# 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.

**1998 ACM Subject Classification** H.2.3 Languages – *Query languages.*

**Keywords and phrases** Graph databases, conjunctive regular path queries, regular queries, containment.

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2015.1

## 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 differ from relational databases in that the topology of the data is as important as the data itself. Thus, typical graph database queries are *navigational*, asking whether some nodes are connected by paths satisfying some specific properties. The most basic query language for graph databases is that of *regular-path queries* (RPQs) [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].

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 UC2RPQs and UCN2RPQs and can express many interesting properties that UCN2RPQs can not (see e.g. [11, 38, 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 $\Omega$ and $\Omega'$, whether the answer of $\Omega$ is contained in the answer of $\Omega'$, 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 $\Gamma$ 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 suitable representation of the *partial mappings* of $\Gamma$ 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. Thus, regular queries seems to be the most expressive fragment of first-order logic with transitive closure that is known to have an elementary containment problem.

**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. Due to space limitations, we only present the main ideas and intuitions behind our proofs. Complete proofs will be given in the full version of the paper.

## 2 Preliminaries

**Graph databases.** Let $\Sigma$ be a finite alphabet. A *graph database* $\mathcal{G}$ over $\Sigma$ 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 $\Sigma$ with the *inverse* of each symbol. The *completion* $\mathcal{G}^{\pm}$ of a graph database $\mathcal{G}$ over $\Sigma$, 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 $\sigma$ 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 $\sigma$, 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 $\theta$ in $\mathcal{D}$. As usual, if $\theta$ is a *boolean* CQ, that is, it has no free variables, we identify the answer $\mathsf{false}$ with the empty relation, and $\mathsf{true}$ with the relation containing the 0-ary tuple.

We want to use CQs for querying graph databases over a finite alphabet $\Sigma$. In order to do this, given an alphabet $\Sigma$, 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 $\Sigma$ 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 $\Sigma$ is simply a conjunctive query over $\sigma(\Sigma^{\pm})$. The answer $\theta(\mathcal{G})$ of a CQ $\theta$ over $\mathcal{G}$ is $\theta(\mathcal{D}(\mathcal{G}^{\pm}))$. A union of CQs (UCQ) $\Theta$ over $\Sigma$ is a set $\{\theta_1(\bar{x}), \ldots, \theta_k(\bar{x})\}$ of CQs over $\Sigma$ 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 $\varphi$ 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 $\Sigma$ 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 $\Sigma$ 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 $\Sigma$, for each $1 \leq i \leq m$. The *answer* $\gamma(\mathcal{G})$ of $\gamma$ over a graph database $\mathcal{G}$ is defined in the obvious way. A union of C2RPQs (UC2RPQ) $\Gamma$ over $\Sigma$ is a finite set $\{\gamma_1(\bar{x}), \ldots, \gamma_k(\bar{x})\}$ of C2RPQs over $\Sigma$ 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* $\Pi$ 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($\Pi$) and EDB($\Pi$) 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 $\Pi$ and $\mathcal{D}$ a database with schema EDB($\Pi$). 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 $\Pi$. Let $P^\infty_\Pi(\mathcal{D})$ be $\bigcup_{i\geq 0} P^i_\Pi(\mathcal{D})$. Then, the *answer* $\Pi(\mathcal{D})$ of $\Pi$ over $\mathcal{D}$ is $Ans^\infty_\Pi(\mathcal{D})$.

A predicate $P$ *depends* on a predicate $Q$ in a Datalog program $\Pi$, if $Q$ occurs in the body of a rule $\rho$ of $\Pi$ and $P$ is the predicate at the head of $\rho$. The *dependence graph* of $\Pi$ is a directed graph whose nodes are the predicates of $\Pi$ and whose edges capture the dependence relation: there is an edge from $Q$ to $P$ if $P$ depends on $Q$. A program $\Pi$ 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 $\Sigma$ is a (nonrecursive) Datalog program $\Pi$ such that EDB($\Pi$) $= \sigma(\Sigma^\pm)$. The *answer* $\Pi(\mathcal{G})$ of a (nonrecursive) Datalog program $\Pi$ over a graph database $\mathcal{G}$ over $\Sigma$ is $\Pi(\mathcal{D}(\mathcal{G}^\pm))$.

▶ Remark. Typically, when we analyze a problem involving 2RPQs over $\Sigma$, 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* $\Omega$ over a finite alphabet $\Sigma$ is a finite set of Datalog and transitive closure rules such that:

1. The extensional predicates of $\Omega$ belong 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 $\Omega$ is acyclic.

The semantics of regular queries is based on the semantics of Datalog. We define a translation function $\lambda$ that transforms a Datalog rule or a transitive closure rule into a set of Datalog rules. If $\rho$ is a Datalog rule then $\lambda(\rho) = \{\rho\}$. When $\rho$ 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 $\Omega = \{\rho_1, \ldots, \rho_k\}$ into a Datalog program $\lambda(\Omega) = \lambda(\rho_1) \cup \cdots \cup \lambda(\rho_k)$. Then, the *answer* $\Omega(\mathcal{G})$ of a regular query $\Omega$ 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 $\Omega$, 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 $\Omega$ and $\Omega'$ over alphabet $\Sigma$, we say that $\Omega$ is *contained* in $\Omega'$ if $\Omega(\mathcal{G}) \subseteq \Omega'(\mathcal{G})$, for each graph database $\mathcal{G}$ over $\Sigma$. The *containment problem* for regular queries asks, given two RQs $\Omega$ and $\Omega'$, whether $\Omega$ is contained in $\Omega'$. We devote the rest of this paper to 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 $\Gamma$ of extended C2RPQ rules, we define *intensional predicates* and the *dependence graph* in the obvious way; now the 2RPQs mentioned in $\Gamma$ play the role of extensional predicates.

▶ **Definition 5. (Nested UC2RPQ)** A *nested UC2RPQ* $\Gamma$ over $\Sigma$ is a finite set of extended C2RPQ rules such that:

1. All 2RPQs mentioned in $\Gamma$ are defined over $\Sigma$.
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 $\Gamma$ is acyclic.
5. For each intensional predicate $S$, there is a unique occurrence of $S$ over rule bodies of $\Gamma$.

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 $\lambda$ that maps an extended C2RPQ rule to a set of Datalog or transitive closure rules. Let $\rho$ 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 $S(x_1,\ldots,x_n) \leftarrow P_1(y_1,y_1'),\ldots,P_m(y_m,y_m')$, where $P_1,\ldots,P_m$ 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 $\Gamma = \{\rho_1,\ldots,\rho_k\}$ to a regular query $\lambda(\Gamma) = \lambda(\rho_1) \cup \cdots \cup \lambda(\rho_k)$. 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 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 *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 suitable representation of the *partial mappings* from a nested UC2RPQ to an expansion of a 2RPQ. This representation is robust against nesting, in the sense that does not involve a non-elementary blow up in size.

### 4.1.1 Reduction to Containment of 2RPQs in nested UC2RPQs

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

Let $\Gamma$ be a nested UC2RPQ over alphabet $\Sigma$ and let $S$ be an intensional predicate. We denote by $\mathsf{rules}(S)$ the set of rules in $\Gamma$ such that $S$ occurs in the head of the rule. An *expansion* $\varphi$ of $S$ is a CQ with equality over $\Sigma$ 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 $\phi_i$ and $\phi_j$ 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 $\Gamma$ and $\Gamma'$ be two nested UC2RPQs. Then, $\Gamma$ is contained in $\Gamma'$ if and only if, for each expansion $\varphi$ of $\Gamma$, there exists an expansion $\varphi'$ of $\Gamma'$ 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 $\theta$ and $\theta'$ are two boolean CQs over $\Sigma$, then a *containment mapping* $\mu$ from $\theta'$ to $\theta$ is a mapping from the variables of $\theta'$ to the variables of $\theta$ such that, for each atom $r(y, y')$ in $\theta'$, with $r \in \Sigma^{\pm}$, either $r(\mu(y), \mu(y'))$ is in $\theta$ or $r^-(\mu(y'), \mu(y))$ is in $\theta$.

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

1. Expansions of $\Gamma$ can be "serialized" and represented by *serialized expansions*, which are strings over a larger alphabet $\Delta$. 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 $\Gamma$. This technique has been already used before [14, 15].
2. Now we need to serialize $\Gamma'$. Proposition 8 basically tell us that $\Gamma$ is contained in $\Gamma'$ iff $\Gamma'$ can be "mapped" to each expansion of $\Gamma$. We have replaced $\Gamma$ by $\tilde{E}$. Thus, expansions of $\Gamma$ are replaced by serialized expansions. By modifying the 2RPQs mentioned in $\Gamma'$, we construct a nested UC2RPQ $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to a serialized expansion $W$ of $\Gamma$ iff $\Gamma'$ can be mapped to the expansion of $\Gamma$ represented by $W$. This is a new technique and constitutes the crux of the reduction.

### Serialization of $\Gamma$

We can represent each expansion $\varphi$ of $\Gamma$, by a serialized expansion, that is, a string over a suitable alphabet $\Delta$. Suppose first that $\Gamma$ 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 $\varphi$ 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 $\Delta$ contains $\Sigma^{\pm}$, a separator \$, and the variable set of $\Gamma$. Assume now that $\Gamma$ 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 $\varphi$ is the expansion of $\Gamma$ 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 $\varphi$ 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 $\Delta$ contains additionally the separator $\star$ that represents fresh intermediate variables that appear when we expand a transitive closure atom.

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

### Serialization of $\Gamma'$

We replaced $\Gamma$ by $\tilde{E}$. Thus we replaced expansions of $\Gamma$ 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 $\varphi$ of $\Gamma$. Our goal is to construct $\tilde{\Gamma}$ such that $\tilde{\Gamma}$ can be mapped to $\theta^W$ iff $\Gamma'$ can be mapped to $\mathtt{neq}(\varphi)$. To construct $\tilde{\Gamma}$, we modify $\Gamma'$ in order to translate mappings from $\Gamma'$ to $\mathtt{neq}(\varphi)$, into mappings from $\tilde{\Gamma}$ to $\theta^W$ (and vice versa). Next we explain the main difficulties in the construction of $\tilde{\Gamma}$.

Let $W$ be a serialized expansion representing an expansion $\varphi$ of $\Gamma$. Note that some symbols in $W$, as the variable symbols or the $\star$ symbol, represent a particular variable in $\varphi$. If the $i$-th symbol of $W$ represents a variable in $\varphi$, we denote this variable by $\mathsf{var}(i)$. Note also that equality atoms in $\varphi$ define equivalence classes over the variables of $\varphi$. 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 $\mathsf{neq}(\varphi)$. Then, in order to simulate mappings from $\Gamma'$ 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 $\alpha$ can be folded into $W$ if $\alpha$ 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. 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, suppose $\Gamma$ is a query over alphabet $\{a, b\}$ defined by the rules

$$Q(t,s) \leftarrow b(t,s) \qquad\qquad Ans() \leftarrow a(x,y), a+\varepsilon(x,z), Q^+(w,z), aa(a+b)(w,u)$$

and consider the expansion $\varphi$ of $\Gamma$ obtained by choosing the strings $\varepsilon$ and $aab$ for the second and last atom, respectively, and no intermediate variables for $Q^+(w,z)$. This expansion is represented by the following serialized expansion $W$

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

Note that $x$ and $z$ are equivalent in $\varphi$, since it contains the atom $x = z$ (as we chose the string $\varepsilon$). Then any occurrence of $x$ is equivalent to any occurrence of $z$ in $W$. In particular, 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 $x\Delta^*xz\Delta^*z$. In general, we have to consider expressions of the form $x\Delta^*xz_1\Delta^*z_1z_2\Delta^*\cdots z_kz\Delta^*z$, for a sequence of variables $z_1, \ldots, z_k$. Note also that the first occurrence of $w$ is equivalent to the occurrence of $t$ in $W$. This is because $t$ is the first free variable of the expansion $b(t,s)$, which is renamed to $w$ in $\varphi$ by definition. 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 $\Delta$, thus the actual definition of $\Delta$ is slightly more involved that the one presented here.

Now we are ready to serialize the nested UC2RPQ $\Gamma'$. 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 $\alpha_0w_1\alpha_1w_2\alpha_2\cdots\alpha_{p-1}w_p\alpha_p$, where, for each $0 \leq i \leq p$, the string $\alpha_i$ is either $\varepsilon$ 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 $\Delta$.

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

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

The idea behind Proposition 10 is as follows. Suppose $\Gamma$ is contained in $\Gamma'$. 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 $\varphi$. We know that there is a containment mapping $\mu$ from $\mathsf{neq}(\varphi')$ to $\mathsf{neq}(\varphi)$, for some expansion $\varphi'$ of $\Gamma'$. We have to find an expansion $\psi$ of $\tilde{\Gamma}$ and a containment mapping $\nu$ from $\mathsf{neq}(\psi)$ to $\theta^W$. 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 $\mu$ to $t$ as $t$ is also an external variable of $\mathsf{neq}(\varphi')$). Let $y$ be a variable in $\varphi$ 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 $\nu$ 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 $\nu$ 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 $\alpha$ 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 similar.

### 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 $\Gamma$ 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 $\theta$ of $E$ such that there is a mapping from $\Gamma$ to $\theta$.

To show that checking containment of two UC2RPQs $\Gamma'$ and $\Gamma$ is in EXPSPACE, Calvanese et al. exploit *two-way* NFAs (2NFAs) in order to construct an automaton $\mathcal{A}_\Gamma$ that accepts expansions $\theta$ of $\Gamma'$ such that $\Gamma$ can be mapped to $\theta$ [14]. Moreover, they annotate the expansions with variables of $\Gamma$, which indicate the potential mapping of $\Gamma$ to the expansion.

This is possible because the number of possible annotations is bounded, as the number of variables in the queries is bounded. But when $\Gamma$ is a nested UC2RPQ the number of variables involved in a mapping from $\Gamma$ to $\theta$ is not bounded anymore, and thus it is by no means obvious how to extend the techniques of [14] in this case. Instead, we follow a different approach, and construct $\mathcal{A}_\Gamma$ directly as a 1NFA. The automaton $\mathcal{A}_\Gamma$ scans the input $\theta$ from left to right and in each step, it guesses a "partial mapping" from $\Gamma$ to $\theta$. This is formalized with the notion of *cut*, which we define next.

We use the fact that the expansions for a 2RPQ $E$ over $\Sigma$ 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 $\Gamma$, we need only to look at those expansions of $\Gamma$ that can be *flattened* into a linear CQ, i.e., those that can actually be mapped to some linear CQ. Formally, given an expansion $\varphi$ of $\Gamma$, a *linearization* of $\varphi$ is a linear CQ $\theta$ such that there is a containment mapping from $\mathsf{neq}(\varphi)$ to $\theta$. Furthermore, the set of linearizations of $\Gamma$ is the union of all linearizations of all the expansions of $\Gamma$. As we mentioned before, the idea of this proof is to show that the set of linearizations of a nested UC2RPQ $\Gamma$ 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 $\Gamma$ be a nested UC2RPQ of depth 0, defined by the rules

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

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

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 $\Gamma$ with the answer predicate in the head follow form (1) above (note that now some of the $\Gamma^i_j$s might be predicates $P^+$ instead of RPQs). Then cuts for $\Gamma$ are again $\ell$-tuples $(C^1, \ldots, C^\ell)$, the only thing that changes is the definition of $\mathcal{S}^i = (s^i_1, \ldots, s^i_{m_i})$, for the cases when $\Gamma^i_j$ 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 $\mathrm{Prev}^i$ is empty, and *final cuts* are those in which at least one of the $\mathrm{Prev}^i$s is equal to $\mathrm{V}ars(\Gamma, i)$. A cut *marks* a variable $x$ if in all $C^i$s that are not $\perp$ we have that $x$ belongs to $\mathrm{Same}^i$.

Let us now give some intuition on the notion of cuts. Suppose $\Gamma$ is a C2RPQ and $\mu$ is a mapping from $\Gamma$ to a linearization $\theta$ of $\Gamma$. If we look at position $k$ in $\theta$, then the partial mapping of $\mu$ until this position can be represented by a cut $(\mathrm{Prev}, \mathrm{Same}, \mathcal{S})$: Prev are the variables that are mapped to positions smaller or equal 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 $y, y' \in (\mathrm{Prev} \cup \mathrm{Same})$ are already satisfied when looking up to $k$. The only information that is missing is that of the 2RPQs that are "cut" by $(\mathrm{Prev} \cup \mathrm{Same})$, that is, the 2RPQs of the form $F(y, y')$ such that $y \in (\mathrm{Prev} \cup \mathrm{Same})$ and

$y' \notin (\text{Prev} \cup \text{Same})$ (or vice versa). Suppose that $\mu$ expands $F$ in a linearization given by the 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 $\mu$ 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 $s_j$. 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 $\mu$. Since we are dealing with UC2RPQs we also need to account for the case when a certain disjunct of $\Gamma$ 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 $\Gamma$ (here we use the fact that $\Gamma$ 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 $\Gamma$. 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 $\Gamma$ and positions of a word $w$ over $\Sigma^{\pm}$, i.e., over pairs from $Cuts(\Gamma) \times \{1, \dots, p+1\}$.

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 $\theta = g(x,x'), g(x',z), g(y',z), g(y,y')$ is a linearization of $\Gamma$. The string associated to $\theta$ 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 $(\{x\},\{x\},\mathcal{S}_1)$, where $\mathcal{S}_1$ only needs to store the state where the automaton for query $g^+$ is, after having read a $g$. We then advance to cut $(\{x,z\},\{z\},\mathcal{S}_2)$ in position 3, $(\{x,z\},\{\},\mathcal{S}_3)$ in position 4, where now $\mathcal{S}_3$ again needs to store a state of the automaton for $g^+$, and finally to cut $(\{x,z,y\},\{y\},\mathcal{S}_4)$ 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 $\Gamma$ 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 pairs $(C,p)$ and $(C',p')$ define an *accepting run* for $\Gamma$ 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 $w = r_1, \dots, r_p$ a string over $\Sigma$. There are pairs $(C,i)$ and $(C',i')$ over $Cuts(\Gamma) \times \{1, \dots, p+1\}$ that define an* accepting run *for $\Gamma$ over $w$ if and only if there is an expansion $\varphi$ of $\Gamma$ and a containment mapping from* $\mathsf{neq}(\varphi)$ *to the linear CQ $Q_w = r_1(z_1,z_2), \dots r_p(z_p,z_{p+1})$ that maps $x$ and $y$ to variables $z_i$ and $z_{i'}$.*

**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 $\Gamma$. However, after a careful analysis one realizes that doing this in a straightforward way results in a much more expensive algorithm, so a little bit of extra work has to be done to avoid additional exponential blowups in our algorithm. In a nutshell, the idea is to extend the alphabet with information about cuts. Given $\Gamma$ and $w$, we add to each symbol of $w$ the information about which cuts are reachable from configurations of the form $(C,p)$ in $T_{(\Gamma,w)}$, where $1 \leq p \leq |w|+1$.

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 $\Gamma$ 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, $\Sigma(\Gamma)$ contains a subset of $Cuts(\Gamma)$ for each cut in $Cuts(\Gamma)$ plus two additional subsets.
- Otherwise assume that $\Gamma$ is of form (1), and that $\mathcal{P}$ is the set of predicates occurring in the rules of $\Gamma$ 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 $\Gamma$.

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

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 $\Gamma$ 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 $\Gamma$. 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. $\Gamma$. We show:

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

We can finally proceed to build the desired 1NFA $A_\Gamma$ that gives the strings corresponding to the linearizations of a nested UC2RPQ $\Gamma$. This 1NFA needs to simulate the system $T_{(\Gamma,w)}$, from an initial cut of a nested UC2RPQ $\Gamma$ 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 $\Gamma$, 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 $\Gamma$ over $\Sigma$, one can construct, in exponential time, a 1NFA $A_\Gamma$ over alphabet $\Sigma(\Gamma) \times \Sigma^{\pm}$ such that $A_\Gamma$ accepts a string $w = r_1, \ldots, r_p$ with valid annotations w.r.t. $\Gamma$ if and only if there are pairs $(C,i)$ and $(C',i')$ over $Cuts(\Gamma) \times \{1,\ldots,p+1\}$ that define an accepting run for $\Gamma$ over $w$.*

Together with Lemma 12, this Lemma implies that $A_\Gamma$ characterizes all those strings $w$ that have *valid annotations* w.r.t. $\Gamma$ such that $\tau(w)$ is accepted by $\Gamma$.

**Main Proof.** To decide containment of a 2RPQ $E$ in a nested UC2RPQ $\Gamma$, 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. $\Gamma$.
- 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 $\Gamma$. 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 $\Gamma$, i.e., if $E$ is not contained in $\Gamma$.

Even though some of these automata can be of doubly exponential size w.r.t. $E$ and $\Gamma$, 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 $\Omega$ be a regular query. Let $h, w$ and $d$ be the height, the width and the depth of $\Omega$, respectively. Then, $\Omega$ is equivalent to a nested UC2RPQ $\Gamma$ 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 $\Gamma$ is double-exponential in the size of $\Omega$.*

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 $\Omega$ may result in a nested UC2RPQ $\Gamma$ that is of double-exponential size with respect to $\Omega$, and thus the number of cuts in $\Gamma$ might be of triple-exponential size with respect to $\Omega$.

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 $\Gamma$, but not on the height. The idea is to define cuts of nested UC2RPQs not as a tuple $(C^1, \dots, C^\ell)$ for each of the $\ell$ 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 $\Gamma$. Using this modified construction and Proposition 16 we can then show that the number of cuts is again double-exponential in the size of $\Omega$, which immediately leads to a 2EXPSPACE algorithm, following the ideas of the proof of Theorem 7.

In summary, the 2EXPSPACE algorithm for containment of regular queries works as follows (for simplicity we assume that the inputs are boolean queries).

1. Given regular queries $\Omega$ and $\Omega'$, we unfold these queries to construct nested UC2RPQs $\Gamma$ and $\Gamma'$. By Proposition 16, we have that (i) $|\Gamma'|$ is doubly-exponential in $|\Omega'|$, (ii) the width of $\Gamma'$ is single-exponential in $|\Omega'|$, and (iii) the depth of $\Gamma'$ is polynomial in $|\Omega'|$. Similarly for $\Gamma$ and $\Omega$.

2. We construct from $\Gamma$ and $\Gamma'$ the 2RPQ $\tilde{E}$ and the nested UC2RPQ $\tilde{\Gamma}$ defined in the reduction from Section 4.1.1. This is a polynomial-time reduction, thus basically all the bounds from (1) still hold: $|\tilde{E}|$ is doubly-exponential in $|\Omega|$; $|\tilde{\Gamma}|$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$ (note that $\tilde{\Gamma}$ now depends on $\Omega$ too); the width of $\tilde{\Gamma}$ is single-exponential in $|\Omega'|$; the depth of $\tilde{\Gamma}$ is polynomial in $|\Omega'|$.
3. With the refined definition of cut explained above, we obtain that, for a nested UC2RPQ $\Gamma''$, the number of cuts is roughly $|\Gamma''|^{w^d}$, where $w$ and $d$ is the width and depth of $\Gamma''$, respectively. Thus the number of cuts of our query $\tilde{\Gamma}$ is doubly-exponential in $|\Omega|$ and $|\Omega'|$. It follows that the size of the automata $\mathcal{A}_{\tilde{E}}$, $\mathcal{A}_{\Sigma(\tilde{\Gamma})}$ and $(\mathcal{A}_{\tilde{\Gamma}})^C$ from Section 4.1.2 is triple-exponential in $|\Omega|$ and $|\Omega'|$. To decide whether $\Omega \subseteq \Omega'$, we check the intersection of these automata for emptiness, which can be done in 2EXPSPACE.

By combining techniques from [19, 14], we can show a matching lower bound for containment of regular queries. This conclude the proof of Theorem 4.

## 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 regular queries 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 regular query 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 regular queries 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 regular queries, we conjecture that this problem is elementary.

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

## 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.