Reasoning on Regular Path Queries

D. Calvanese, G. De Giacomo, M. Lenzerini
Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

M. Y. Vardi
Dept. of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

Abstract

Current information systems are required to deal with more complex data with respect to traditional relational data. The database community has already proposed abstractions for these kinds of data, in particular in terms of semistructured data models. A semistructured model conceives a database essentially as a finite directed labeled graph whose nodes represent objects, and whose edges represent relationships between objects. In the same way as conjunctive queries form the core of any query language for the relational model, regular path queries (RPQs) and their variants are considered the basic querying mechanisms for semistructured data.

Besides the basic task of query answering, i.e., evaluating a query over a database, databases should support other reasoning services related to querying. One of the most important is query containment, i.e., verifying whether for all databases the answer to a query is a subset of the answer to a second query. Another important reasoning service that has received considerable attention in the recent years is view-based query processing, which amounts to processing queries based on a set of materialized views, rather than on the raw data in the database.

The goal of this paper is to describe basic results and techniques concerning query containment and view based query processing for the class of two-way regular-path queries (which extend RPQs with the inverse operator). We will demonstrate that the basic services for reasoning about two-way regular path queries are decidable, thus showing that the limited form of recursion expressible by these queries does not endanger the decidability of reasoning. Besides the specific results, our methods show the power of two-way automata in reasoning on complex queries.

Database Principles Column. Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

1 Introduction

Nowadays, information systems are required to deal with more complex data with respect to traditional relational data. For example, data on the web, or biological data are better described by resorting to more flexible structuring mechanisms than those provided by relational systems. The database community has already proposed abstractions for these kinds of data, in particular in terms of semistructured data models. A semistructured model conceives a database essentially as a finite directed labeled graph whose nodes represent objects, and whose edges represent relationships between objects [1].

In the same way as conjunctive queries [18] (CQs) form the core of any query language for the relational model, regular path queries (RPQs) are considered the basic querying mechanisms for semistructured data [7, 3, 1]. Query languages for this data model must indeed be equipped with flexible mechanisms for navigating the graph representing the database. 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 reflexive-transitive closure. Plain RPQs allow for navigating the edges of a semistructured databases only in the forward direction. However, it is obviously of interest to be able to navigate edges in both forward and backward directions, as, e.g., supported by the predecessor axis of XPath [20, 6]. RPQs extended with the ability of navigating database edges backward are called *two-way regular-path queries* (2RPQs) [11].

Besides the basic task of query answering, i.e., evaluating a query over a database, databases should support other reasoning services related to querying. One of the most important is query containment (which generalizes query equivalence), i.e., verifying whether for all databases the answer to a query is a subset of (resp., equal to) the answer to a second query. Checking containment of queries is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering [27, 35, 19, 4, 38, 34, 8, 23]. In [18] it is shown that CQ containment is equivalent to CQ evaluation (NP-complete).

(For some extensions, see [5, 39, 31, 43].) On the other hand, it is shown in [41] that containment of Datalog queries is undecidable. Note that Datalog may be seen as the language obtained by adding recursion (and union) to conjunctive queries.

View-based query processing is another form of reasoning that has recently drawn a great deal of attention in the database community [28, 29]. In several contexts, such as data integration, query optimization, query answering with incomplete information, and data warehousing, the problem arises of processing queries posed over the schema of a virtual database, based on a set of materialized views, rather than on the raw data in the database [42, 2, 32]. For example, an information integration system exports a global virtual schema over which user queries are posed, and such queries are answered based on the data stored in a collection of data sources, whose content in turn is described in terms of views over the global schema. In such a setting, each data source corresponds to a materialized view, and the global schema exported to the user corresponds to the schema of the virtual database. Notice that typically, in data integration, the data in the sources are correct (i.e., sound) but incomplete with respect to their specification in terms of the global schema. This is due the fact that typically the global schema is not designed taking the sources into account, but rather the information needs of users. Hence it may not be possible to precisely describe the information content of the sources. In this paper we will concentrate on this case (*sound views*)[1].

There are two approaches to view-based query processing, called *view-based query rewriting* and *view-based query answering*, respectively. In the former approach, we are given a query and a set of view definitions, and the goal is to reformulate the query into an expression of a fixed language that refers only to the views and provides the answer to the query. The crucial point is that the language in which we want the rewriting is fixed, and in general coincides with the language used for expressing the original query. In view-based query answering, besides the query and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples that are answers to the query in all the databases that are consistent with the views.

While query containment and view-based query processing have been extensively studied in the relational model in the last decades, results on these problems in the context of semistructured data are more recent and less known. The goal of this paper is to describe basic results and techniques concerning query containment and view based query processing for the class of two-way regular-path queries. We will demonstrate that the basic services for reasoning about two-way regular path queries are decidable, thus showing that the limited form of recursion expressible by these queries does not endanger the decidability of reasoning.

The paper is organized as follows. Section 2 provides the formal definitions of all the notions used in the paper. Section 3 illustrates the results on query containment. Sections 4 and 5 present the results for view-based query answering, and view-based query rewriting, respectively. Section 6 concludes the paper.

[1] This corresponds to adopting the so-called "open world assumption" for the views.

2 Framework

Following the usual approach in semistructured data [1], we define a *semistructured database* as a finite directed graph whose edges are labeled by elements from a given 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. Observe that a semistructured database can be seen as a (finite) relational structure over the set Σ of binary relational symbols. A *relational structure* (or simply *structure*) $\mathcal{B}$ over Σ is a pair $(\Delta^{\mathcal{B}}, \cdot^{\mathcal{B}})$, where $\Delta^{\mathcal{B}}$ is a finite domain and $\cdot^{\mathcal{B}}$ is a function that assigns to each relation symbol in $r \in \Sigma$ a binary relation $r^{\mathcal{B}}$ over $\Delta^{\mathcal{B}}$, also denoted by $r(\mathcal{B})$.

A *regular-path query* (RPQ) over Σ is expressed as a regular expression or a finite-state automaton over Σ. The *answer* $Q(\mathcal{B})$ *to an RPQ* Q *over a database* $\mathcal{B}$ is the set of pairs of objects connected in $\mathcal{B}$ by a directed path traversing a sequence of edges forming a word in the regular language $L(Q)$ defined by Q.

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) [11].

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* $Q(\mathcal{B})$ *to a 2RPQ* Q *over a database* $\mathcal{B}$ is the set of pairs of objects connected in $\mathcal{B}$ by a semipath that conforms to the regular language $L(Q)$. A *semipath* in $\mathcal{B}$ 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 $(y_{i-1}, y_i) \in r(\mathcal{B})$, and if $p_i = r^-$ then $(y_i, y_{i-1}) \in r(\mathcal{B})$. 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. A semipath is said to be *simple* if no object in it appears more than once. 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 $Q(\mathcal{B})$ if and only if, by starting from x, it is possible to reach y by navigating on $\mathcal{B}$

according to one of the words in $L(Q)$.

Besides the basic task of query answering, i.e., evaluating a query over a database, databases should support other reasoning services related to querying. One of the most important is query containment, i.e., verifying whether for all databases the answer to a query is a subset of (resp., equal to) the answer to a second query. Formally, given two queries Q_1 and Q_2 over Σ, we say that Q_1 is *contained in* Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for every database $\mathcal{B}$ over Σ, we have that $Q_1(\mathcal{B}) \subseteq Q_2(\mathcal{B})$.

Consider now a semistructured database that is accessible only through a collection of views expressed as RPQs/2RPQs, and suppose we need to answer a RPQ/2RPQ over the database only on the basis of our knowledge on the views. Specifically, the collection of views is represented by a finite set $\mathcal{V}$ of *view symbols*, each denoting a binary relation. Each view symbol $V \in \mathcal{V}$ has an associated *view definition* V^{Σ}, which is an RPQ/2RPQ over Σ. A $\mathcal{V}$*-extension* $\mathcal{E}$ is a relational structure over $\mathcal{V}$. We consider views to be *sound* [2, 25], i.e., we model a situation where the extension of the views provides a subset of the results of applying the view definitions to the database. Given a set $\mathcal{V}$ of views and a database $\mathcal{B}$, we use $\mathcal{V}^{\Sigma}(\mathcal{B})$ to denote the $\mathcal{V}$-extension $\mathcal{E}$ such that $V(\mathcal{E}) = V^{\Sigma}(\mathcal{B})$, for each $V \in \mathcal{V}$. We say that a $\mathcal{V}$-extension $\mathcal{E}$ is *sound wrt a database* $\mathcal{B}$ if $\mathcal{E} \subseteq \mathcal{V}^{\Sigma}(\mathcal{B})$. In other words, for a view $V \in \mathcal{V}$, all the tuples in $V(\mathcal{E})$ must appear in $V^{\Sigma}(\mathcal{B})$, but $V^{\Sigma}(\mathcal{B})$ may contain tuples not in $V(\mathcal{E})$.

Given a set $\mathcal{V}$ of views, a $\mathcal{V}$-extension $\mathcal{E}$, and a query Q over Σ, the set of *certain answers (under sound views)* to Q with respect to $\mathcal{V}$ and $\mathcal{E}$ is the set of pairs (x, y) of objects such that $(x, y) \in Q(\mathcal{B})$ for every database $\mathcal{B}$ wrt which $\mathcal{E}$ is sound, i.e., $\mathcal{E} \subseteq \mathcal{V}^{\Sigma}(\mathcal{B})$. *View-based query answering (under sound views)* consists in deciding whether a given pair of objects is a certain answer to Q with respect to $\mathcal{V}$ and $\mathcal{E}$. Given a set $\mathcal{V}$ of views and a query Q, we denote by $cert_{Q,\mathcal{V}}$ the query that, for every $\mathcal{V}$-extension $\mathcal{E}$, returns the set of certain answers under sound views to Q with respect to $\mathcal{V}$ and $\mathcal{E}$.

View-based query answering has also been tackled using an indirect approach, based on *view-based query rewriting*. According to such an approach, a query Q over the database alphabet is processed by first reformulating Q into an expression of a fixed query language over the view alphabet $\mathcal{V}$ (called *rewriting*), and then evaluating such an expression over the view extensions. The relationship between view-based query answering and view-based query rewriting is investigated in [29, 12, 13, 32].

Note that, in our setting, views are sound, and this property must be taken into account in the reformulation step of the rewriting process. However, most papers on rewriting queries using views are based, either implicitly or explicitly, on the exact view assumption, which states that the extension of a view provides exactly the result of applying the view definition to the database. It follows that we need to provide an adequate definition of rewriting in a setting where views are sound: let Q be a query over the database alphabet, and let Q_r be a query over the view alphabet $\mathcal{V}$. We say that Q_r is a *rewriting of* Q *under sound views* $\mathcal{V}$, if for every database $\mathcal{B}$ and for every $\mathcal{V}$-extension $\mathcal{E}$ with $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$, we have that $Q_r(\mathcal{E}) \subseteq Q(\mathcal{B})$.

Obviously, in view-based query rewriting, we are not interested in arbitrary rewritings, but we aim at computing rewritings that capture the original query at best. The problem of view-based query rewriting can in general be defined as follows: given a query Q and a set of views $\mathcal{V}$, find the set of rewritings of Q under sound views $\mathcal{V}$ that are *maximal* in a given class $\mathcal{C}$ of queries. A query Q_r in $\mathcal{C}$ is a rewriting of Q under sound views $\mathcal{V}$ that is maximal in $\mathcal{C}$ if:

1. Q_r is a rewriting of Q under sound views $\mathcal{V}$, and
2. there is no query Q'_r in $\mathcal{C}$ that is a rewriting of Q under sound views $\mathcal{V}$ and such that, for every database $\mathcal{B}$ and for every $\mathcal{V}$-extension $\mathcal{E}$ with $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$, we have that $Q_r(\mathcal{E}) \subsetneq Q'_r(\mathcal{E})$.

Since in this paper we are focusing on 2RPQs, we restrict our attention to the case where also rewritings are 2RPQs over the view alphabet $\mathcal{V}$, i.e., rewritings are expressed in the same language as queries over the database. For 2RPQs, we have that if Q'_r and Q''_r are rewritings of Q under sound views $\mathcal{V}$ that are maximal in the class of 2RPQs, then $Q'_r + Q''_r$ is still a maximal rewriting in the class of 2RPQs. It follows that Q'_r and Q''_r are equivalent, i.e., for every database $\mathcal{B}$, and for every $\mathcal{V}$-extension $\mathcal{E}$ such that $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$, both $Q'_r(\mathcal{E}) \subseteq Q''_r(\mathcal{E})$, and $Q''_r(\mathcal{E}) \subseteq Q'_r(\mathcal{E})$. Hence, all 2RPQ maximal rewritings for a 2RPQ Q coincide modulo equivalence, and we can refine the definition of maximal rewriting as follows. A 2RPQ Q_r over the alphabet $\mathcal{V}$ is a *maximal rewriting* of Q under sound views $\mathcal{V}$ if:

1. Q_r is a rewriting of Q under sound views $\mathcal{V}$, and
2. for every 2RPQ Q'_r that is a rewriting of Q under sound views $\mathcal{V}$, we have that, for every database $\mathcal{B}$, and for every $\mathcal{V}$-extension $\mathcal{E}$ with $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$, we have that $Q'_r(\mathcal{E}) \subseteq Q_r(\mathcal{E})$.

3 Query Containment

To illustrate our approach to query containment, we first consider RPQs, where we do not allow inverse symbols. We characterize query containment via a fundamental lemma.

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

Proof. Suppose first that $Q_1 \sqsubseteq Q_2$. Let $w = w_1 \cdots w_k \in L(Q_1)$. Consider a database $\mathcal{B}$ of the form:

$$x \cdot \xrightarrow{w_1} \cdots \xrightarrow{w_k} \cdot y.$$

That is, $\mathcal{B}$ consists of a path from x to y labeled with w. Clearly $(x,y) \in Q_1(\mathcal{B})$, so $(x,y) \in Q_2(\mathcal{B})$. It follows that $w \in L(Q_2)$).

Conversely, suppose that $L(Q_1) \subseteq L(Q_2)$. Let $(x,y) \in L(Q_1)$. Then there is a path $(y_0, p_1, y_1, \ldots, y_{n-1}, p_n, y_n)$ in $\mathcal{B}$, where $y_0 = x$, $y_n = y$, and $p_1 \cdots p_n \in L(Q_1)$. But then also $p_1 \cdots p_n \in L(Q_2)$, and $(x,y) \in Q_2(\mathcal{B})$. □

Since containment of regular expressions is known to be PSPACE-complete [36], 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 (1NFAs) A_1, A_2 such that $L(A_i) = L(E_i)$. This step involves a linear blow-up.
2. Construct a 1NFA $\overline{A_2}$ 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 a 1NFA $A = A_1 \times \overline{A_2}$ 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 [40] 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 [36] 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.

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

Proof. Suppose first that $Q_1 \sqsubseteq Q_2$. Let $u = u_1 \cdots u_n \in L(Q_1)$. Consider a database $\mathcal{B}$ of the form:

$$x \cdot \xrightarrow{u_1} \cdots \xrightarrow{u_n} \cdot y.$$

That is, $\mathcal{B}$ consists of a path from x to y labeled with u. Clearly $(x,y) \in Q_1(\mathcal{B})$, so $(x,y) \in Q_2(\mathcal{B})$. It follows that that there is a semipath in $\mathcal{B}$ of the form $(y_0, v_1, y_1, \ldots, y_{m-1}, v_m, y_m)$, where $y_0 = x$, $y_m = y$, and $v = v_1 \cdots v_m \in L(Q_2)$. It can be shown that v folds onto u. Thus, $u \in \mathit{fold}(L(Q_2))$.

Conversely, suppose that $L(Q_1) \subseteq \mathit{fold}(L(Q_2))$. If $(x,y) \in Q_1(\mathcal{B})$, then there is a semipath in $\mathcal{B}$ of the form $(y_0, u_1, y_1, \ldots, y_{m-1}, u_m, y_m)$, where $y_0 = x$, $y_m = y$, and $u = u_1 \cdots u_m \in L(Q_1)$. It follows that there is a word $v \in L(Q_2)$) such that $v \rightsquigarrow u$. But it can be then shown that $(x,y) \in Q_2(\mathcal{B})$. □

Observe that, in the proof of the above lemma we exploit the fact that, if we consider a database $\mathcal{B}_w$ constituted by a single semipath

$$x \cdot \xrightarrow{w} \cdot y$$

then $(x,y) \in Q(\mathcal{B}_w)$ if and only if $w \in \mathit{fold}(Q)$. We are going to exploit this property also later.

We now show that if A is a 1NFA, then $\mathit{fold}(L(A))$ can be represented by a "small" *two-way nondeterministic finite-state automaton* (2NFA). Recall that a 1NFA 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 *Let A be an n-state 1NFA over $\Sigma^{\pm}$. Then there is a 2NFA for $fold(L(A))$ with $n \cdot (|\Sigma^{\pm}| + 1)$ states.*

Proof. Let $A = (\Sigma^{\pm}, S, S_0, \rho, F)$ be a 1NFA. We describe a 2NFA $A' = (\Sigma^{\pm}, S \cup (S \times \Sigma), S_0, \rho', F)$ for $fold(L(A))$. Note that A''s state set contains for each state $s \in S$ additional copies of s, tagged with all the letters in Σ. The initial-state set and final-state set are unchanged. The transition function ρ' is defined as follows:

- $\rho'((s,a),b) = \begin{cases} \{(s,0)\} & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases}$

  Intuitively, when a state is tagged with a letter a, we just check that the next letter it reads is indeed a.

- $\rho'(s,a) = \rho(s,a) \times \{1\} \cup \{((s',b),-1) \ : \ s' \in \rho(s,b^-), b \in \Sigma^{\pm}\}$.

  Intuitively, A' can emulate A forward or backwards. When it emulates it backwards, it guesses a letter and uses it to tag the state in order to check it later. It can be shown that $L(A') = fold(L(A))$. □

According to Language-Theoretic Lemma 2, to check that $Q_1 \sqsubseteq Q_2$, we need to check $L(Q_1) \subseteq fold(L(Q_2))$. This requires the ability to complement 2NFAs. If we use the standard approach, we'd first convert the 2NFA to a 1NFA 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 [45] *Let A be a 2NFA over Σ. There is a 1NFA A^c such that*

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

Proof. Let $A = (\Sigma, S, S_0, \rho, F)$. The construction is based on the following observation: $u_1 \cdots u_n \notin L(A)$ iff there is a sequence $T_0, T_1, \ldots, T_n$ of subsets of S such that

1. $S_0 \subseteq T_0$ and $T_n \cap F = \emptyset$.
2. If $s \in T_i$ and $(t,1) \in \rho(s, u_{i+1})$, then $t \in T_{i+1}$, for $0 \leq i < n$.
3. If $s \in T_i$ and $(t,0) \in \rho(s, u_{i+1})$, then $t \in T_i$, for $0 \leq i < n$.
4. If $s \in T_i$ and $(t,-1) \in \rho(s, u_{i+1})$, then $t \in T_{i-1}$, for $0 < i < n$.

We call this sequence a *counterexample sequence*. The 1NFA A^c simply guesses a counterexample witness.

A^c is the automaton $(\Sigma, Q, Q_0, \delta, G)$. The state set Q is $2^S \cup (2^S)^2$, i.e., sets of states and pairs of sets of states. The start-state set Q_0 is $\{T : S_0 \subseteq T \subseteq S\}$, i.e., the collection of state sets that contain S_0. The final-state set G is $\{T : T \cap F = \emptyset\} \cup \{(T,U) : U \cap F = \emptyset\}$, i.e., the collection of sets that do not intersect F and pair of sets where the second component does not intersect F.

It remains to define the transition function δ. We have $(T,U) \in \delta(T,a)$ if the following holds:

1. If $s \in T$ and $(t,0) \in \rho(s,a)$, then $t \in T$, and
2. if $s \in T$ and $(t,1) \in \rho(s,a)$, then $t \in U$.

We have $(U,V) \in \delta((T,U),a)$ if the following holds:

1. If $s \in U$ and $(t,-1) \in \rho(s,a)$, then $t \in T$,
2. If $s \in U$ and $(t,0) \in \rho(s,a)$, then $t \in U$, and
3. if $s \in U$ and $(t,1) \in \rho(s,a)$, then $t \in V$.

It is easy to verify that A^c accepts a word $u = u_1 \cdots u_n$ if and only if there exists a counterexample sequence, which means that u is not accepted by 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 1NFAs 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 $L(A'_2) = fold(L(A_2))$. The step involves a polynomial blow-up (Lemma 3).
3. Construct a 1NFA A^c_2 such that $L(A^c_2) = (\Sigma^{\pm})^* - L(A'_2)$. This step involves an exponential blow-up (Lemma 4).
4. Construct a 1NFA $A = A_1 \times A^c_2$ such that $L(A) = L(Q_1) - fold(L(Q_2))$. This requires taking the product of A_1 and A^c_2, 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. □

4 View-based Query Answering

In this section we address the problem of view-based query answering by making use of a strong connection between view-based query answering and the constraint-satisfaction problem.

A *constraint-satisfaction problem (CSP)* is traditionally defined in terms of a set of variables, a set of values, and a set of constraints, and asks whether there is an assignment of the variables with the values that satisfies the constraints. A characterization of CSP can be given in terms of homomorphisms between relational structures [22]. Here we consider relational structures whose relations are of arbitrary arity.

A *homomorphism* $h : A \to B$ between two relational structures A and B over the same alphabet is a mapping $h : \Delta^A \to \Delta^B$ such that, if $(c_1, \ldots, c_n) \in r(A)$, then $(h(c_1), \ldots, h(c_n)) \in r(B)$, for every relation symbol r in the alphabet. Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of structures. The *(uniform) constraint-satisfaction problem* $CSP(\mathcal{A}, \mathcal{B})$ is the following decision problem: given a structure $A \in \mathcal{A}$ and a structure $B \in \mathcal{B}$ over the same alphabet, is there a homomorphism $h : A \to B$? When $\mathcal{B}$ consists of a single structure B and $\mathcal{A}$ is the set of all structures over the alphabet of B, we get the so-called *non-uniform* constraint-satisfaction problem, denoted by $CSP(B)$, where B is fixed and the input is just a structure $A \in \mathcal{A}$. As usual, we use $CSP(B)$ also to denote the set of structures A such that there is a homomorphism from A to B.

From the very definition of CSP it follows directly that every $CSP(\mathcal{A}, \mathcal{B})$ problem is in NP. In particular, a non-uniform constraint-satisfaction problem $CSP(B)$, where B is a fixed structure, is still in NP, i.e., checking whether $A \in CSP(B)$ is in NP in the size of A. In general, the actual complexity of $CSP(B)$ depends on B, and there are structures B for which $CSP(B)$ is PTIME and structures B for which $CSP(B)$ is NP-complete. For example, $CSP(K_2)$, is the problem of *2-colorability*, which is in PTIME, while $CSP(K_3)$ is the problem of *3-colorability*, which is NP-complete (K_n is the complete graph with n nodes).

A tight relationship between non-uniform CSP and view-based query answering for RPQs and 2RPQs has been developed in [12, 17]. Such a relationship is based on the notions of constraint template, associated to the query and view definitions, and constraints instance, associated to the view extension. We illustrate such a relationship for 2RPQs.

Given a 2RPQ Q and a set $\mathcal{V}$ of 2RPQ views, the *constraint template* $CT_{Q,\mathcal{V}}$ of Q wrt $\mathcal{V}$ is the relational structure C defined as follows.

- The alphabet of C is $\mathcal{V} \cup \{U_i, U_f\}$.
- Let $A_Q = (\Sigma, S, S_0, \rho, F)$ be a 1NFA for Q. The structure $C = (\Delta^C, \cdot^C)$ is given by:
  - $\Delta^C = 2^S$;
  - $\sigma \in U_i(C)$ iff $S_0 \subseteq \sigma$;
  - $\sigma \in U_f(C)$ iff $\sigma \cap F = \emptyset$;
  - for a view $V \in \mathcal{V}$, we have that $(\sigma_1, \sigma_2) \in V^C$ iff there exists a word $q_1 \cdots q_k \in L(V^\Sigma)$ and a sequence $T_0, \ldots, T_k$ of subsets of S such that the following hold:
    1. $T_0 = \sigma_1$ and $T_k = \sigma_2$,
    2. if $s \in T_i$ and $t \in \rho(s, q_{i+1})$ then $t \in T_{i+1}$, for $0 \le i < k$, and
    3. if $s \in T_i$ and $t \in \rho(s, q_i^-)$ then $t \in T_{i-1}$, for $0 < i \le k$.

Intuitively, the constraint template represents for each view V how the states of A_Q (i.e., of the 1NFA for Q) change when we follow database edges according to what specified by words in $L(V^\Sigma)$. Specifically, the last condition above corresponds to saying that a pair of sets of states (σ_1, σ_2) is in $V(C)$ if and only if there is some word w in $L(V^\Sigma)$ such that the following holds: if we start from a state in σ_1 on the left edge of w and move back and forth on w according to the transitions in A_Q, then, if we end up at the left edge of w we can be only in states in σ_1, and if we end up at the right edge of w we can be only in states in σ_2; similarly, if we start from a state in σ_2 on the right edge of w. Moreover, the sets of states in $U_i(C)$ contain all initial states of A_Q, while the sets of states in $U_f(C)$ do not contain any final state of A_Q. This takes into account that we aim at characterizing counterexamples to view-based containment, and hence we are interested in not getting to a final state of A_Q, regardless of the initial state from which we start and how we follow transitions.

Observe that, to check the existence of a word $q_1 \cdots q_k \in L(V^\Sigma)$ and of a sequence $T_0, \ldots, T_k$ of subsets of S such that conditions 1–3 above are satisfied, we can resort to a construction analogous to the one in the proof of Lemma 4. Hence such a check can be done in polynomial space in the size of Q, and in fact in nondeterministic logarithmic space in the size of V^Σ.

Given a $\mathcal{V}$-extension $\mathcal{E}$ and a pair of objects c and d, the *constraint instance* $\mathcal{E}^{c,d}$ for $CT_{Q,\mathcal{V}}$ is the structure $I = (\Delta^I, \cdot^I)$ over the alphabet $\mathcal{V} \cup \{U_i, U_f\}$ defined as follows:

- $\Delta^I = \Delta^{\mathcal{E}} \cup \{c, d\}$;
- $V(I) = V(\mathcal{E})$, for each $V \in \mathcal{V}$;
- $U_i(I) = \{c\}$, and $U_f(I) = \{d\}$.

The following theorem provides the characterization of view-based query answering in terms of CSP.

Theorem 6 *Let Q be a 2RPQ, $\mathcal{V}$ a set of 2RPQ views, $\mathcal{E}$ a $\mathcal{V}$-extension, and c, d a pair of objects. Then, $(c, d) \notin cert_{Q,\mathcal{V}}(\mathcal{E})$ if and only if $\mathcal{E}^{c,d} \in CSP(CT_{Q,\mathcal{V}})$ (i.e., there is a homomorphism from $\mathcal{E}^{c,d}$ to $CT_{Q,\mathcal{V}}$).*

Proof. "$\Leftarrow$" Given a homomorphism h from $\mathcal{E}^{c,d}$ to $CT_{Q,\mathcal{V}}$, we construct a database $\mathcal{B}$ which is a counterexample to (c, d) being a certain answer. In other words, $\mathcal{B}$ is such that $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$ but $(c, d) \notin Q(\mathcal{B})$. To construct $\mathcal{B}$ we proceed as follows. For every view V and every pair $(a, b) \in V(\mathcal{E})$ we choose a word $w \in L(V^\Sigma)$, satisfying the conditions for $(h(a), h(b)) \in V(CT_{Q,\mathcal{V}})$ (cf. last item of the construction of the constraint template), and we introduce in $\mathcal{B}$ a simple semipath

$$a\cdot \xrightarrow{\quad w \quad} \cdot b$$

where the intermediate objects are all new and pairwise distinct. By contradiction, suppose that $(c, d) \in Q(\mathcal{B})$, i.e., there is a (not necessarily simple) semipath

$$a_0\cdot \xrightarrow{\quad \ell_1 \quad} \cdot a_1 \cdots a_{m-1}\cdot \xrightarrow{\quad \ell_m \quad} \cdot a_m$$

from $c = a_0$ to $d = a_m$ in $\mathcal{B}$ with $\ell_1 \cdots \ell_m \in L(Q)$, and where $a_0, \ldots, a_m$ are the only objects in the view extension on such a path. By construction of $\mathcal{B}$, each simple semithpath ℓ_i corresponds to a navigation over one of the words w used in the construction of B. The semipath ℓ_i may navigate from the left side of w to its right side or vice versa, or it may start and end at the same side, when $a_{i-1} = a_i$. Now, consider a sequence $\delta = (s_0, \ldots, s_m)$ of states of A_Q such that $s_0 \in S_0$, $s_{i+1} \in \rho(s_i, \ell_{i+1})$ [2] for $0 \leq i < m$, and $s_m \in F$. If $s_i \in h(a_i)$ and $a_{i+1} \neq a_i$, since we have reached s_{i+1} by navigating on a word w satisfying the conditions for $(h(a_i), h(a_{i+1})) \in V(CT_{Q,\mathcal{V}})$ (or $(h(a_{i+1}), h(a_i)) \in V(CT_{Q,\mathcal{V}})$, if w is traversed from right to left) for some view V, we must have that $s_{i+1} \in h(a_{i+1})$. Similarly when $a_{i+1} = a_i$. Now we have that $s_0 \in h(c)$, and hence, by induction on m, we have that $s_m \in h(d)$. But since $h(d) \in U_f$, we have that $s_m \notin F$. This leads to a contradiction.

"$\Rightarrow$" Given a database $\mathcal{B}$ such that $\mathcal{E} \subseteq \mathcal{V}(\mathcal{B})$ and two objects c, d such that $(c, d) \notin Q(\mathcal{B})$, we build a homomorphism from $\mathcal{E}^{c,d}$ to $CT_{Q,\mathcal{V}}$. To do so, we first build a mapping $h' : \Delta^\mathcal{B} \to 2^S$ as follows: initially h' assigns the empty set to each element of $\mathcal{E}^{c,d}$, except for $h'(c) = S_0$; then we repeat the following until h' does not change any more: if $(x, y) \in r(\mathcal{B})$ and $s \in h'(x)$ then add $\rho(s, r)$ to $h'(y)$, and if $(x, y) \in r(\mathcal{B})$ and $s \in h'(y)$ then add $\rho(s, r^-)$ to $h'(x)$. Note that, since $(c, d) \notin Q(\mathcal{B})$, we have that $h'(d) \cap F = \emptyset$. Projecting h' on $\Delta^\mathcal{E}$ we obtain the homomorphism we were looking for. $\square$

With respect to computational complexity, we provide an analysis distinguishing the different sources of complexity [44]. In particular, we consider separately *data complexity*, i.e., the complexity wrt the data in the view extension, *expression complexity*, i.e., the complexity wrt the query and the view definitions, and *combined complexity*, i.e., the complexity wrt view extensions, query, and view definitions.

[2] With a slight abuse of notation we denote with $\rho(s_i, \ell_{i+1})$ the set of states reached from s_i by following the word labeling the semipath ℓ_{i+1}.

We start by noting that the constraint instance $\mathcal{E}^{c,d}$ has polynomial size in the view extension $\mathcal{E}$ and does not depend on Q and on the view definitions $\mathcal{V}^\Sigma$. The constraint template $CT_{Q,\mathcal{V}}$, instead, is in general exponential in the size of Q, polynomial in the size of $\mathcal{V}^\Sigma$, and does not depend on the view extension $\mathcal{E}$. Now, consider that for two structures A and B over the same alphabet, checking whether $A \in CSP(B)$, where B is fixed, is in NP in the size of A. By taking as structure A the constraint instance $\mathcal{E}^{c,d}$ and as structure B the constraint template $CT_{Q,\mathcal{V}}$, by Theorem 6, we get that the data complexity of view-based query answering is in coNP. For combined complexity, we can build the constraint instance and guess a mapping h from the constraint instance to the constraint template (without representing the constraint template explicitly). Such a mapping h can be represented using logarithmic space in the size of the constraint template, and hence polynomial space in the size of Q and logarithmic space in the size of the view definitions. Then we can check whether h is a homomorphism, by checking for each view V and for each $(a, b) \in V(\mathcal{E}^{c,d})$, whether $(h(a), h(b)) \in V(CT_{Q,\mathcal{V}})$. As discussed above, this check can be done in nondeterministic logarithmic space in the size of V and in polynomial space in the size of Q.

Theorem 7 *View-based query answering for 2RPQs is in coNP wrt data complexity and in PSPACE wrt combined (and hence expression) complexity.*

The upper bounds established above are tight.

Theorem 8 *View-based query answering for RPQs is coNP-hard wrt data complexity.*

Proof. To show this hardness result we reduce the problem of graph 3-colorability, known to be NP-complete [24], to the problem of checking whether a pair of objects (c, d) is not a certain answer to a fixed query Q with respect to a set of views $\mathcal{V}$ whose definition is fixed. The alphabet is given by $\Sigma = \{R_{xy} \mid x, y \in \{r, g, b\}, x \neq y\} \cup \{S_r, S_g, S_b\} \cup \{F_r, F_g, F_b\}$. Intuitively, R_{xy} denotes a directed edge connecting two nodes of the graph colored respectively x and y, S_x denotes the connection from a fixed starting object c not part of the original graph to a node of the graph colored by x, and F_x denotes the connection from a node colored by x to a fixed final object d not part of the original graph. We introduce three views V_s, V_f, and V_G with definitions:

$$\begin{array}{rcl} V_s^\Sigma & = & S_r + S_g + S_b \\ V_f^\Sigma & = & F_r + F_g + F_b \\ V_G^\Sigma & = & R_{rg} + R_{gr} + R_{rb} + R_{br} + R_{gb} + R_{bg} \end{array}$$

Now consider a graph $G = (N, E)$ to be checked for 3-colorability. From G we define a view extension $\mathcal{E}$ as follows (where c and d are the fixed objects not in N):

$$\begin{array}{rcl} V_s(\mathcal{E}) & = & \{(c, a) \mid a \in N\} \\ V_f(\mathcal{E}) & = & \{(a, d) \mid a \in N\} \\ V_G(\mathcal{E}) & = & \{(a, b), (b, a) \mid (a, b) \in E\} \end{array}$$

Intuitively, V_G represents G given as a symmetric directed graph, while V_s and V_f are used to connect c and d to all nodes of the graph. The query is

$$Q = \sum_{\substack{x,y\in\{r,g,b\}\\ x\neq y}} S_x \cdot F_y + \sum_{\substack{x,y,w,z\in\{r,g,b\}\\ x\neq y \vee w\neq z}} S_x \cdot R_{y,w} \cdot F_z$$

Intuitively, Q describes the existence of an error in assigning colors to the nodes of the graph. Indeed, if the graph G is 3-colorable, then we can construct a database $\mathcal{B}$ containing as objects the nodes of the graph plus c and d. By taking as extension of S_x (respectively F_x) all pairs (c, a) (respectively (a, d)) such that the color assigned to a is x, and taking as extension of R_{xy} the pairs (a, b) such that the color assigned to a is x and the color assigned to b is y, we have that $\mathcal{E}$ is sound wrt $\mathcal{B}$ and $(c, d) \notin Q(\mathcal{B})$. Conversely, from a database $\mathcal{B}$ such that $(c, d) \notin Q(\mathcal{B})$, we can directly obtain a 3-coloring of the graph, by assigning to a node a the color x determined by the (necessarily unique) relation S_x (or, equivalently F_x) with $(c, a) \in S_x(\mathcal{B})$ (respectively, $(a, d) \in F_x(\mathcal{B})$). $\square$

In the above proof we made use of union and chaining (i.e., join) in the query and in the views, but did not exploit reflexive-transitive closure. Thus, the coNP lower bound for view-based query answering holds also when queries and views are unions of conjunctive queries [2].

In fact, the above reduction can straightforwardly be generalized to reduce any instance of CSP over directed graphs to view-based query answering [12].

Theorem 9 *View-based query answering for RPQs is PSPACE-hard wrt expression (and hence combined) complexity.*

Proof. By reduction from regular expression universality, known to be PSPACE-complete [24]. We reduce universality of a regular expressions A to answering query $Q = A$ using a single view V with definition $V^\Sigma = \Sigma^*$ and extension $\mathcal{E}$ such that $V(\mathcal{E}) = \{(c, d)\}$. It is easy to verify that $L(\Sigma^*) \subseteq L(A)$ if and only if $(c, d) \in cert_{Q,V}(\mathcal{E})$. $\square$

5 View-Based Query Rewriting

In this section, we address the problem of view-based query rewriting for 2RPQs, i.e., finding the maximal rewriting of a 2RPQ Q under sound 2RPQ views $\mathcal{V}$. We denote the alphabet $\mathcal{V} \cup \{V^- \mid V \in \mathcal{V}\}$ including a new symbol V^- for each V in $\mathcal{V}$ by $\mathcal{V}^\pm$. In the following, we make use of the notion of *expansion* of a word over $\mathcal{V}^\pm$ wrt the definition of the views in $\mathcal{V}$. Given a word $v = v_1 \cdots v_n$ over $\mathcal{V}^\pm$, we denote by $expand_\Sigma(v)$ the set of all the words $w_1 \cdots w_n$ over $\Sigma^\pm$ such that, for $1 \leq i \leq n$, we have that $w_i \in L({v_i}^\Sigma)$. In other words, every word in $expand_\Sigma(v)$ is obtained from v by substituting every symbol v_i appearing in v with one word w_i belonging to the language of the view definition $L({v_i}^\Sigma)$.

Our technique for computing the maximal rewriting of Q under sound views $\mathcal{V}$ is based on characterizing the words over the alphabet $\mathcal{V}^\pm$ that do *not* belong to any rewriting of Q under sound views $\mathcal{V}$. These are the words v over $\mathcal{V}^\pm$ such that there is at least one word in $expand_\Sigma(v)$ that is not in $fold(Q)$ (cf. Section 3), i.e., considering the word as a linear database, the endpoints are not in the answer to Q. In the following we show how to construct a 2NFA that accepts exactly such words. The complement $R_{\mathcal{V},Q}$ of such a 2NFA has the property that it accepts exactly all words that belong to at least one rewriting of Q under sound views $\mathcal{V}$. It follows that $R_{\mathcal{V},Q}$ is the maximal rewriting of Q under sound views $\mathcal{V}$.

The construction of $R_{\mathcal{V},Q}$ is based on the idea of representing in a single word both a word over $\mathcal{V}^\pm$ and its expansion. Specifically, we consider words over $\Sigma^\pm \cup \mathcal{V}^\pm \cup \{\$, :\}$ of the form

$$\$ V_{i_1} : w_{i_1} \$ \cdots \$ V_{i_m} : w_{i_m} \$$$

with $V_{i_j} \in \mathcal{V}^\pm$, and $w_{i_j} \in (\Sigma^\pm)^*$. Using suitable projections of such words, respectively on $\Sigma^\pm$ and on $\mathcal{V}^\pm$, we can check conditions related to Q, view definitions, and possible rewritings.

We construct a 2NFA A_1 that accepts words of the form above such that $w_{i_1} \cdots w_{i_m} \in fold(Q)$ (cf. Lemma 3). Let A_2 be a 1NFA that complements A_1 (cf. Lemma 4). Let A_3 be a 1NFA that accepts a word of the form

$$\$ V_{i_1} : w_{i_1} \$ \cdots \$ V_{i_m} : w_{i_m} \$$$

if and only if for every i_j, the word w_{i_j} is in $L(V_{i_j}^\Sigma)$, i.e., if and only if the word $w_{i_1} \cdots w_{i_m}$ is in $expand_\Sigma(V_{i_1} \cdots V_{i_m})$. Now consider the 1NFA $A_3 \cap A_2$. A word

$$\$ V_{i_1} : w_{i_1} \$ \cdots \$ V_{i_m} : w_{i_m} \$$$

is accepted by this 1NFA if and only if the word $w_{i_1} \cdots w_{i_m}$ is in $expand_\Sigma(V_{i_1} \cdots V_{i_m})$ and is not in $fold(Q)$. Let A_4 accept all words $V_{i_1} \cdots V_{i_m}$ that are projections on the V_{i_j}'s of the words

$$\$ V_{i_1} : w_{i_1} \$ \cdots \$ V_{i_m} : w_{i_m} \$$$

that are accepted by $A_3 \cap A_2$. By construction, A_4 accepts all words $V_{i_1} \cdots V_{i_m}$ such that there is a word in $expand_\Sigma(V_{i_1} \cdots V_{i_m})$ that is not in $fold(Q)$. Finally, let $R_{\mathcal{V},Q}$ be the complement of A_4. Hence, $R_{\mathcal{V},Q}$ accepts all words $V_{i_1} \cdots V_{i_m}$ such that every word in $expand_\Sigma(V_{i_1} \cdots V_{i_m})$ is in $fold(Q)$. It can be shown that $R_{\mathcal{V},Q}$ is a maximal rewriting of Q under sound views $\mathcal{V}$ (see [11, 17]).

With regard to the complexity of the above method, observe that the size of A_1 is polynomial in the size of Q, the size of A_2 is exponential in the size of A_1, and the size of A_3 is polynomial in the size of $\mathcal{V}$. Finally, the size of A_4 is polynomial in the size of A_3 and A_2. Therefore, the size of $R_{\mathcal{V},Q}$ is exponential in the size of A_4, which means double exponential in the size of Q, and exponential in the size of the view definitions. It follows that the problem of view-based query rewriting for 2RPQs is in 2EXPTIME. In [16] it is also shown that the problem of verifying the existence of a nonempty rewriting of an RPQ Q under sound views $\mathcal{V}$ is EXPSPACE-complete, and that there are cases where the smallest maximal rewriting of an RPQ Q is doubly exponential in the size of Q. This implies that the above method for computing the maximal rewriting of a 2RPQ under sound 2RPQ views is essentially optimal.

Once we have computed the maximal rewriting, the problem arises of checking whether such a rewriting indeed provides all certain answers, when evaluated over a view extension. The coNP data complexity result of view-based query answering (cf. Theorem 8) and the fact that a 2RPQ can be evaluated in polynomial time data complexity tells us that in general the maximal rewriting will *not* provide all certain answers. A technique to check whether a specific rewriting provides all certain answers is developed in [17], again exploiting the connection with CSP. Such a technique gives us a NEXPTIME upper bound in the size of the rewriting, which can also shown to be tight [17].

Related to the problem of computing rewritings, is the problem of checking whether a given 2RPQ Q_r over $\mathcal{V}$ is a rewriting (wrt a set $\mathcal{V}$ of views) of a given 2RPQ Q over Σ. For monotone query languages, and hence for RPQs and 2RPQs, Q_r is a rewriting of Q under sound views $\mathcal{V}$ if and only if the query Q_r^{Σ}, obtained from Q_r by replacing each view symbol with the corresponding view definition, is contained in Q [17][3]. Hence, to check whether Q_r is a rewriting of Q under sound views, we can resort to query containment.

Also of interest is checking whether a given rewriting Q_r (e.g., the maximally contained rewriting computed by the above algorithm) is *exact*, i.e., equivalent modulo the views to the original query [17]. By the above observation this amounts to checking whether, after expanding view definitions, Q_r becomes equivalent to the original query. The existence of an exact rewriting can be shown to be 2EXPSPACE-complete, both for RPQs and 2RPQs [16]. Note that exactness of a rewriting is a sufficient (but not necessary) condition for the rewriting to provide all certain answers.

6 Conclusions

In this paper we have presented basic results and techniques concerning query containment and view based query processing for the class of two-way regular-path queries. We believe that, besides the specific results, our methods have the merit of showing the power of two-way automata in reasoning on complex queries. Indeed, the techniques described in this paper can be adapted to reasoning about queries of more general forms. First results in this direction are reported in [10, 14] for the problem of checking containment of conjunctive regular-path queries with inverse.

Although in this paper we concentrated our attention to basic techniques for query containment and view-based query processing, we notice that the problem of reasoning on regular path queries and their variants is currently under investigation in several research projects. For example, containment and answering under constraints have been studied for RPQs in [26] and for subclasses of XPath including restricted forms of RPQs [21].

Also, containment and view-based query processing are not the only reasoning services of interest on 2RPQs. For example, additional inference tasks that have been already considered include view-based containment [17, 37] and losslessness [15].

Acknowledgments This work was supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, by EU projects SEWASIE IST-2001-34825 and INFOMIX IST-2001-33570, and by MIUR Strategic Project "Società dell'informazione" - Subproject SP1.

[3]Notice that, because of this property, for monotone languages, the original definition of rewriting based on expanding view definitions and then checking containment (see, e.g., [33, 16]), actually corresponds to the notion of rewriting under sound views.

References

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

[2] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS'98*, pages 254–265, 1998.

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

[4] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of ACM SIGMOD*, pages 137–148, 1996.

[5] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence among relational expressions. *SIAM J. on Computing*, 8:218–246, 1979.

[6] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML query language – w3c working draft. Technical report, World Wide Web Consortium, Aug. 2003. Available at `http://www.w3.org/TR/xquery`.

[7] P. Buneman. Semistructured data. In *Proc. of PODS'97*, pages 117–121, 1997.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of KR'98*, pages 2–13, 1998.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of PODS'99*, pages 194–204, 1999.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of KR 2000*, pages 176–185, 2000.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of PODS 2000*, pages 58–66, 2000.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of LICS 2000*, pages 361–371, 2000.

[13] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. What is query rewriting? In *Proc. of KRDB 2000*, pages 17–27. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-29/`, 2000.

[14] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query answering and query containment over semistructured data. In *Proc. of DBPL 2001*, 2001.

[15] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Lossless regular views. In *Proc. of PODS 2002*, pages 58–66, 2002.

[16] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. *J. of Computer and System Sciences*, 64(3):443–465, 2002. Extended and revised version of [9].

[17] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query containment. In *Proc. of PODS 2003*, pages 56–67, 2003.

[18] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOC'77*, pages 77–90, 1977.

[19] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of ICDE'95*, 1995.

[20] J. Clark and S. DeRose. XML Path Language (XPath) version 1.0 – W3C recommendation 16 november 1999. Technical report, World Wide Web Consortium, 1999. Available at `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[21] A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *Proc. of DBPL 2001*, 2001.

[22] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction. *SIAM J. on Computing*, 28:57–104, 1999.

[23] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc. of AAAI'99*, pages 67–73. AAAI Press/The MIT Press, 1999.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability — A guide to NP-completeness*. W. H. Freeman and Company, San Francisco (CA, USA), 1979.

[25] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of ICDT'99*, volume 1540 of *LNCS*, pages 332–347. Springer, 1999.

[26] G. Grahne and A. Thomo. Query containment and rewriting using views for regular path queries under constraints. In *Proc. of PODS 2003*, pages 111–122, 2003.

[27] A. Gupta and J. D. Ullman. Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In *Workshop on Deductive Databases (In conjunction with JICSLP)*, page 195, Washington D.C. (USA), 1992.

[28] A. Y. Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(4):40–47, 2000.

[29] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[30] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley Publ. Co., 1979.

[31] A. C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[32] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[33] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of PODS'95*, pages 95–104, 1995.

[34] A. Y. Levy and M.-C. Rousset. Verification of knowledge bases: a unifying logical view. In *Proc. of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems*, Leuven, Belgium, 1997.

[35] A. Y. Levy and Y. Sagiv. Semantic query optimization in Datalog programs. In *Proc. of PODS'95*, pages 163–173, 1995.

[36] 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.

[37] T. D. Millstein, A. Y. Levy, and M. Friedman. Query containment for data integration systems. In *Proc. of PODS 2000*, pages 67–75, 2000.

[38] A. Motro. Panorama: A database system that annotates its answers to queries with their properties. *J. of Intelligent Information Systems*, 7(1), 1996.

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

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

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

[42] J. D. Ullman. Information integration using logical views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer, 1997.

[43] R. van der Meyden. *The Complexity of Querying Indefinite Information*. PhD thesis, Rutgers University, 1992.

[44] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.

[45] M. Y. Vardi. A temporal fixpoint calculus. In *Proc. of POPL'88*, pages 250–259, San Diego (CA, USA), 1988.