rho_1$. For each atom $R(x, y)$ in ρ_1, we perform a substitution as before. We end up with a rule $Ans(x_1, \ldots, x_n) \leftarrow \rho_2$, where the size of ρ_2 is at most w^3. We continue this process until we end up with a rule $Ans(x_1, \ldots, x_n) \leftarrow \rho_k$, where each atom in ρ_k is either an EDB or a transitive closure atom $S^+(x, y)$ (thus the rule is an extended C2RPQ rule). Observe that the number of iteration in this process cannot exceed the depth of the program Ω. Thus we have that $k \leq d$ and the size of ρ_k is at most w^{d+1}.

Observe we have many choices to generate the rule $Ans(x_1, \ldots, x_n) \leftarrow \rho_k$ according to the different choices in $\mathsf{rules}(P)$ when we substitute an atom $P(x, y)$. A simple counting argument shows that we have at most $h^w h^{w^2} \cdots h^{w^d} = h^{O(w^d)}$ choices to generate $Ans(x_1, \ldots, x_n) \leftarrow \rho_k$. We add to Γ all the rules of the form $Ans(x_1, \ldots, x_n) \leftarrow \rho_k$. We repeat this for each rule of the form $Ans(x_1, \ldots, x_n) \leftarrow \rho$ in Ω (there is at most h of these). Summing up, we add at most $h \cdot h^{O(w^d)} = h^{O(w^d)}$ rules to Γ, and the size of the body of each rule is at most w^{d+1}. In particular, $|\mathsf{rules}(Ans)|$ in Γ is $h^{O(w^d)}$.

Now, pick any rule $Ans(x_1, \ldots, x_n) \leftarrow \eta \in \mathsf{rules}(Ans)$ in Γ. Let $P_1, \ldots, P_k$ be all the intensional predicates in η. We rename these predicates with fresh names $P'_1, \ldots, P'_k$ to ensure that they are all distinct (and ensure condition (5) of nested UC2RPQs). Now, for each $1 \leq i \leq k$, we rename P_i by P_i' in the set $\mathsf{rules}(P_i)$ of Ω. We repeat the process described before, but instead of considering the predicate Ans, we consider the predicate P_i'. Observe that the same bounds apply: We add at most $h^{O(w^d)}$ rules to $\mathsf{rules}(P_i')$ in Γ and the size of the body of these rules is at most w^{d+1}. We repeat this process iteratively until all the atoms in our rules are EDBs.

Clearly, the resulting query Γ is an equivalent nested UC2RPQ. Moreover, $|\mathsf{rules}(P')|$ is at most $h^{O(w^d)}$, for all predicates in Γ, which implies that the height of Γ is at most $h^{O(w^d)}$. The size of the rules body is always bounded by w^{d+1}, thus the width of Γ is at most w^{d+1}. Finally, note that we never increase the depth, thus the depth of Γ is at most d. This proves the proposition.

A.5 Proof of Theorem 17

The proof is based on ideas from [19]. 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 M's 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 regular queries Ω and Ω' that encode accepting computations of M. An expansion of Ω represents a sequence of configurations. The role of Ω' is to detect *errors* that prevent this sequence from being an accepting computation. Thus, accepting computations are identified with expansion of Ω without errors, that is, such that Ω' cannot be mapped to it. Hence, we shall have that M accepts the empty tape iff Ω is not contained in Ω'. 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 [19], 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 $c_{2^n} = 1$, $c_i = 1$ iff $a_{i+1} = 1$ and $c_{i+1} = 1$, for $1 \leq i < 2^n$, and $b_i = 0$ iff $a_i = c_i$, for $1 \leq i \leq 2^n$.

Now we define Ω. We have extensional predicates `E,$, Start, IsAddress, IsSymbol, Zero, One, Carry0` and `Carry1`. For each configuration symbol a, we also have an extensional predicate $\mathtt{Q}_a$. The intensional predicates are $Bit, ConfAddress, ConfSymbol, Comp$ and $Final$. The query is boolean, so Ans is a 0-ary predicate. We have rules

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

In the first four rules, the variable x represents a position in an address. We have to consider the four possible values for the address bit and carry bit. The atom $\texttt{E}(x,y)$ connects adjacent positions. The predicate $ConfAddress$ describes sequences of address and carry bits. We also have rules

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

The predicate $ConfSymbol$ describes an address, followed by a symbol a. The atom $\texttt{E}(z,y)$ connects symbols in the same configuration. The atom $\texttt{\$}(z,y)$ 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 Ω rules of the form

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

for symbols a of the form $a = (s, a')$, where s is an accepting state. Finally, to encode a computation we use the following rule

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

The intuition is that each expansion of Ω 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 Ω' to detect errors in expansions of Ω. For each $0 \leq i \leq n$, we have intensional predicates $dist_i, dist_{\leq i}, dist_{<i}$ and $equal_i$. For each $0 < i \leq n$, we have rules

$$dist_i(x,y) \leftarrow dist_{i-1}(x,z), dist_{i-1}(z,y).$$

and the rule

$$
\begin{aligned}
dist_0(x,y) &\leftarrow \texttt{E}(x,y).\\
dist_0(x,y) &\leftarrow \texttt{\$}(x,y).
\end{aligned}
$$

Clearly, the predicate $dist_i(x,y)$ holds precisely when there is a path of length 2^i from x to y, consisting of $\texttt{E}$-labeled or $\texttt{\$}$-labeled edges. For each $0 < i \leq n$, we also have rules

$$dist_{\leq i}(x,y) \leftarrow dist_{\leq i-1}(x,z), dist_{\leq i-1}(z,y).$$
$$dist_{<i}(x,y) \leftarrow dist_{<i-1}(x,z), dist_{\leq i-1}(z,y).$$

and the rules

$$dist_{\leq 0}(x,y) \leftarrow \texttt{E}(x,y).$$
$$dist_{\leq 0}(x,y) \leftarrow \texttt{\$}(x,y).$$
$$dist_{\leq 0}(x,x) \leftarrow \mathsf{true}.$$
$$dist_{<0}(x,x) \leftarrow \mathsf{true}.$$

Here, $dist_{\leq i}$ hold precisely when there is a path of length at most 2^i from x to y, and $dist_{<i}(x,y)$ holds precisely when there is a path of length at most $2^i - 1$ from x to y (again, the paths consist of E-labeled or \$-labeled edges). Rules of the form $S(x,x) \leftarrow \mathsf{true}.$ can be easily simulated by Datalog rules. Now we define the $equal_i$ predicates. For each $0 < i \leq n$, we have rules

$$equal_i(x,y) \leftarrow equal_{i-1}(x,y), equal_{i-1}(x',y'), dist_{i-1}(x,x'), dist_{i-1}(y,y').$$

and the rules

$$equal_0(x,y) \leftarrow \texttt{E}(x,x'), \texttt{E}(y,y'), \texttt{Zero}(x,x), \texttt{Zero}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{E}(x,x'), \texttt{\$}(y,y'), \texttt{Zero}(x,x), \texttt{Zero}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{\$}(x,x'), \texttt{E}(y,y'), \texttt{Zero}(x,x), \texttt{Zero}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{\$}(x,x'), \texttt{\$}(y,y'), \texttt{Zero}(x,x), \texttt{Zero}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{E}(x,x'), \texttt{E}(y,y'), \texttt{One}(x,x), \texttt{One}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{E}(x,x'), \texttt{\$}(y,y'), \texttt{One}(x,x), \texttt{One}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{\$}(x,x'), \texttt{E}(y,y'), \texttt{One}(x,x), \texttt{One}(y,y).$$
$$equal_0(x,y) \leftarrow \texttt{\$}(x,x'), \texttt{\$}(y,y'), \texttt{One}(x,x), \texttt{One}(y,y).$$

We are only interested in the behavior of $equal_i$ over expansions of Ω. 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 Ω are basically directed paths whose edges are labeled by E or \$, and whose variables (or nodes) are labeled with symbols in

$$\{\texttt{Start, IsAddress, IsSymbol, Zero, One, Carry0, Carry1}\} \cup \{\texttt{Q}_a \mid \text{ for each symbol } a\}$$

It is easy to see that in such a models, $equal_i(x,y)$ 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 Ω, we use ideas from [19]. First, we need to verify that the first block of the expansion corresponds an address of length 2^n followed by a symbol. We do this by putting in Ω' the rules

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

The first rule finds expansions where one of the first 2^n edges is a $-labeled edge. The second rule finds expansions where one of the first 2^n variables is a *symbol* variable, that is, a variable labeled with `IsSymbol`. The last rule detects expansions where the (2^n+1)-th variable is an *address* variable, that is, a variable labeled with `IsAddress`. We also add rules

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

Similarly, the first and second 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 third and fourth rules 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 $0,\dots,0$ and two adjacent addresses are successive. For example, a possible error is that the first address is not $0,\dots,0$. Such an error can be found by the following rule

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

Another possible error is when the i-th carry bit is 0, but the $(i+1)$-th carry and address bit are 1. This can be detected by the rule

$$Ans() \leftarrow \texttt{IsAddress}(x,x), \texttt{E}(x,y), \texttt{Carry0}(x,x), \texttt{One}(y,y), \texttt{Carry1}(y,y).$$

A more interesting case is when the i-th carry and address bit are the same but the i-th address bit in the next address is 1, instead of 0. This is detected by the following rules

$$Ans() \leftarrow \texttt{IsAddress}(x,x), \texttt{Zero}(x,x), \texttt{Carry0}(x,x), dist_n(x,y), \texttt{E}(y,z), \texttt{One}(z,z).$$
$$Ans() \leftarrow \texttt{IsAddress}(x,x), \texttt{One}(x,x), \texttt{Carry1}(x,x), dist_n(x,y), \texttt{E}(y,z), \texttt{One}(z,z).$$

Note that corresponding address variables in successive addresses are exactly at distance $2^n + 1$. The rest of the cases can be easily detected by similar rules.

We now have to ensure that every sequence of addresses starting with $0,\dots,0$ describe a single configuration; that is, configurations change exactly when the address is $1,\dots,1$. Thus, there are two types of error here: (1) a configuration changes when the address is not $1,\dots,1$, or (2) a configuration does not change when the address is $1,\dots,1$. 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

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

Now we can detect errors of type (2) as follows

$$Ans() \leftarrow AllOnes_n(x,y), \texttt{E}(y,z).$$

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, $SameConf(x,y)$ will be true exactly when x and y are variables in the same configuration. Similary, $NextConf(x,y)$ holds exactly when x and y are in adjacent configurations. These predicates can be defined as follows

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

Now we can verify that the first configuration is actually the initial configuration. Suppose that $\perp$ correspond 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

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

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 Ω' of the form

$$\begin{aligned}
Ans() \leftarrow\ &\texttt{IsSymbol}(x_1,x_1), \texttt{Q}_a(x_1,x_1), \texttt{E}(x_1,z_2),\\
&dist_n(z_2,x_2), \texttt{IsSymbol}(x_2,x_2), \texttt{Q}_b(x_2,x_2), \texttt{E}(x_2,z_3),\\
&dist_n(z_3,x_3), \texttt{IsSymbol}(x_3,x_3), \texttt{Q}_c(x_3,x_3),\\
&dist_n(z,x), \texttt{IsSymbol}(x,x), \texttt{Q}_d(x,x),\\
&NextConf(z_2,z), equal_n(z_2,z).
\end{aligned}$$

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 this set of 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 Ω and Ω' can be carried out in polynomial time in n and the size of M.