A Theory of Regular Queries

Moshe Y. Vardi
Rice University
vardi@cs.rice.edu

ABSTRACT

A major theme in relational database theory is navigating the tradeoff between expressiveness and tractability for query languages, where the query-containment problem is considered a benchmark of tractability. The query class UCQ, consisting of unions of conjunctive queries, is a fragment of first-order logic that has a decidable query containment problem, but its expressiveness is limited. Extending UCQ with recursion yields Datalog, an expressive query language that has been studied extensively and has recently become popular in application areas such as declarative networking. Unfortunately, Datalog has an undecidable query containment problem. Identifying a fragment of Datalog that is expressive enough for applications but has a decidable query-containment problem has been an open problem for several years.

In the area of graph databases, there has been a similar search for a query language that combines expressiveness and tractability. Because of the need to navigate along graph paths of unspecified length, transitive closure has been considered a fundamental operation. Query classes of increasing complexity — using the operations of disjunction, conjunction, projection, and transitive closure — have been studied, but the classes lacked natural closure properties. The class RQ of regular queries has emerged only recently as a natural query class that is closed under all of its operations and has a decidable query-containment problem.

RQ turned out to be a fragment of Datalog where recursion can be used only to express transitive closure. Furthermore, it turns out that applying this idea to Datalog, that is, restricting recursion to the expression of transitive closure, does yield the long-sought goal — an expressive fragment of Datalog with a decidable query-optimization problem.

1. INTRODUCTION

The development of declarative database-query languages is one of the great accomplishments of computer science. Following the introduction of relational databases by Codd [22], he proposed using first-order logic as a declarative database query language [23]. This led to the development of both SEQUEL [16] and QUEL [54], as practical relational query languages realizing Codd’s idea, ultimately giving in 1986 rise to the SQL standard [27], which has continued to evolve over the past 30 years.

Starting in the late 1970s, Codd’s definition of first-order logic as “expressively complete” came under serious criticism [4], and various proposals emerged on how to extend its expressive power [4, 6, 17], whose essence was to extend first-order logic with recursion, which was added to SQL in 1999 by way of common table expressions [29]. On the research side, the most popular relational language embodying recursion is Datalog [15], described in more details below.

From the very beginning of research on relational query languages it became clear that query optimization is a central problem due to the declarative nature of these languages [23, 32]. Fundamentally, query optimization requires us to transform a query Q to an equivalent query Q’ that is easier to evaluate. Query equivalence can be reduced to query containment — Q is equivalent to Q’ if Q is contained in Q’ and Q’ is contained in Q. Thus, query containment is a key database-theoretic problem. In fact, it has found uses in many other database contexts, including query reuse [47], query reformulation [31], data integration [33, 36], cooperative query answering [44], and more.

Unfortunately, query containment is essentially logical implication; thus, it is undecidable for first-order logic, and, consequently, for SQL [2]. A major fragment of SQL for which query containment is decidable is the class UCQ, union of conjunctive queries, which is the class of queries that be be defined using the relational operator Select, Project, Join, and Union [18, 50]. In fact, UCQ is also the largest class of relational queries for which we have a robust theory and practice of query optimization [19].

Datalog is essentially the closure of UCQ under recursion. The main appeal of Datalog comes from its syntactical simplicity; a Datalog query consists of a set of rules, where each rule expresses a conjunctive query (Select-Project-Join query) [15]. Due to this syntactical simplicity, Datalog has recently found applications outside the database area, for example, in declarative networking [37]. Unfortunately, the presence of recursion in Datalog is sufficient to make the containment problem undecidable [52]. The only fragment of Datalog that is known to have a decidable containment problem is the class of monadic Datalog, in which all recursive rules return monadic relations (i.e., sets) [25]. This fragment, however, is too weak for practical applications.
such as declarative networking, due to its inability to express network connectivity properties.

In a different development, information systems nowadays are required to deal with more complex data with respect to traditional relational data. For example, web data, social networks, or biological data are better modeled by resorting to more flexible structuring mechanisms than those provided by relational systems [10]. This development led to the emergence of the graph-database model as a data model that is less expressive but is more flexible than the classical relational model. The graph-data model conceives a database essentially as a finite directed labeled graph whose nodes represent objects, and whose edges represent relationships between objects [1], that is, an edge-labeled graph. (It is convenient to restrict attention, without loss of generality, to edge-labeled graphs.) Over the past 20 years, graph databases have been a major focus of research and practical interest [41, 49].

As in relational databases, a major research focus in graph databases is identifying a core query language that has a decidable query-containment problem. In the same way that UCQ forms the decidable core of of both SQL and Datalog, regular-path queries (RPQs) are considered the basic querying mechanisms for graph databases [1, 3, 10]. Query languages for graph databases must indeed be equipped with flexible mechanisms for navigating the graph representing the data. This includes the ability to follow a sequence of edges of the graph whose length is not specified a priori, something which is directly provided by RPQs via a limited form of recursion in the form of transitive closure. Thus, RPQs are simply regular expressions over the edge alphabet of the graph database.

The theory of regular languages is one of the most thoroughly studied and best understood theories in theoretical computer science [30]. Regular languages have robust definability properties. That is, different means of defining regular languages, e.g., regular expressions vs. automata, have the same expressive power. They also have robust closure properties, that is, they are closed under many set-theoretic and algebraic operations. In particular, the containment problem for regular expression can be shown to be decidable in polynomial space (in fact, PSPACE-complete) using standard automata-theoretic constructions [30]. Can this results from regular-language theory be applied to the containment problem for RPQs?

RPQs allow for navigating the edges of a graph database only in the forward direction. It is obviously of interest, however, to be able to navigate edges in both forward and backward directions, as, e.g., supported by the predecessor axis of XPath [9, 21]. RPQs extended with the ability of navigating database edges backward are called two-way regular-path queries (2RPQs) [12]. It turns out that while regular-expression containment techniques can be applied to RPQ containment, these techniques cannot easily be applied to 2RPQ containment, because the difference between “inverse letters” in words and “inverse edges” in graphs. That is, the theories of regular expressions over words and regular expression over graphs diverge [12]. Nevertheless, containment of 2RPQs can still be shown to be PSPACE-complete, using automata-theoretic techniques [12].

The divergence between the the theories of regular expressions over words and regular expressions over graphs expands when we consider the conjunction operation. Over
FROM CONJUNCTIVE QUERIES TO DATALOG

2.1 Union of Conjunctive Queries

A conjunctive query is a positive existential conjunctive first-order formula, i.e., the only propositional connective allowed is ∧ and the only quantifier allowed is ∃. Without loss of generality, we can assume that conjunctive queries are given as formulas \( \theta(x_1, \ldots, x_k) \) of the form \((\exists y_1, \ldots, y_m)(a_1 \land \ldots \land a_n)\) with free variables among \(x_1, \ldots, x_k\), where the \(a_i\)'s are atomic formulas of the form \(p(z_1, \ldots, z_l)\) over the variables \(x_1, \ldots, x_k, y_1, \ldots, y_m\). For example, the conjunctive query \((\exists y)(E(x,y) \land E(y,z))\) is satisfied by all pairs \((x,z)\) such that there is a path of length 2 between \(x\) and \(z\). The free variables are also called distinguished variables. A union of conjunctive queries is a disjunction

\[
\bigvee_{i=1}^k \theta_i(x_1, \ldots, x_k)
\]

of conjunctive queries. Every positive first-order formula that has only existential quantifiers can be written (with a possible blow-up in size) as a union of conjunctive queries. The class of such queries is denoted UCQ. This class is essentially the class of relational algebraic queries composed of the Select, Project, Join, and Union operators, which cover a major fraction of relational queries used in practice [2].

A union of conjunctive queries \(\Theta(x_1, \ldots, x_k)\) with distinguished variables \(x_1, \ldots, x_k\) can be applied to a database \(D\). The result

\[
\Theta(D) = \{(a_1, \ldots, a_k) | D \models \Theta(a_1, \ldots, a_k)\}
\]

is the set of \(k\)-ary tuples representing assignments to the distinguished variables that satisfy \(\Theta\) in \(D\).

2.2 Datalog

A Datalog program consists of a set of Horn rules. A Horn rule consists of a single atom in the head of the rule and a conjunction of atoms in the body, where an atom is a formula of the form \(p(z_1, \ldots, z_l)\) where \(p\) is a predicate symbol and \(z_1, \ldots, z_l\) are variables. Note that every such rule can be viewed as a conjunctive query, under the convention that variables that appear in the body but not in the head are quantified existentially. The predicates that occur in head of rules are called intensional (IDB) predicates. The rest of the predicates are called extensional (EDB) predicates. Let \(\Pi\) be a Datalog program. Let \(P_i^\Pi(D)\) be the collection of facts about an IDB predicate \(P\) that can be deduced from a database \(D\) by at most \(i\) applications of the rules in \(\Pi\) and let \(P_i^\Pi(D)\) be the collection of facts about \(P\) that can be deduced from \(D\) by an arbitrary number of applications of the rules in \(\Pi\), that is,

\[
P_i^\Pi(D) = \bigcup_{i \geq 0} P_i^\Pi(D).
\]

A Datalog query \(Q\) is a Datalog program \(\Pi\) with one IDB predicate \(P\) designated as the goal predicate. Then we have \(Q(D) = P_\infty^\Pi(D)\). For detailed semantics, see [15].
cates, that is, they define sets of values. Monadic Data-
log is more expressive than nonrecursive Datalog, while still 
having a decidable query-containment problem [25]. Yet 
Monadic Datalog is not expressive enough for applications.
In Monadic Datalog we can express reachability properties,
i.e., we can capture the set $Q$ of elements that have a path 
to nodes in a given set $P$, using the program:

$$
Q(X) : = E(X,Y), P(Y)
$$

$$
Q(X) : = E(X,Y), Q(Y)
$$

Yet we cannot express in Monadic Datalog the binary rela-
tion $E^+$, which is the transitive closure of a binary relation, 
$E$ [7]. That is, we cannot express the following Datalog 
program:

$$
E^+(X,Y) : = E(X,Y)
$$

$$
E^+(X,Z) : = E(X,Y), E^+(Y,Z)
$$

(Note that Monadic Datalog can have non-monadic goals; 
it is only the recursive predicates that are restricted to be 
monadic.) But it is exactly these kind of connectivity prop-
erties that are needed in applications such as declarative net-
working. Thus, Monadic Datalog does not provide a query 
language that is expressive enough, but with a decidable 
query-containment problem.

3. GRAPH DATABASES

3.1 Regular Path Queries

Following the usual approach in modeling graph data [1], 
we define a graph database $D$ as a finite directed graph whose 
edges are labeled by elements from a finite alphabet $\Sigma$. Each 
node represents an objects and an edge from object $x$ to 
object $y$ labeled by $r$, denoted $r(x,y)$, represents the fact 
that relation $r$ holds between $x$ and $y$. (We omit node labels, 
as these can be captured by edge labels.) For a given edge 
label $r$, let $r(D)$ denote the binary relation consisting of the 
pairs of nodes connected by an edge labeled by $r$.

Observe that a graph database can be seen as a (finite) re-
lation schema over the set $\Sigma$ of binary relational symbols. 
Thus, the edge alphabet $\Sigma$ can be viewed as the relational 
schema of the database. But while in relational databases 
schema design is a "heavy duty" process and schema changes 
are not undertaken lightly, the edge alphabet of a graph 
database is simply part of the data and can be changed sim-
ply by updating the database. This flexibility is one feature 
of the "noSQL" movement, cf. [40].

A regular-path query (RPQ) $Q$ over a graph database $D$ 
is expressed as a regular expression (or, equivalently, a finite-
state automaton) over the edge alphabet $\Sigma$. The answer 
$Q(D)$ to a RPQ $Q$ over a database $D$ is the set of pairs of objects 
connected in $D$ by a semipath that conforms to the regular language 
$L(Q)$. A semipath in $D$ from $x$ to $y$ (labeled with $p_1 \cdots p_n$) 
is a sequence of the form $(y_0,p_1,y_1,\ldots,y_{n-1},p_n,y_n)$, where 
$n \geq 0$, $y_0 = x$, $y_n = y$, and for each $y_{i-1},p_i,y_i$, we have 
that $p_i \in \Sigma^\pm$, and, if $p_i \neq \epsilon$ then $(y_{i-1},y_i) \in r(D)$, and 
if $p_i = \epsilon$ then $(y_{i-1},y_i) \in \epsilon(D))$. Intuitively, a semi-
path $(y_0,p_1,y_1,\ldots,y_{n-1},p_n,y_n)$ corresponds to a naviga-
tion of the database from $y_0$ to $y_n$, following edges forward or 
backward, according to the sequence of edge labels $p_1 \cdots p_n$. 
Note that the objects in a semipath are not necessarily dis-
tinct. We say that a semipath $(y_0,p_1,\ldots,p_n,y_n)$ conforms 
to a 2RPQ $Q$ if $p_1 \cdots p_n \in L(Q)$. Summing up, a pair $(x,y)$ 
of objects is in the answer $Q(D)$ if and only if, by starting 
from $x$, it is possible to reach $y$ by navigating in $D$ according 
to one of the words in $L(Q)$.

3.2 Query Containment

To solve query containment for 2RPQs, we first consider 
RPQs, where we do not allow inverse symbols. We charac-
terize query containment via a fundamental lemma [14].

**Lemma 1.** (Language-Theoretic Lemma 1): Let $Q_1, Q_2$ 
be RPQs. Then $Q_1 \subseteq Q_2$ iff $L(Q_1) \subseteq L(Q_2)$.

Since containment of regular expressions is known to be 
PSPACE-complete [42], it follows from Language-Theoretic 
Lemma 1 that containment of RPQs is PSPACE-complete. Before 
we try to extend this result to 2RPQs, it is instruc-
tive to recall the proof of the upper bound. The key is the obser-
vation that $L(E_1) \subseteq L(E_2)$ iff $L(E_1) - L(E_2) = \emptyset$. The 
algorithm for checking whether $L(E_1) \subseteq L(E_2)$ proceeds 
as follows, using classical automata-theoretic constructions 
[30]:

1. Construct nondeterministic finite-state automata (NFAs) 
$A_1, A_2$ such that $L(A_1) = L(E_1)$. This step involves a linear 
  blow-up.
2. Construct an NFA $\overline{A_2}$ such that $L(\overline{A_2}) = \Sigma^* - L(A_2)$. 
  This step involves an exponential blow-up, as comple-
  mentation requires an application of the subset con-
  struction.
3. Construct an NFA $A = A_1 \times \overline{A_2}$ such that $L(A) = 
  L(E_1) - L(E_2)$. This requires taking the product of 
  $A_1$ and $\overline{A_2}$, involving a quadratic blow-up.
4. Check if there is a path from start state to final state 
in $A$. This requires nondeterministic logarithmic space 
in the size of $A$.

A naive application of steps (3–4) would require exponential-
space. Instead, we construct $A$ on the fly, constructing states 
only as we search for a path from a start state to a final state 
in $A$. This can be done in polynomial space, establishing the 
upper bound (formally, we need to appeal to Savitch’s The-
orem [51] to eliminate the nondeterminism in step (4).)

Extending this result to 2RPQs encounters two difficul-
ties. The first difficulty is that an automata-theoretic ap-
proach would most likely involve two-way automata, due 
to the presence of inverse letters, but extending the re-
sult of [42] to two- 
way automata is not straightforward. While it is known that two-way automata can be reduced
to one-way automata, that reduction has an exponential cost [30], making a naive approach to containment exponentially harder. An even more fundamental difficulty is that Language-Theoretic Lemma 1 fails for 2RPQs.

Consider the 2RPQs $Q_1 = p$, and $Q_2 = pp^-p$. It is easy to see that $Q_1 \subseteq Q_2$, as every semipath $(x, p, y)$, which establishes that $(x, y) \in Q_1(B)$, corresponds to the semipath $(x, p, y, p^-, x, p, y)$, establishing that $(x, y) \in Q_2(B)$. At the same time, we clearly do not have $L(Q_1) \subseteq L(Q_2)$, since $p \not\in L(Q_2)$. Our first step in studying query containment for 2RPQs is revising the language-theoretic characterization, which requires the notion of folding. Let $u, v \in \Sigma^\pm$. We say that $v$ folds onto $u$, $v \sim u$, if $v$ can be “folded” on $u$, e.g., $abb^-bc \sim abc$. Formally, we say that $v = u_1 \cdots u_m$ folds onto $u = u_1 \cdots u_n$ if there is a sequence $i_0, \ldots, i_m$ of positive integers between 0 and $|u|$ such that

- $i_0 = 0$ and $i_n = n$, and
- for $0 \leq j < m$, either $i_{j+1} = i_j + 1$ and $v_{j+1} = u_{i_j+1}$, or $i_{j+1} = i_j - 1$ and $v_{j+1} = u_{i_j+1}^\perp$.

(In particular, $v_0 = i_0$ and $v_m = u_n$.) For example, the sequence demonstrating that $abb^-bc \sim abc$ is 0, 1, 2, 1, 2, 3. Pictorially, $\Sigma \sim \Sigma \sim \Sigma \sim \Sigma \sim \Sigma \sim \Sigma$. Let $L$ be a language $\Sigma^\pm$. We define $\text{fold}(L) = \{u : v \sim u, v \in L\}$. We can now offer a language-theoretic characterization for containment of 2RPQs [14].

**Lemma 2.** (Language-Theoretic Lemma 2) Let $Q_1$ and $Q_2$ be 2RPQs. Then $Q_1 \subseteq Q_2$ iff $L(Q_1) \subseteq \text{fold}(L(Q_2))$.

We now show that if $A$ is an NFA, then $\text{fold}(L(A))$ can be represented by a “small” two-way nondeterministic finite-state automaton (2NFA). Recall that an NFA is a tuple $A = (\Sigma, S, S_0, \rho, F)$, where $\Sigma$ is a finite alphabet, $S$ is a finite state set, $S_0 \subseteq S$ is an initial-state set, $F \subseteq S$ is a final-state set, and $\rho : S \times \Sigma \to 2^S$ is a transition function, providing for each state and letter a set of possible successor states. $A$ is a 2NFA if it has a transition function $\rho : S \times \Sigma \to 2^{S \times \Sigma}$ is a transition function, providing for each state and letter a set of possible successor states and directions. An accepting run of $A$ on a word $w = w_1 \cdots w_m$ is a sequence $(s_1, i_1), \ldots, (s_m, i_m)$, where $s_j \in S$ and $1 \leq i_j \leq m$ for $1 \leq j \leq m, s_1 \in S_0, i_1 = 1, s_m \in F$, and $i_m = m + 1$, and the following holds for $1 \leq j < m$: there is a pair $(s_j, a_{i_j}) \in \rho(s_j, a_{i_j})$ such that $s_{j+1} = s_j + c$.

**Lemma 3.** [14] Let $A$ be an $n$-state NFA over $\Sigma^\pm$. Then there is a 2NFA for $\text{fold}(L(A))$ with $n \cdot (|\Sigma|^\pm + 1)$ states.

According to Language-Theoretic Lemma 2, to check that $Q_1 \subseteq Q_2$, we need to check $L(Q_1) \subseteq \text{fold}(L(Q_2))$. This requires the ability to complement 2NFAs. If we use the standard approach, we’d first convert the 2NFA to an NFA with an exponential blow-up and then complement the latter with another exponential blow-up [30], resulting in a doubly-exponential blow-up. Instead, we accomplish both tasks on a singly-exponential blow-up.

**Lemma 4.** [56] Let $A$ be a 2NFA over $\Sigma$. There is an NFA $A'$ such that

- $L(A') = \Sigma^\pm - L(A)$
- $|A'| \in 2^{O(|A|)}$

We now have the “technology” to establish complexity bounds for 2RPQ containment.

**Theorem 5.** Containment of 2RPQs is PSPACE-complete.

**Proof.** Containment of RPQs is a special case, which implies PSPACE-hardness. To establish the PSPACE upper bound, we use the following steps in order to test $Q_1 \subseteq Q_2$:

1. Construct NFAs $A_1, A_2$ such that $L(A_1) = L(Q_1)$. This step involves a linear blow-up [30].
2. Construct a 2NFA $A_2'$ such that $L(A_2') = \text{fold}(L(A_2))$. The step involves a polynomial blow-up (Lemma 3).
3. Construct an NFA $A_3'$ such that $L(A_3') = (\Sigma^\pm)^* - L(A_2')$. This step involves an exponential blow-up (Lemma 4).
4. Construct an NFA $A = A_1 \times A_3'$ such that $L(A) = L(Q_1) - \text{fold}(L(Q_2))$. This requires taking the product of $A_1$ and $A_3'$, involving a quadratic blow-up [30].
5. Check if there is a path from start state to final state in $A$. This requires nondeterministic logarithmic space in the size of $A$.

Again, we construct $A$ on the fly, constructing states only as we search for a path from a start state to a final state in $A$. This can be done in polynomial space, establishing the upper bound.

What the above treatment of query containment for 2RPQs shows is that the theory of regular expressions over the extended alphabet $\Sigma^\pm$ diverges from the theory of 2RPQs over $\Sigma^\pm$. Regular expressions over $\Sigma^\pm$ simply define languages over $\Sigma^\pm$, while 2RPQs over $\Sigma^\pm$ define queries by means of semipaths. Nevertheless, we were able to develop automata-theoretic technique to develop a query-containment algorithm for RPQs.

### 3.3 Adding Conjunction

The difference between regular expressions and regular path queries becomes even sharper when we consider the conjunction operator. Over word languages, conjunction corresponds to intersection. That is, the semantics of the regular expression $e_1 \land e_2$ is $L(e_1) \cap L(e_2)$. But regular languages are closed under intersection, which means that adding intersection to regular expressions does not make them more expressive. (Though they do make them more succinct [53].) Let, however, $Q_1$ and $Q_2$ be RPQs, and consider the following two queries: $(Q_1 \cap Q_2)(x, y)$ and $(Q_1(x, y) \land Q_2(x, y))$. The former requires $x$ and $y$ to be connected by a path matching both $Q_1$ and $Q_2$. In contrast, the latter requires $x$ and $y$ to be connected by two paths, one matching $Q_1$ and one matching $Q_2$. Thus, while conjunction does not add expressiveness to regular expressions, it does add expressiveness to regular path queries.

Thus, it makes sense to define an analogue of CQs in the context of graph databases as the class of conjunctive 2RPQs, or C2RPQs [11]. A C2RPQ is a conjunctive query where instead of atoms $r(x, y)$ we have atoms $e(x, y)$, where $r$ is a 2RPQ. To evaluate a C2RPQ $Q$ over a graph database $D$ we first evaluate all the 2RPQs appearing in $Q$, instantiating each as a binary relation over the elements of $D$, and then evaluating $Q$ as a conjunctive query over this collection of relations. We can then define the query class UC2RPQ.
as the class of unions of C2RPQ, analogously to the class UCQ over relational databases. The class UC2RPQ is not only natural as the graph-database analog of UCQ, but is also well-motivated by graph-database applications [10, 43].

Example 1. Consider the query in C2RPQ defined by means of the following rule:

\[ Q(x, y) : -r(x, y) \land r(x, z) \land r(y, z) \]

Here a pair \((a, b)\) is in the answer \(Q(D)\) if there is a node \(c\), and \(r\)-edges from \(a\) to \(b\), from \(a\) to \(c\), and from \(b\) to \(c\). If we now add the rule:

\[ Q(x, y) : -r(x, y) \land r(y, z) \land r(z, x) \]

then \(Q\) is in UC2RPQ.

Solving the query-containment problem requires going beyond just automata-theoretic techniques. After all, UCQs over graph databases are a special case of UC2RPQ, so a query-containment algorithm for UC2RPQ also has to solve the query-containment problem for UCQ. What is required is an algorithm that combines the automata-theoretic techniques described above for query containment for 2RPQs, with the homomorphism-based techniques developed in [18, 50] for solving the containment of Datalog in UCQ. A framework that combines automata with homomorphisms was developed in [20] for solving the containment of Datalog in UCQ. The basic idea of that framework is the observation that one needs to consider only a finite (although exponentially large) set of homomorphism types, which can be encoded by a finite alphabet, enabling the application of automata. This approach was pursued in [11], where the following theorem is proved:

Theorem 6. The query-containment problem for UC2RPQ is 2EXSPACE-complete.

3.4 Adding Transitive Closure

What are the basic operations from which queries in UC2RPQ are composed? Just like UCQ, we have selection, projection, conjunction, and unions, but we also have transitive closure that appear inside 2RPQs. But while the class UC2RPQ is closed under all the operations of UCQs, it is not closed under transitive closure. Consider for example the “triangle query” we saw earlier:

\[ Q(x, y) : -r(x, y) \land r(y, z) \land r(z, x) \]

(which is in C2RPQ). The syntax of UC2RPQ does not allow us to form a query expressing the transitive closure \(Q^+\) of \(Q\), and it is not difficult to show that \(Q^+\) is not in UC2RPQ. Thus, unlike CQ and UCQ, which can be defined as a class of queries closed under some basic operations, this is not the case for UC2RPQ. We therefore define the class \(RQ\) of regular queries by simply closing UC2RPQ under transitive closure. That is, \(RQ\) consists of the class of queries one can form from atomic queries \(r(x, y)\) using the following operations:

- If \(Q(\bar{x})\) is in \(RQ\), and \(y\) and \(z\) are variables in \(\bar{x}\), then \(Q \land y = z\) is in \(RQ\) (selection).
- If \(Q(\bar{x})\) is in \(RQ\), and \(y\) is a variable in \(\bar{x}\), then \((\exists y)Q\) is in \(RQ\) (projection).
- If \(Q(\bar{x})\) and \(Q_2(\bar{y})\) are in \(RQ\), then \(Q_1 \lor Q_2\) and \(Q_1 \land Q_2\) are in \(RQ\) (disjunction and conjunction).
- If \(Q(\bar{x}, y)\) is in \(RQ\), then \(Q^+\) is in \(RQ\) (transitive closure).

Note that the first four operations define UCQ, so \(RQ\) just add transitive closure to set of basic operations defining the query class.

It remains to consider the query-containment problem for \(RQ\). In [34] it was argued that, one one hand, \(RQ\) can be viewed as a fragment of Datalog (see discussion below), and, on the other hand, \(RQ\) can be viewed as a fragment of monadic second-order logic (MSO). Then using MSO techniques developed by Courcelle [26], decidability of containment can be shown, cf. [20]. This approach, however, yields an upper bound of nonelementary time, that is, its time complexity cannot be bounded by a bounded-height tower of exponential, so the question of elementary decidability was left open. Finally, the following theorem was proved in [48]:

Theorem 7. The query-containment problem for \(RQ\) is 2EXSPACE-complete.

The proof builds on techniques, developed in [11, 13, 20] for combining automata-theoretic techniques with database-theoretic techniques, but the techniques in [48] are quite more sophisticated than those required for UC2RPQs, pushing the upper bound (with a matching lower bound) to 2EXPSPACE. The bottom line, however, is that \(RQ\) is an expressive and natural class of graph queries, with an elementarily decidable query-containment problem.

4. BACK TO DATALOG

4.1 From RQ to Datalog

As mentioned earlier, \(RQ\) can be embedded in Datalog:

- Atoms: If \(r(x, y)\) is an atom, use the rule
  \[ Q(x, y) : -r(x, y) \]
- Selection: If \(Q(\bar{x})\) is IDB, and \(y\) and \(z\) are variables in \(\bar{x}\), then use the rule
  \[ Q'(\bar{x} - y) : -Q(\bar{x}[y/x]) \]
  where \(\bar{x} - y\) is \(\bar{x}\) with \(y\) deleted, and \(\bar{x}[y/x]\) is \(\bar{x}\) with \(y\) replaced by \(x\).
- Projection: If \(Q(\bar{x})\) is an IDB, and \(y\) is a variable in \(\bar{x}\), then use the rule
  \[ Q'(\bar{x} - y) : -Q(\bar{x}) \]
- Union: If \(Q_1(\bar{x})\) and \(Q_2(\bar{y})\) are IDBs, then use the rules
  \[ Q(\bar{x}) : -Q_1(\bar{x}) \]
  \[ Q(\bar{x}) : -Q_2(\bar{x}) \]
- Conjunction: If \(Q_1(\bar{x})\) and \(Q_2(\bar{y})\) are IDBs, then use the rule
  \[ Q(\bar{x} \cup \bar{y}) : -Q_1(\bar{x}), Q_2(\bar{y}) \]
  where \(\bar{x} \cup \bar{y}\) combines the variables in \(\bar{x}\) and \(\bar{y}\).
Datalog rules. The fifth items tell us that recursion can be in the list above; they correspond to standard nonrecursive Datalog rules. The fifth items tell us that recursion can be used only to define transitive closure of binary relations. It is this restriction of recursion in Datalog that is the basis for the decidability of query containment for RQ.

Note that this version of Datalog is defined over graph databases. It is only the first rule that is restricted to binary atoms. If we allow arbitrary relational atoms, then we get a version of Datalog where recursion is limited to defining transitive closure of binary relations. We call this class of queries GRQ, for Generalized Regular Queries. The decidability results of for RQ query containments lifts fairly easily to GRQ, since it is possible to encode relation of arbitrary arity by binary relations [48].

**Theorem 8.** The query-containment problem for GRQ is 2EXPSPACE-complete

So GRQ is a fragment of Datalog that can express connectivity properties, yet has an elementarily decidable query-containment problem.

### 4.2 Practical Aspects

The search for an expressive query languages with a decidable query-containment problem was motivated by practical applications, but it is too early to state that the main result, the decidability of query containment for GRQ, is a practical result. After all, the problem 2EXPSPACE-complete, which means that we cannot hope to have a better than a triply-exponential time upper bound. We should not confuse, however, a theoretical worst-case complexity bound with algorithmic behavior on real-world instances. Over the past 15 years we have witnessed impressive progress in the development of tools that perform well on real-world instances, in spite of pessimistic worst-case complexity bounds, cf. [24, 38]. To assess the usefulness of the algorithms underlying the decidability results here, would require careful empirical research [57, 58].

Furthermore, despite the centrality of the query-containment problem in database theory, it is far from clear that this centrality has translated to impact on database practice. The standard approach in query optimization is cost-based optimization, where we search for a low-cost plan in a space of alternative query-evaluation plans and their estimated costs [19]. (See [55] for an example of recent work in this area.) Very little empirical research has been done on structural query optimization, as was envisioned, say, in [5] (but see [39] for exception). Thus, the practical usefulness of query containment is still very much an open question.

### 5. ACKNOWLEDGMENTS:

I am grateful for my collaborators on this research project: Diego Calvanese, Giuseppe De Giacomo, Vincent Jugé, Maurizio Lenzerini, Miguel Romero, and Juan Reutter.

### 6. REFERENCES


