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 off *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], describe 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 language [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 relationals 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

PODS '16, June 26–July 1, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2902251.2902305

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 that 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 words, conjunction and intersection coincide, and regular languages are closed under intersection. Thus, extending regular expressions with intersection/conjunction does not yield an increase in expressive power [30]. In contrast, over graphs, conjunction differs from intersection. Since we are seeking a graph-database analogue to the class UCQ of relational queries, it does make sense to close the class 2RPQ of 2-way regular path queries under both disjunction (which corresponds to union) and conjunction, resulting in the class UC2RPQ, which is the class of unions of conjunctions of 2RPQs. The decidability of of the containment problem for UC2RPQ cannot be shown using only automata-theoretic techniques; rather one has to combine automata-theoretic techniques with database-theoretic techniques, showing that the problem is EXPSPACE-complete [11].

Yet the class UC2RPQ cannot be considered a natural class of graph-database queries. A natural class should be defined by means of basic queries and closure under certain operations. For example, the class CQ is the class of atomic relational queries, closed under selection, conjunction, and projection, while the class UCQ is closed also under disjunction (union). In contrast, the class UC2RPQ is closed under election, isjunction, and conjunction, but it is not closed under transitive closure, which can appear only inside 2RPQs. A natural definition for the class RQ of *regular queries* is the closure of atomic queries under selection, projection, disjunction, conjunction, and transitive closure.

The query-containment problem for RQ has been open for several years. Decidability can be shown via embedding in monadic-second order logic (MSO) [34], but this yields a nonelemenary complexity upper bound, that is, a tower of exponentials of unbounded height. Only recently it was shown that the problem has elementary complexity – it is 2EXPSPACE-complete [48]. As in [11], the proof requires a combination automata-theoretic and database-theoretic techniques (cf. [20]), but these techniques are quite more sophisticated than those required for UC2RPQs.

The classes of graph-database queries we have discussed so far, namely, RPQ, 2RPQ, UC2RPQ, and RQ, can all be expressed in graph-database Datalog. Thus, they can all be viewed as fragments of Datalog with a decidable query-containment problem. The class RQ, however, is notable, as it can be viewed as Datalog where recursion can be used only to express only transitive closure.

It turns out that this restriction of recursion is the key to the decidability result for RQ. Let GRQ be the class of *generalized regular queries*, consisting of all Datalog queries where recursion is used only to express transitive closure. The query-containment problem for GRQ can be reduced to that of RQ yielding the same 2EXPSPACE bound [48]. Thus, the class GRQ provides an answer to the long-standing open question of identifying a fragment of Datalog that is expressive enough to capture connectivity properties, while still having a decidable query-containment problem.

The outline of the paper is as follows. Following the introduction, we introduce in Section 2 the relational query languages UCQ and Datalog, and review the open question of finding a fragment of Datalog that is expressive enough for applications yet has a decidable query-containment problem. In Section 3 we review the development of more and more expressive graph-database query languages with a decidable query-containment problem, culminating with the class RQ of Regular Queries. We conclude in Section 4 with the de-

scription of GRQ, the class of Generalized Regular Queries, as an expressive fragment of Datalog with an elementarily decidable query-containment problem, and discuss the practical implications of this result.

2. 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 $\wedge$ and the only quantifier allowed is $\exists$. Without loss of generality, we can assume that conjunctive queries are given as formulas $\theta(x_1, \ldots, x_k)$ of the form $(\exists y_1, \ldots, y_m)(a_1 \wedge \ldots a_n)$ 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 $(\exists y)(E(x, y) \wedge E(y, z))$ is satisfied by all pairs $\langle x, z\rangle$ 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 Θ 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 $p(z_1, .., z_l)$ where p is a predicate symbol and $z_1..z_l$ 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 Π 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 Π and let $P^\infty_\Pi(D)$ be the collection of facts about P that can be deduced from D by an arbitrary number of applications of the rules in Π, that is,

$$P^\infty_\Pi(D) = \bigcup_{i \geq 0} P^i_\Pi(D).$$

A Datalog query Q is a Datalog program Π with one IDB predicate P designated as the *goal predicate*. Then we have $Q(D) = P^\infty_\Pi(D)$. For detailed semantics, see [15].

It is known (cf. [46]) that the relation defined by an IDB predicate in a Datalog program Π, i.e., $Q^\infty_\Pi(D)$, can be defined by a *possibly infinite* union of conjunctive queries. That is, for each IDB predicate Q there is a possibly infinite sequence $\phi_0, \phi_1, \ldots$ of conjunctive queries such that for every database D, we have $Q^\infty_\Pi(D) = \bigcup_{i=0}^\infty \phi_i(D)$.

A predicate P *depends* on a predicate Q in a program Π, if Q occurs in the body of a rule r of Π and P is the predicate at the head of r. The *dependence graph* of Π is a directed graph whose nodes are the predicates of Π, and whose edges capture the dependence relation, i.e., there is an edge from Q to P if P appears in the body of a rule with Q in the head; that is Q *depends* on P. If there is a dependence-graph path from a predicate Q to itself, we say that Q is a *recursive predicate*. A program Π is *nonrecursive* if it has no recursive predicates, i.e., no predicate depends recursively on itself. It is well-known [2] that a nonrecursive program can be expressed as a *finite* union of conjunctive queries. Thus, nonrecursive Datalog is equivalent to the query class UCQ. Thus, we can view Datalog as the extension of the query class UCQ with recursion.

Datalog has been a subject of extensive research both from a theoretical, cf. [35], and applied, cf. [45], perspectives. This research led to the extension of SQL standard with recursion, *common table expressions* [29]. In recent years variants of Datalog have gained popularity in various application areas from access-control policies [28] to declarative networking [37]. These applications use recursion in an essential way, which means that nonrecursive Datalog is not expressive enough for them. For example, in declarative networking it is important to say that there is a network connection of some unknown length between nodes x and y.

2.3 Query Containment

Let Q_1 and Q_2 be two queries. We say that Q_1 is *contained* in Q_2, denoted $Q_1 \subseteq Q_2$, if $Q_1(D) \subseteq Q_2(D)$ for every database D. Clearly, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$, then Q_1 and Q_2 are equivalent, which explains the centrality of the query-containment problem in relational database research. As discussed in Section 1, many database applications make use of query containment. A celebrated results in database theory is the decidability of query containment for CQ – the problem is NP-complete [18]. (Of course, tractability is an important issue, to be discussed in Section 4.) This was extended a few years later to UCQ, though with higher complexity [50]. As nonrecursive Datalog is equivalent to UCQ, it follows that decidability of query containment extends also to the former, but with an even higher complexity [8].

It is easy, however, to see that decidability of query containment does not extend to full Datalog. The syntactic similarity of Datalog programs and context-free grammars suggests that the containment problem for context-free grammars can be reduced to the containment problem for Datalog, implying undecidability [52]. So nonrecursive Datalog has a decidable query-containment problem but is not expressive enough for applications, yet full Datalog is expressive enough, but has an undecidable query-containment problem. Can we find a fragment of Datalog that would give us both expressiveness and decidability? That question has been open for many years.

Monadic Datalog is a fragment of Datalog. In Monadic Datalog all recursive predicates must be one-place predi-

cates, that is, they define sets of values. Monadic Datalog 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:

$$\begin{array}{ll} Q(X) & :-E(X,Y), P(Y) \\ Q(X) & :-E(X,Y), Q(Y) \end{array}$$

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

$$\begin{array}{ll} E^+(X,Y) & :-E(X,Y) \\ E^+(X,Z) & :-E(X,Y), E^+(Y,Z) \end{array}$$

(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 properties that are needed in applications such as declarative networking. Thus, Monadic Datalog does not provide a query language that is expressive enough, but with a decidable query-containment problem.

3. GRAPHH 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 Σ. 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) relational structure over the set Σ of binary relational symbols. Thus, the edge alphabet Σ 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 simply 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 Σ. The *answer* $Q(D)$ to an RPQ Q over D is the set of pairs of objects connected in D by a directed path traversing a sequence of edges forming a word in the regular language $L(Q)$ defined by Q [41].

RPQs allow for navigating the edges of a semistructured databases only in the forward direction. RPQs extended with the ability of navigating database edges backward are called *two-way regular-path queries* (2RPQs) [12]. Formally, we consider an alphabet $\Sigma^{\pm} = \Sigma \cup \{r^- \mid r \in \Sigma\}$, which includes a new symbol r^- for each relation symbol r in Σ. The symbol r^- denotes the *inverse* of the binary relation r. If $p \in \Sigma^{\pm}$, then we use p^- to mean the inverse of p, i.e., if p is r, then p^- is r^-, and if p is r^-, then p^- is r. A 2RPQ over Σ is expressed as a regular expression or a finite-state automaton over $\Sigma^{\pm}$. The *answer* $Q(D)$ *to a 2RPQ* 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 $p_i = r$ then $(y_{i-1}, y_i) \in r(D)$, and if $p_i = r^-$ then $(y_i, y_{i-1}) \in r(D)$). Intuitively, a semipath $(y_0, p_1, y_1, \ldots, y_{n-1}, p_n, y_n)$ corresponds to a navigation 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 distinct. 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 2RPQ, we first consider RPQs, where we do not allow inverse symbols. We characterize query containment via a fundamental lemma [14].

LEMMA 1. (Language-Theoretic Lemma 1)*: Let* Q_1, Q_2 *be RPQs. Then* $Q_1 \sqsubseteq Q_2$ *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 instructive to recall the proof of the upper bound. The key is the observation 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 $L(A_i) = L(E_i)$. 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 complementation requires an application of the subset construction.
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 requires 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 Theorem [51] to eliminate the nondeterminism in step (4).)

Extending this result to 2RPQs encounters two difficulties. The first difficulty is that an automata-theoretic approach would most likely involve two-way automata, due to the presence of inverse letters, but extending the result 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 $Q_1 \sqsubseteq Q_2$, as every semipath (x, p, y), which establishes that $(x, y) \in Q_1(\mathcal{B})$, corresponds to the semipath (x, p, y, p^-, x, p, y), establishing that $(x, y) \in Q_2(\mathcal{B})$. At the same time, we clearly do not have $L(Q_1) \subseteq L(Q_2)$, since $p \notin L(Q_2)$. 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, $v \rightsquigarrow u$, if v can be "folded" on u, e.g., $abb^-bc \rightsquigarrow abc$. Formally, we say that $v = v_1 \cdots v_m$ 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 $i_m = n$, 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 $v_{j+1} = u^-_{i_{j+1}}$.

(In particular, $v_0 = u_0$ and $v_m = u_n$.) For example, the sequence demonstrating that $abb^-bc \rightsquigarrow abc$ is $0, 1, 2, 1, 2, 3$. Pictorially, $\xrightarrow{a} \cdot \xrightarrow{b} \cdot \xleftarrow{b} \cdot \xrightarrow{b} \cdot \xrightarrow{c} \rightsquigarrow \xrightarrow{a} \cdot \xrightarrow{b} \cdot \xrightarrow{c}$. Let L be a language $\Sigma^{\pm}$. We define $\mathit{fold}(L) = \{u \ : \ v \rightsquigarrow u, v \in L\}$. 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 $Q_1 \sqsubseteq Q_2$ iff $L(Q_1) \subseteq \mathit{fold}(L(Q_2))$.*

We now show that if A is an NFA, then $\mathit{fold}(L(A))$ 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 Σ 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 $\rho : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$, providing for each state and letter a set of possible successor states and directions. An accepting run of A on a word $w = w_1 \cdots w_n$ is a sequence $(s_1, i_1), \ldots, (s_m, i_n)$, where $s_j \in S$ and $1 \leq i_j \leq n$ for $1 \leq j < m$, $s_1 \in S_0$, $i_1 = 1$, $s_m \in F$, and $i_m = n + 1$, and the following holds for $1 \leq j < m$: there is a pair $(s_{j+1}, c) \in \rho(s_j, a_{i_j})$ such that $i_{j+1} = i_j + c$.

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

According to Language-Theoretic Lemma 2, to check that $Q_1 \sqsubseteq Q_2$, we need to check $L(Q_1) \subseteq \mathit{fold}(L(Q_2))$. 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 Σ. There is an NFA A^c such that*

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

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 $Q_1 \sqsubseteq Q_2$:

1. Construct NFAs A_1, A_2 such that $L(A_i) = L(Q_i)$. This step involves a linear blow-up [30].
2. Construct a 2NFA A_2' such that $L(A_2') = \mathit{fold}(L(A_2))$. The step involves a polynomial blow-up (Lemma 3).
3. Construct an NFA A_2^c such that $L(A_2^c) = (\Sigma^{\pm})^* - L(A_2')$. This step involves an exponential blow-up (Lemma 4).
4. Construct an NFA $A = A_1 \times A_2^c$ such that $L(A) = L(Q_1) - \mathit{fold}(L(Q_2))$. This requires taking the product of A_1 and A_2^c, 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 defines languages over $\Sigma^{\pm}$, while 2RPQs over $\Sigma^{\pm}$ definde queries by means of semipaths. Nevertheless, we were able to develop automata-theoretic technique to develop a query-containment algorithm for 2RPQs.

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 $e_1 \wedge e_2$ 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 succint [53].) Let, however, Q_1 and Q_2 be RPQs, and consider the following two queries: $(Q_1 \cap Q_2)(x, y)$ and $Q_1(x, y) \wedge Q_2(x, y)$. 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 $\kappa(x, y)$, where κ 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:

$$Q(x,y) \ :- r(x,y)\&r(x,z)\&r(y,z)$$

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:

$$Q(x,y) \ :- r(x,y)\&r(y,z)\&r(z,x)$$

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 tecniques developed in [18, 50] for solving query containment for UCQs. 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 EXPSPACE-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 sloced under all the operations of UCQs, it is not closed under transitive closure. Consider for example the "triangle query" we saw earlier:

$$Q(x,y) \ :- r(x,y)\&r(y,z)\&r(z,x)$$

(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 us 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 $Q(\vec{x})$ is in RQ, and y and z are variables in $\vec{x}$, then $Q \wedge y = z$ is in RQ (selection).
- If $Q(\vec{x})$ is in RQ, and y is a variable in $\vec{x}$, then $(\exists y)Q$ is in RQ (projection).
- If $Q_1(\vec{x})$ and $Q_2(\vec{y})$ are in RQ, then $Q_1 \vee Q_2$ and $Q_1 \wedge Q_2$ are in RQ (disjunction and conjunction).
- If $Q(x,y)$ 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 2EXPSPACE-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 $Q(\vec{x})$ is IDB, and y and z are variables in $\vec{x}$, then use the rule
$$Q'(\vec{x}-y) : -Q(\vec{x}[y/x])$$
where $\vec{x}-y$ is $\vec{x}$ with y deleted, and $\vec{x}[y/x]$ is $\vec{x}$ with y replaced by x.
- Projection: If $Q(\vec{x})$ is an IDB, and y is a variable in $\vec{x}$, then use the rule
$$Q'(\vec{x}-y) : -Q(\vec{x})$$
,
- Union: If $Q_1(\vec{x})$ and $Q_2(\vec{x})$ are IDBs, then use the rules
$$Q(\vec{x}) : -Q_1(\vec{x})$$
$$Q(\vec{x}) : -Q_2(\vec{x})$$
- Conjunction: If $Q_1(\vec{x})$ and $Q_2(\vec{y})$ are IDBs, then use the rule
$$Q(\vec{x} \cup \vec{y}) : -Q_1(\vec{x}), Q_2(\vec{y})$$
where $\vec{x} \cup \vec{y}$ combines the variables in $\vec{x}$ and $\vec{y}$.

- Transitive Closure: If $Q(x, y)$ is IDB, then use the rules

$$Q^+(x, y) : -Q(x, y)$$

$$Q^+(x, z) : -Q^+(x, y), Q(y, z)$$

Note that there is nothing special about the first four items 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 relatoions [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 then 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 algoritms 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 alterntaive 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, Vencent Jugé, Maurizio Lenzerini, Miguel Romero, and Juan Reutter.

6. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann, San Mateo, CA, 2000.

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

[3] S. Abiteboul and V. Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[4] A. Aho and J. Ullman. Universality of data retrieval languages. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 110–117, 1979.

[5] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.

[6] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA, 1988.

[7] M. Benedikt, P. Bourhis, and M. Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *Proc. 31st IEEE Symp. on Logic in Computer Science*. ACM, 2016. To Appear.

[8] M. Benedikt and G. Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1):297–308, 2010.

[9] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML query language – w3c working draft. Technical report, World Wide Web Consortium, 2003.

[10] P. Buneman. Semistructured data. In *Proc. 16th ACM Symp. on Principles of database systems*, pages 117–121. ACM, 1997.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning*, pages 176–185, 2000.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Query processing using views for regular path queries with inverse. In *Proc. 19th ACM Symp. on Principles of Database Systems*, pages 58–66, 2000.

[13] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.*, 336(1):33–56, 2005.

[14] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Vardi. Reasoning on regular path queries. In *SIGMOD Record*, volume 32:4, pages 83–92, 2003.

[15] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowledge and Data Engineering*, 1(1):146–166, 1989.

[16] D. Chamberlin and R. Boyce. SEQUEL: A structured english query language. In *Proc. ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 249–264. ACM, 1974.

[17] A. Chandra and D. Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 1:1–15, 1985.

[18] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc.*

9th ACM Symp. on Theory of Computing, pages 77–90, 1977.

[19] S. Chaudhuri. An overview of query optimization in relational systems. In *Proc. 17th ACM Symposium on Principles of Database Systems*, pages 34–43. ACM Press, 1998.

[20] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *Journal of Computer and System Sciences*, 54:61–88, 1997.

[21] J. Clark and S. DeRose. XML Path Language (XPath) version 1.0 – W3C recommendation 16 november 1999. Technical report, World Wide Web Consortium, 1999.

[22] E. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13:377–387, 1970.

[23] E. Codd. Relational completeness of databases sublanguages. In R. Rustin, editor, *Data Base Systems*, pages 65–98. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[24] B. Cook, A. Podelski, and A. Rybalchenko. Proving program termination. *Communications of the ACM*, 54(5):88–98, 2011.

[25] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 477–490, 1988.

[26] B. Courcelle. The monadic second-order theory of graphs iii: Tree decmpositions, minors, and complexity issues. *Theoret. Inform. Applic.*, 26:257–286, 1992.

[27] C. Date and H. Darwen. *A Guide To SQL Standard*, volume 3. Addison-Wesley Reading, 1997.

[28] D. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Automated Reasoning*, pages 632–646. Springer, 2006.

[29] A. Eisenberg and J. Melton. Sql: 1999, formerly known as sql3. *ACM Sigmod record*, 28(1):131–138, 1999.

[30] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley, 2000.

[31] I. I. Tatarinov and A. Halevy. Efficient query reformulation in peer data management systems. In *Proc. of the 2004 ACM Conference on Management of Data*, pages 539–550. ACM, 2004.

[32] Y. Ioannidis. Query optimization. *Computing Surveys)*, 28(1):121–123, 1996.

[33] J. D. J.D. Ullman. Information integration using logical views. In *Proc. Int'l Conf. on Database Theory*, pages 19–40. Springer, 1997.

[34] V. Jugé and M. Vardi. On the containment of Datalog in Regular Datalog. Technical report, Rice University, 2009.

[35] P. G. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995.

[36] M. Lanzerini. Data integration: A theoretical perspective. In *Proc. 21st ACM Symp. on Principles of Database Systems*, pages 233–246, 2002.

[37] B. Loo, T. Condie, M. Garofalakis, D. Gay, J. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.

[38] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009.

[39] B. McMahan, G. Pan, P. Porter, and M. Vardi. Projection pushing revisited. In *Proc. 9th Int'l Conf. on Extending Database Technology*, volume 2992 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2004.

[40] E. Meijer and G. Bierman. A co-relational model of data for large shared data banks. *Communications of the ACM*, 54(4):49–58, 2011.

[41] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[42] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. of the 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972.

[43] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. 7th Int'l Conf. on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1999.

[44] J. Minker. An overview of cooperative answering in databases. In *Flexible Query Answering Systems*, pages 282–285. Springer, 1998.

[45] K. Morris, J. D. Ullman, and A. Van Gelder. Design overview of the nail! system. In *Proc. 3rd Int'l Conf. on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 554–568. Springer, 1986.

[46] J. Naughton. Minimizing function-free recursive inference rules. *J. ACM*, 36(1):69–91, 1989.

[47] J. Rao and K. Ross. Reusing invariants: A new strategy for correlated queries. In *ACM SIGMOD Record*, volume 27:2, pages 37–48. ACM, 1998.

[48] J. Reutter, M. Romero, and M. Vardi. Regular queries on graph databases. In *Proc. 18th Int'l Conf. on Database Theory*, volume 31 of *LIPIcs*, pages 177–194. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[49] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2015.

[50] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[51] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[52] O. Shmueli. Equivalence of Datalog queries is undecidable. *J. of Logic Programming*, 15(3):231–241, 1993.

[53] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 1–9, 1973.

[54] M. Stonebraker, G. Held, E. Wong, and P. Kreps. The design and implementation of ingres. *ACM*

Transactions on Database Systems, 1(3):189–222, 1976.

[55] I. Trummer and C. Koch. An incremental anytime algorithm for multi-objective query optimization. In *Proc. ACM Int'l Conf. on Management of Data*, pages 1941–1953, 2015.

[56] M. Vardi. A note on the reduction of two-way automata to one-way automata. *Inf. Process. Lett.*, 30(5):261–264, 1989.

[57] M. Vardi. Solving the unsolvable. *Commun. ACM*, 54(7):5, 2011.

[58] M. Vardi. Boolean satisfiability: theory and engineering. *Commun. ACM*, 57(3):5, 2014.