View-based Query Processing and Constraint Satisfaction

Diego Calvanese
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
calvanese@dis.uniroma1.it

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
degiacomo@dis.uniroma1.it

Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lenzerini@dis.uniroma1.it

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

Abstract

View-based query processing requires to answer a query posed to a database only on the basis of the information on a set of views, which are again queries over the same database. This problem is relevant in many aspects of database management, and has been addressed by means of two basic approaches, namely, query rewriting and query answering. In the former approach, one tries to compute a rewriting of the query in terms of the views, whereas in the latter, one aims at directly answering the query based on the view extensions. We study view-based query processing for the case of regular-path queries, which are the basic querying mechanisms for the emergent field of semi-structured data. Based on recent results, we first show that a rewriting is in general a co-NP function wrt to the size of view extensions. Hence, the problem arises of characterizing which instances of the problem admit a rewriting that is PTIME. A second contribution of the work is to establish a tight connection between view-based query answering and constraint-satisfaction problems, which allows us to show that the above characterization is going to be difficult. As a third contribution of our work, we present two methods for computing PTIME rewritings of specific forms. The first method, which is based on the established connection with constraint-satisfaction problems, gives us rewritings expressed in Datalog with a fixed number of variables. The second method, based on automata-theoretic techniques, gives us rewritings that are formulated as unions of conjunctive regular-path queries with a fixed number of variables.

1. Introduction

Several recent papers in the literature show that the problem of view-based query processing [45, 2] is relevant in many aspects of database management, including query optimization, data warehousing, data integration, and query answering with incomplete information. Informally speaking, the problem requires answering a query posed to a database only on the basis of the information on a set of views, which are again queries over the same database. In query optimization, the problem is relevant because using the views may speed up query processing. In data integration, the views represent the only information sources accessible to answer a query. A data warehouse can be seen as a set of materialized views, and, therefore, query processing reduces to view-based query answering. Finally, since the views provide partial knowledge on the database, view-based query processing can be seen as a special case query answering with incomplete information.

There are two approaches to view-based query processing, called *query rewriting* and *query answering*, respectively. In the former approach, we are given a query Q and a set of view definitions, and the goal is to reformulate the query into an expression that refers only to the views, and provides the answer to Q. Typically, the rewriting is expressed in the same language used for both the query and the views. In the latter approach, besides Q and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples that are implied by these extensions, i.e., the set of tuples t such that t satisfies Q in all the databases that are consistent with the views.

Notice the difference between the two approaches. In query rewriting, query processing is divided in two steps, where the first re-expresses the query in terms of a given query language over the alphabet of the view names, and the second evaluates the rewriting over the view extensions. In query answering, we do not pose any limit to query processing, and the only goal is to exploit all possible information, in particular the view extensions, to compute the answer to the query.

In the last years a large number of results have been reported for both problems. Query rewriting has been studied under different assumptions on the form of the queries and views in [34, 38, 42, 6, 19, 7, 28, 20, 17, 27]. Rewriting techniques for query optimization are described, for example, in [16, 5, 43, 23, 35, 36]. A comprehensive framework for view-based query answering, as well as several interesting results, are presented in [2, 26]. In [11, 13, 15] view-based query processing has been studied for the case of regular-path queries (RPQs).

In this paper, we address view-based query processing in the context of semistructured data. Semistructured data are modeled by labeled directed graphs, and have been introduced with the aim of capturing data in the web. The main difficulty arising in this context is that languages for querying semistructured data enable expressing RPQs [4, 10, 22]. An RPQ asks for all pairs of nodes in the database connected by a path conforming to a regular expression, and therefore may contain a restricted form of recursion. Note that when the query contains unrestricted recursion, both view-based query rewriting and view-based query answering become undecidable, even when the views are not recursive [19]. To the best of our knowledge, the only known decidability results for view-based query processing in the case where both the query and the views may contain recursion are reported in [11, 13, 15].

In spite of the large amount of work on the subject, the relationship between view-based query rewriting and view-based query answering is not completely clarified yet. The first contribution of our work concerns a study of this relationship. We define a *rewriting* of a query with respect to a set of views as a function that, given the extensions of the views, returns a set of pairs of objects that is contained in the answer set of the query with respect to the views. We call the rewriting that returns exactly such set the *perfect* rewriting of the query wrt the views. Thus, view-based query answering, and evaluating the perfect rewriting over given view extensions, are equivalent problems.

Typically, one is interested in queries that are PTIME functions (in data complexity). Hence, we would like rewritings to be PTIME as well. By exploiting the relationship between view-based query rewriting and view-based query answering, and by using the results in [13], we show that the perfect rewritings are not PTIME in general, assuming P$\neq$NP. Hence, the problem arises of characterizing which instances of query rewriting admit a perfect rewriting that is PTIME. A second contribution of the work is to show that this is going to be difficult. To draw such conclusion, we establish a tight connection between view-based query answering and constraint-satisfaction problems. 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. An elegant characterization of CSP can be given in terms of homomorphisms between relational structures [21]. Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of finite relational structures. The 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 vocabulary, is there a homomorphism $h : A \rightarrow B$? We show that CSP is polynomially reducible to view-based query answering and vice versa. This indicates that there is a close relationship between the problem of characterizing the instances of query rewriting that admit a perfect rewriting that is in PTIME and the problem of characterizing the instances of CSP that are in PTIME. As discussed in [31, 21], the latter problem is a longstanding open problem that appears to be difficult to solve.

This result suggests that one should look for other ways to compute a PTIME rewriting (in general not perfect). Here comes the third contribution of our work. We present two methods for computing PTIME rewritings for RPQs, considering rewritings that can be formulated in expressive but tractable query languages. First, based on the connection between view-based query answering and CSP, we show how to obtain rewritings expressed in Datalog with a fixed number of variables. We show that this rewriting is maximal in a certain sense. In particular, the rewriting obtained is perfect if a perfect rewriting of such form exists. Second, using automata-theoretic techniques we show how to obtain maximal rewritings expressed as unions of conjunctive regular-path queries (CRPQs) with a fixed number of variables.

2. View-based query processing for regular-path queries

We consider a setting in which databases are expressed in terms of edge-labeled graphs, and queries ask for pairs of nodes connected by a specified path. This setting is typical in semistructured data, where all data models share the characteristic that data are organized in a labeled graph, where the nodes represent objects, and the edges represent links between objects [37, 9, 8, 1, 24].

Formally, we consider a *database* as an edge labeled graph $DB = (\mathcal{D}, \mathcal{E})$, where $\mathcal{D}$ is a set of nodes (called the *domain*) that represent the objects of DB, and $\mathcal{E} = \{r_e \mid e \in \Sigma\}$ is a set of binary relations corresponding to the edges of the graph labeled by elements from an alphabet Σ.

Such edges represent links between objects labeled by attribute names. We denote an edge from node x to node y labeled by r, i.e., $(x, y) \in r$, with $x \xrightarrow{r} y$.

As query mechanism we consider *regular-path queries* (RPQs), which are the basic constituents of full-fledged query languages over semistructured data [10, 1, 23, 35, 18]. Such queries denote all the paths corresponding to words of a specified regular language over the alphabet Σ, and hence are expressed by means of regular expressions or finite automata [11]. The *answer set of an RPQ Q over a database DB* is $ans(Q, DB) = \{(x, y) \mid \text{there is a path } x \xrightarrow{r_1} \cdots \xrightarrow{r_n} y \text{ in } DB \text{ s.t. } r_1 \cdots r_n \in L(Q)\}$, where $L(Q)$ is the regular language defined by Q.

Next we introduce the problem of view-based query answering [2, 26, 33, 13]. Consider a database that is accessible only through a set $\mathcal{V} = \{V_1, \ldots, V_k\}$ of views, and suppose we want to answer an RPQ only on the basis of our knowledge on the views. Specifically, associated to each view V_i we have:

- its definition $def(V_i)$ in terms of an RPQ over the alphabet Σ;
- information about its extension in terms of a set $ext(V_i)$ of pairs of objects[1].

We denote $(def(V_1), \ldots, def(V_k))$ by $def(\mathcal{V})$, $(ext(V_1), \ldots, ext(V_k))$ by $ext(\mathcal{V})$, and the set of objects appearing in $ext(\mathcal{V})$ by $\mathcal{D}_\mathcal{V}$.

We say that a database DB is *consistent with the views* $\mathcal{V}$ if $ext(V_i) \subseteq ans(def(V_i), DB)$, for each $V_i \in \mathcal{V}$. The *certain answer set of Q wrt the views $\mathcal{V}$* is the set $cert(Q, \mathcal{V}) \subseteq \mathcal{D}_\mathcal{V} \times \mathcal{D}_\mathcal{V}$ such that $(c, d) \in cert(Q, \mathcal{V})$ if and only if $(c, d) \in ans(Q, DB)$, for every DB that is consistent with $\mathcal{V}$.

The problem of *view-based query answering* is the following: Given

- a set $\mathcal{V}$ of views, their definitions $def(\mathcal{V})$, and extensions $ext(\mathcal{V})$,
- a query Q,
- a pair of objects $c, d \in \mathcal{D}_\mathcal{V}$,

decide whether $(c, d) \in cert(Q, \mathcal{V})$.

The complexity of the problem can be measured in three different ways [46]:

- *Data complexity*: as a function of the size of $ext(\mathcal{V})$.
- *Expression complexity*: as a function of the size of Q and of the expressions in $def(\mathcal{V})$.
- *Combined complexity*: as a function of the size of $ext(\mathcal{V})$, Q, and $def(\mathcal{V})$.

In [13] the following complexity characterization of view-based query answering is given.

Theorem 2.1 ([13]) *View-based query answering for RPQs is co-NP-complete in data complexity and PSPACE-complete in expression and combined complexity.*

The definition of view-based query answering given above reflects two implicit assumptions. (i) The views are *sound*, i.e., from the fact that a pair (a, b) is in $ext(V_i)$ we can conclude that (a, b) is in $ans(def(V_i), DB)$, but not vice-versa. (ii) The domain is *open*, i.e., a database consistent with the views may contain additional objects that do not appear in the view extensions. Other assumptions about the accurateness of the knowledge on the objects of the database and the pairs satisfying the views, have been studied [2, 26, 13].

We now study the relationship between view-based query answering and query rewriting. An instance of *query rewriting* is given by a query Q and a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$. One then tries to generate a new query Q' over the symbols in $\mathcal{V}$ such that Q' approximates the answer to Q, when V_i is interpreted as $ext(V_i)$, for each $V_i \in \mathcal{V}$. Formally, we require $ans(Q', ext(\mathcal{V})) \subseteq cert(Q, \mathcal{V})$. In the context of RPQs, Q and $def(V_1), \ldots, def(V_k)$ are regular expressions over the alphabet Σ, while Q' is a regular expression over the alphabet $\mathcal{V}$.

A solution to the problem of *RPQ rewriting* is described in [11], where an algorithm is given to compute the maximal rewriting wrt to all rewritings that are RPQs. However, such rewriting is in general not maximal if we allow for rewritings that belong to a larger class of queries. The next example shows that we gain already by considering *conjunctive regular-path queries (CRPQs)*, i.e., conjunctive queries whose atoms are RPQs.

Example 2.2 Consider the query $Q = R_1 \cdot R_3 + R_2 \cdot R_4$ and the views V_1, V_2, V_3 with

$$\begin{aligned} def(V_1) &= R_1 \\ def(V_2) &= R_2 \\ def(V_3) &= R_3 + R_4 \end{aligned}$$

The maximal rewriting of Q in terms of V_1, V_2, and V_3 that is an RPQ is empty. Now, consider the CRPQ over $\{V_1, V_2, V_3\}$

$$R(x, y) \leftarrow x\, V_1\, z,\ x\, V_2\, z,\ z\, V_3\, y$$

By expanding V_1, V_2, and V_3 with their definitions we obtain

$$R(x, y) \leftarrow x\, R_1\, z,\ x\, R_2\, z,\ z\, (R_3 + R_4)\, y,$$

[1] We assume that objects are represented by constants, and we adopt the *unique name assumption* [39], i.e., different constants denote different objects and therefore different nodes.

which is contained in Q. Hence R is a CRPQ that is a (nonempty) rewriting of Q. ∎

From a more abstract point of view, we can define a *rewriting of* Q *wrt* $\mathcal{V}$ as a function that, given $ext(\mathcal{V})$, returns a set of pairs of objects that is contained in the certain answer set $cert(Q, \mathcal{V})$. We call the rewriting that returns exactly $cert(Q, \mathcal{V})$ the *perfect rewriting* of Q wrt $\mathcal{V}$. The problem of *view-based query rewriting* is the one of computing a rewriting of Q wrt $\mathcal{V}$. The problem comes in different forms, depending of the properties that we require for the rewriting. In particular:

- It is sometimes interesting to consider rewritings that are expressible in a certain query language, e.g., Datalog.
- It is also interesting to consider rewritings belonging to a certain data complexity class, for example, polynomial time[2]. A rewriting f belongs to a data complexity class $\mathcal{C}$ if the problem of deciding whether a pair of objects (c, d) is in $f(ext(\mathcal{V}))$ is in the class $\mathcal{C}$, where the complexity of the problem is measured with respect to the size of $ext(\mathcal{V})$.
- Finally, it is worthwhile to compute rewritings that are maximal in a certain class. A rewriting f of Q wrt $\mathcal{V}$ is *maximal in a class* $\mathcal{C}$ if, for every rewriting $g \in \mathcal{C}$ of Q wrt $\mathcal{V}$, we have that $g(ext(\mathcal{V})) \subseteq f(ext(\mathcal{V}))$ for every $ext(\mathcal{V})$.

An algorithm for view-based query answering is an algorithm that takes as input a query, a set of view definitions, and a set of view extensions, and determines whether a given pair of objects is in the answer set of the query for every database that is consistent with the views. Hence, if we fix the query Q and the view definitions $def(\mathcal{V})$, we can consider every algorithm for view-based query answering as an algorithm for the recognition problem for the perfect rewriting of Q wrt $\mathcal{V}$. We remind the reader that the *recognition problem* for a query Q is to check whether a certain tuple is in the answer of Q over a given database. This observation establishes a tight connection between view-based query answering [2] and query rewriting [45].

Now, considering that in the present setting view-based query answering is co-NP-complete in data complexity (see Theorem 2.1), we obtain the following result.

Theorem 2.3 *The perfect rewriting of an RPQ wrt RPQ views is a co-NP function. There is an RPQ Q and a set $\mathcal{V}$ of RPQ views such that the rewriting of Q wrt $\mathcal{V}$ is a co-NP-complete function.*

[2]A query belongs to a data complexity class $\mathcal{C}$ if the corresponding recognition problem is in $\mathcal{C}$ wrt data complexity.

Proof. The membership in co-NP follows by using the algorithm for view-based query answering as the rewriting. For the hardness, note that we can obtain a method for view-based query answering as follows: compute the perfect rewriting R of the query wrt the views and evaluate it against the extension of the views. Since view-based query answering is co-NP-hard wrt the size of the view extensions, and computing R does not depend on the view extensions, it follows that evaluating R against the extension of the views is co-NP-hard in general. □

Typically, one is interested in queries that are PTIME functions. Hence, we would like rewritings to be PTIME as well. Unfortunately, by Theorem 2.3, the perfect rewritings are not PTIME in general, assuming P≠NP. Hence it would be interesting to characterize which instances of query rewriting admit a perfect rewriting that is PTIME. Note, however, that finding such instances corresponds to finding those instances of view-based query answering that are PTIME in data complexity. In Section 4 we show that this is going to be difficult, by exhibiting a tight connection between view-based query answering and constraint satisfaction.

3. Constraint-satisfaction problems

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. An elegant characterization of CSP can be given in terms of homomorphisms between relational structures [21].

A *vocabulary* is a set $V = \{R_1, \ldots, R_t\}$ of predicates, each with an associated arity. A *relational structure* $A = (\Delta^A, \cdot^A)$ over V is a *domain* Δ^A together with an *interpretation function* $\cdot^A$ that assigns to each predicate R_i a relation R_i^A of the appropriate arity over Δ^A. A *homomorphism* $h : A \to B$ between two relational structures A and B over the same vocabulary 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 predicate R in the vocabulary.

Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of finite relational 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 vocabulary, is there a homomorphism $h : A \to B$? We denote such instance as CSP(A, B), and if such a homomorphism exists we say that CSP(A, B) is *satisfiable*. We also consider the special case where $\mathcal{B}$ consists of a single relational structure B and $\mathcal{A}$ is the set of all relational structures over the vocabulary of B, and denote it by CSP(B). Such problem is a (special case of) *non-uniform* constraint-satisfaction problem, i.e., with B fixed, the input is just a structure $A \in \mathcal{A}$. In the case where we take the relational structures to be (directed) graphs, CSP corresponds to *directed-graph homomorphism*. Since general CSP is polynomially equivalent to directed-graph homomorphism [21], that is, for each structure B there is a directed graph G_B such that CSP(B) is polynomially equivalent to CSP(G_B), we restrict attention without loss of generality to CSP over directed graphs, unless explicitly stated otherwise.

From the very definition of CSP it follows directly that every CSP($\mathcal{A}, \mathcal{B}$) problem is in NP. In general, the complexity of a non-uniform constraint-satisfaction problem CSP(B) depends on B. For example, CSP(K_2), is the *Two-Colorability Problem*, while CSP(K_3) is the *Three-Colorability Problem* (K_n is the n-node complete graph); the former is in PTIME, while the latter is NP-complete. In some cases, e.g., when the domain of B has at most two elements or when B is an undirected graph, it is known that CSP(B) is either in PTIME or NP-complete [40, 29]. The Dichotomy Conjecture states that this holds for every structure B [21]. (Recall that if PTIME is different than NP then there are problems that are neither in PTIME nor NP-complete [32].) It is an open problem whether the Dichotomy Conjecture holds. A related open question is that of characterizing the structures B for which CSP(B) is in PTIME [21].

4. CSP and view-based query answering

We establish a tight relationship between constraint-satisfaction problems and view-based query answering. We show first that every CSP is polynomially reducible to view-based query answering.

Theorem 4.1 *Let B be a directed graph. There exists an RPQ Q and RPQ views $\mathcal{V}$ with definitions $def(\mathcal{V})$ such that the following holds: for every directed graph A, there are extensions $ext(\mathcal{V})$ and objects c, d such that $(c, d) \notin cert(Q, \mathcal{V})$ if and only if CSP(A, B) is satisfiable.*

Proof. Let $A = (N_A, E_A)$ and $B = (N_B, E_B)$. We define an instance of view-based query answering as follows:

- The alphabet is $\Sigma = \Sigma_N \cup \Sigma_E$, where $\Sigma_N = \{S_x \mid x \in N_B\} \cup \{F_x \mid x \in N_B\}$ and $\Sigma_E = \{R_{x,y} \mid (x, y) \in E_B\}$.

- The set of objects in the view extensions is $\mathcal{D}_\mathcal{V} = N_A \cup \{c, d\}$, where c, d are two symbols not in N_A.

- The views are V_s, V_f, and V_A with

$$\begin{array}{rcl} def(V_s) & = & \sum_{x \in N_B} S_x \\ def(V_f) & = & \sum_{x \in N_B} F_x \\ def(V_A) & = & \sum_{(x,y) \in E_B} R_{x,y} \\ ext(V_s) & = & \{(c, a) \mid a \in N_A\} \\ ext(V_f) & = & \{(a, d) \mid a \in N_A\} \\ ext(V_A) & = & E_A \end{array}$$

  Intuitively, the extension of V_A represents A, while V_s and V_f are used to connect c and d to all nodes of A, using respectively the "start" relations S_x and "final" relations F_x.

- The query is

$$\begin{array}{rccc} Q = & \displaystyle\sum_{x,y \in N_B,\, x \neq y} & S_x{\cdot}F_y & + \\ & \displaystyle\sum_{\substack{x \in N_B \\ y \neq x,\, (y,z) \in E_B}} & S_x{\cdot}R_{y,z}{\cdot}F_z & + \\ & \displaystyle\sum_{\substack{x \in N_B \\ (x,y) \in E_B,\, z \in N_B \setminus \{y\}}} & S_x{\cdot}R_{x,y}{\cdot}F_z & \end{array}$$

We show that there is a homomorphism from A to B if and only if $(c, d) \notin cert(Q, \mathcal{V})$.

"$\Rightarrow$" Let h be such a homomorphism, and consider the database $DB = (\mathcal{D}, \mathcal{E})$, where $\mathcal{D} = \mathcal{D}_\mathcal{V}$ and $\mathcal{E}$ is defined as follows: for each node $a \in N_A$, there are two edges $c \xrightarrow{S_{h(a)}} a$ and $a \xrightarrow{F_{h(a)}} d$, and for each pair of nodes $a, b \in N_A$, there is an edge $a \xrightarrow{r} b$ if and only if $(a, b) \in E_A$ and $r = R_{h(a),h(b)}$. By construction, and since h is a homomorphism, DB is consistent with the views. Suppose towards contradiction that $(c, d) \in ans(Q, DB)$. Clearly, for no pair $x, y \in N_B$, $x \neq y$, we have that $(c, d) \in ans(S_x{\cdot}F_y, DB)$. Hence, there must be two objects $a_1, a_2 \in \mathcal{D}_\mathcal{V}$ such that $c \xrightarrow{r_1} a_1 \xrightarrow{r_2} a_2 \xrightarrow{r_3} d$, and $r_1 r_2 r_3 \in L(Q)$. By construction of DB, $r_1 = S_{h(a_1)}$, $r_2 = R_{h(a_1),h(a_2)}$, and $r_3 = F_{h(a_2)}$. We get a contradiction to $(c, d) \in ans(Q, DB)$.

"$\Leftarrow$" Let $DB = (\mathcal{D}, \mathcal{E})$ be a database consistent with the views and such that $(c, d) \notin ans(Q, DB)$. By the form of the definitions and of the extensions of the views, we have that (i) $\mathcal{D} = \mathcal{D}_\mathcal{V}$, (ii) for each node $a \in N_A$, there are in $\mathcal{E}$ two edges $c \xrightarrow{r_1} a$, and $a \xrightarrow{r_2} d$, with $r_1 = S_x$ and $r_2 = F_y$ for some $x, y \in N_B$, (iii) for each edge $(a, b) \in E_A$ there is in $\mathcal{E}$ an edge $a \xrightarrow{r} b$, with $r = R_{x,y}$ for some $x, y \in N_B$. By the form of $def(V_A)$ we have that, given a database DB' consistent with the views, if for a pair of nodes a, b of DB' we remove from DB' all but one labeled edge connecting a and b, we obtain a database that is still consistent with the views. Hence, we can assume wlog that for each pair of nodes $a, b \in \mathcal{D}$ there is at most one labeled edge in DB

connecting a and b. We define a mapping h from A to B as follows: for each node $a \in N_A$, we take $h(a) = x$, where $x \in N_B$ is determined by the label S_x of the (unique) edge $c \xrightarrow{S_x} a$ in $\mathcal{E}$. We show that $(c,d) \notin ans(Q, DB)$ implies that h is a homomorphism from A to B. Indeed, for each node $b \in N_A$, the edge in $\mathcal{E}$ from b to d must be labeled by $F_{h(b)}$. Moreover, for each edge $(a,b) \in E_A$, we must have that $R_{h(a),h(b)}$ is defined and labels the (unique) edge in $\mathcal{E}$ from a to b. It follows that E_B contains an edge from $h(a)$ to $h(b)$, and therefore h is a homomorphism. $\square$

The reduction in the proof above is polynomial, so we get the following corollary.

Corollary 4.2 *Every uniform CSP is polynomially reducible to view-based query answering. Every non-uniform CSP is polynomially reducible to the recognition problem for perfect rewritings.*

Observe that the difference between view-based query answering and the recognition problem for perfect rewritings is that, in the first case the input includes the query, the view definitions, and the view extensions, while in the latter case one already has the perfect rewriting and wants to check whether a given pair is in the answer. Hence, in the latter case the input is the perfect rewriting and the view extensions.

We show next that the co-NP data complexity results of [13] can be recast as a reduction from view-based query answering to CSP. To this end, given a query Q and a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$, we call the *constraint template* of Q wrt $\mathcal{V}$ the structure B defined as follows. The vocabulary of B is $\mathcal{V} \cup \{U_c, U_d\}$, where symbols in $\mathcal{V}$ denote binary predicates, and U_c and U_d denote unary predicates. Let $A_Q = (\Sigma, S, S_0, \rho, F)$ be a (nondeterministic) automaton for Q. The structure $B = (\Delta^B, \cdot^B)$ is given by:

- $\Delta^B = 2^S$;
- $(\sigma_1, \sigma_2) \in V_i^B$ iff there exists a word $w \in L(def(V_i))$ such that $\rho(\sigma_1, w) \subseteq \sigma_2$;
- $\sigma \in U_c^B$ iff $S_0 \subseteq \sigma$, and $\sigma \in U_d^B$ iff $\sigma \cap F = \emptyset$.

Theorem 4.3 *Let Q be an RPQ and $\mathcal{V}$ a set of RPQ views with definitions $def(\mathcal{V})$. Then the problem of verifying, given $ext(\mathcal{V})$ and objects c, d, whether $(c,d) \notin cert(Q, \mathcal{V})$ is polynomially reducible to CSP(B), where B is the constraint template of Q wrt $\mathcal{V}$.*

Proof. Given the view extensions $ext(\mathcal{V})$ and a pair of objects c, d, we construct the *constraint instance* $A_{\mathcal{V}}^{c,d} = (\Delta^A, \cdot^A)$ of CSP(B) as follows:

- $\Delta^A = \mathcal{D}_{\mathcal{V}}$, where $\mathcal{D}_{\mathcal{V}}$ is the set of objects in the extensions (which includes also c and d);
- $V_i^A = ext(V_i)$;
- $U_c^A = \{c\}$, and $U_d^A = \{d\}$.

We show that $(c,d) \notin cert(Q, \mathcal{V})$ if and only if there is an homomorphism $h : A_{\mathcal{V}}^{c,d} \to B$.

"$\Leftarrow$" Given a homomorphism $h : A_{\mathcal{V}}^{c,d} \to B$, we construct a database DB as follows: for every view V_i and every pair $(a,b) \in ext(V_i)$ we (i) choose a word $w = r_1 \cdots r_n \in L(def(V_i))$ such that $\rho(h(a), w) \subseteq h(b)$ and (ii) introduce in DB a path $a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b$, where $x_1, \ldots, x_{n-1}$ are new objects. DB is consistent with the views by construction and it can be verified that $(c,d) \notin ans(Q, DB)$.

"$\Rightarrow$" Given a database DB that is consistent with the views and such that $(c,d) \notin ans(Q, DB)$, we build a mapping $h' : \mathcal{D} \to 2^S$ by putting each state in S_0 in $h'(c)$ and repeating the following until h' does not change any more: if there is an edge $x \xrightarrow{r} y$ in DB and $s \in h'(x)$, then add $\rho(s, r)$ to $h'(y)$. Projecting h' on $\mathcal{D}_{\mathcal{V}}$ we obtain a homomorphism $h : A_{\mathcal{V}}^{c,d} \to B$. $\square$

As a consequence of this theorem we have the following corollary.

Corollary 4.4 *View-based query answering is polynomially (in data complexity) reducible to uniform CSP. The recognition problem for perfect rewritings is polynomially reducible (in data complexity) to non-uniform CSP.*

Theorems 4.1 and 4.3 exhibit a very strong connection between CSP and view-based query answering.[3] This indicates that there is a close relationship between the problem of characterizing the instances of query rewriting that admit perfect rewriting that is in PTIME and the problem of characterizing the instances of CSP that are in PTIME. As discussed in [31, 21], the latter problem is a longstanding open problem that appears to be difficult to solve.

5. Rewriting via CSP

As we saw in Section 2, we cannot hope to always have perfect rewritings in PTIME. Furthermore, it follows from the results in Section 4 that characterizing when a PTIME perfect rewriting is possible seems to be a rather difficult problem. This motivates to consider rewritings that can be expressed in an expressive but tractable query language. *Datalog*, which is the language of database logic programming, is such a language and has received tremendous amount of attention over the past two decades (cf. [3]).

[3]This is quite different than the connection applied between CSP and query containment in [31], since the connection there applies to *conjunctive queries*. As the proof in the appendix shows, the connection we establish here applies to union of path queries, whose containment problem is decidable in polynomial time.

A Datalog program is a finite set of rules of the form $t_0 \leftarrow t_1, \ldots, t_m$, where each t_i is an atomic formula $R(x_1, \ldots, x_n)$. The relational predicates that occur in the heads of the rules are the *intensional database* predicates (IDBs), while all others are the *extensional database* predicates (EDBs). One of the IDBs is designated as the *goal* of the program. Note that IDBs may occur in the bodies of rules and, thus, a Datalog program is a recursive specification of the IDBs with semantics obtained via least fixed-points of monotone operators (see [44]). Each Datalog program defines a query, which, given a set of EDB predicates, returns the value of the goal predicate. If the goal predicate is 0-ary, then the program is a Boolean query, i.e., it either holds or does not. Note that a Datalog query is computable in polynomial time, since the bottom-up evaluation of the least fixed-point of the program terminates within a polynomial number of steps (in the size of the given EDBs) (see [44]). Thus, expressibility in Datalog is a sufficient condition for tractability of a query. It is easy to see that RPQs are a special case of Datalog queries.

Suppose we are given a query Q and a set $\mathcal{V}$ of views, with definitions $def(\mathcal{V})$. A Datalog query Q' is a *Datalog rewriting* of Q wrt $\mathcal{V}$ if $ans(Q', ext(\mathcal{V})) \subseteq cert(Q, \mathcal{V})$. It is not clear, however, how we can check whether Q' is a Datalog rewriting of Q wrt $\mathcal{V}$. The algorithm in [11] critically depends on the regularity of candidate rewritings, using the fact that containment of regular expressions is decidable. On the other hand, it is known that containment of Datalog queries is undecidable [41]. Somewhat surprisingly the connection between view-based query answering and CSP can be used to obtain Datalog rewritings for RPQs.

Let $\neg$CSP(B) be the class of structures A such that there is no homomorphism $h : A \to B$. Feder and Vardi [21] provided a unifying explanation for the tractability of many non-uniform CSP(B) problems by showing that $\neg$CSP(B) is expressible in Datalog. That is, they show that in many cases in which CSP(B) is tractable there is a boolean Datalog program P such that $P(A)$ holds iff $A \notin$ CSP(B), for every structure A (of the same vocabulary as B.) In fact, they also showed how to obtain *sound* Datalog programs, i.e., Datalog programs P such that if $P(A)$ holds then $A \notin$ CSP(B). A key parameter that shows up in this analysis is the number of variables used. For every positive integer n, let n-Datalog be the collection of all Datalog programs in which the body of every rule has at most n distinct variables and also the head of every rule has at most n variables (the variables of the body may be different from the variables of the head). For example, the query Non-2-Colorability is expressible in 4-Datalog, since it is definable by the goal predicate Q of the following Datalog program, which asserts that a cycle of odd length exists:

$$\begin{aligned} p(x,y) &\leftarrow e(x,y) \\ p(x,y) &\leftarrow p(x,z), e(z,w), e(w,y) \\ q &\leftarrow p(x,x). \end{aligned}$$

The key fact about n-Datalog and CSP is the existence of *canonical* n-Datalog programs.

Theorem 5.1 ([21, 31]) *Let B be a relational structure and let n be a positive integer. There exists an n-Datalog program P^n_B such that P^n_B is the maximal sound n-Datalog program for $\neg$CSP(B), that is, it is sound for $\neg$CSP(B) and it contains every n-Datalog program that is sound for $\neg$CSP(B).*

Note that Theorem 5.1 implies that if $\neg$CSP(B) is expressible in n-Datalog, then it is expressible by P^n_B. The proof of the theorem is constructive. That is, there is an (exponential-time) algorithm that constructs P^n_B, given B and n. We now show how the theorem can be used to derive Datalog query rewritings.

Let Q be an RPQ and let $\mathcal{V}$ be a set of RPQ views with definitions $def(\mathcal{V})$. Construct a constraint template B for Q wrt $\mathcal{V}$ as in Section 4. As shown in the appendix, $(c,d) \notin cert(Q, \mathcal{V})$ if and only if there is an homomorphism $h : A \to B$, where A consists of $ext(\mathcal{V})$ with the addition of unary relations U_c and U_d for the objects c and d, respectively. Given a positive integer n, we construct the Datalog program P^n_B. By Theorem 5.1, if $P^n_B(A)$ holds, then $(c,d) \in cert(Q, \mathcal{V})$. What we want, however, is a Datalog program that computes an answer, rather than check whether a pair of objects is in the answer. That is, we want the program to check that $(c,d) \in cert(Q, \mathcal{V})$ simultaneously for all pairs (c,d). To that end we modify P^n_B in the following way:

1. let s, t be a pair of variables not occurring in P^n_B,
2. replace an IDB atom $R(x_1, \ldots, x_m)$ in P^n_B by $R(x_1, \ldots, x_m, s, t)$ (in particular, the 0-ary goal predicate G becomes $G(s,t)$), and
3. replace atoms $U_c(x_i)$ (resp., $U_d(x_i)$) by $x_i = s$ (resp., $x_i = t$).

(It is well known that equality atoms can be eliminated [44].) Call the resulting Datalog program Q^n_B. We now show that Q^n_B is in some sense a maximal rewriting.

Theorem 5.2 *Let Q be an RPQ and let $\mathcal{V}$ be a set of RPQ views with definitions $def(\mathcal{V})$. Let B be the constraint template of Q wrt $\mathcal{V}$, and let n be a positive integer. Q^n_B is a rewriting of Q wrt $\mathcal{V}$ that contains all n-Datalog rewritings of Q wrt $\mathcal{V}$. In particular if there is an n-Datalog program that is a perfect rewriting of Q wrt $\mathcal{V}$, then Q^n_B is a perfect rewriting of Q wrt $\mathcal{V}$.*

Proof. Let P be an n-Datalog rewriting of Q wrt $\mathcal{V}$. Let G be the (binary) goal predicate of P. (So we can assume that $n > 1$.) Add to P the rule

$$goal \quad \leftarrow \quad G(s,t),\, U_c(s),\, U_d(t)$$

Call the resulting n-Datalog program P'. Given an extension $ext(\mathcal{V})$ and objects c, d, let $A^{c,d}_{\mathcal{V}}$ be the corresponding constraint instance, defined in the proof of Theorem 4.3. Clearly, if $P'(A^{c,d}_{\mathcal{V}})$ holds then $A^{c,d}_{\mathcal{V}} \notin$ CSP(B). Thus, P' is sound for $\neg$CSP(B), so, by Theorem 5.1, it is contained in P^n_B. Suppose now that $(c,d) \in P(ext(\mathcal{V}))$, then we have that $P'(A^{c,d}_{\mathcal{V}})$ holds, so $P^n_B(A^{c,d}_{\mathcal{V}})$ holds. But then we have that $(c,d) \in Q^n_B(ext(\mathcal{V}))$. $\square$

Note that Q^n_B is an $(n+2)$-Datalog program. It is an open question whether we can get a maximal n-Datalog rewriting. Note also that Q^n_B is a perfect rewriting of Q wrt $\mathcal{V}$ if there exists an n-Datalog program that is a perfect rewriting of Q wrt $\mathcal{V}$. We do not know, however, how to check whether Q^n_B is a perfect rewriting of Q wrt $\mathcal{V}$, and we do not even know how to check whether Q^n_B is *exact*, i.e., to check whether Q^n_B is logically equivalent to the query modulo the definitions of the views [12]. In the next section we describe a rewriting technique using a more restricted class of queries for which we do know how to check whether we have obtained an exact rewriting.

6. Rewriting via automata-theoretic techniques

We exploit automata-theoretic techniques to construct rewritings that are expressed as unions of CRPQs (UCRPQs) with a fixed number of variables. In particular we use standard one-way and two-way finite automata (1NFA, 2NFA) [30, 47].

For technical reasons, we start by considering CRPQs with a fixed skeleton. A *skeleton* for a CRPQ is defined by the number n of variables that appear in the body of the query and for each pair of variables, by the number of conjuncts that involve that pair. Without loss of generality we assume that the variables in the body of the query are $x_1, \ldots, x_n$, and that x_1 and x_2 are the distinguished variables that appear in the head of the CRPQ. Hence we can denote a skeleton by a pair $S = (n, \{(y_1,y_2), \ldots, (y_{2s-1}, y_{2s})\})$, where the second component is a multiset of pairs of variables that range over $x_1, \ldots, x_n$.

A *CRPQ Q with a skeleton*

$$S = (n, \{(y_1,y_2), \ldots, (y_{2s-1}, y_{2s})\})$$

is an CRPQ of the form

$$Q(x_1, x_2) \;\leftarrow\; y_1\, E_1\, y_2,\; \ldots,\; y_{2s-1}\, E_s\, y_{2s}$$

i.e., the body of Q contains one conjunct $y_{2i-1}\, E_i\, y_{2i}$ for each pair (y_{2i-1}, y_{2i}) in the multiset of S. A UCRPQ with skeleton S is a union of CRPQs with skeleton S.

We provide now a method that, given an RPQ Q over the alphabet Σ, a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$, and a skeleton $S = (n, \{(y_1,y_2), \ldots, (y_{2k-1}, y_{2k})\})$, computes an UCRPQ with skeleton S, that is the maximal rewriting of Q wrt $\mathcal{V}$ among the UCRPQs with skeleton S. The method is based on characterizing the counterexamples to the containment of the expansion of a rewriting in Q by means of words of a special form. In particular, we consider words over the alphabet $\Sigma \cup \mathcal{V} \cup \{x_1, \ldots, x_n\} \cup \{\$, \#, :\}$ of the form

$$\$ y_1 \ell_1 y_2 \$ \cdots \$ y_{2s-1} \ell_s y_{2s} \$ \qquad (1)$$

where each ℓ_i has the form

$$\# V_{h_{i,1}} {:} e_{i,1} \# \cdots \# V_{h_{i,m_i}} {:} e_{i,m_i} \#$$

with each $V_{h_{i,j}} \in \mathcal{V}$ and each $e_{i,j} \in \Sigma^*$. Such a word represents a database containing nodes $x_1, \ldots, x_n$, constituting of s paths that are node and edge disjoint (i.e., only start and end nodes can be shared between different paths), such that y_{2i-1} is connected to y_{2i} by a path labeled by $e_{i,1} \cdots e_{i,m_i}$. Moreover the word associates the view $V_{h_{i,j}}$ to the fragment $e_{i,j}$ of the path.

To construct the rewriting we first construct an automaton $A^S_{Q,\mathcal{V}}$ as follows:

1. Construct a 2NFA A_1 that accepts words of the form (1) iff Q, when evaluated on the database represented by the word, is not empty. The construction of A_1 exploits the ability of 2NFA to jump from one occurrence of a symbol x_i to any other occurrence of x_i in the word (as in [14]).

2. Construct a 1NFA A_2 that complements A_1. The automaton A_2 accepts a word of the form (1) iff Q, when evaluated on the database represented by the word, is empty.

3. Construct a 1NFA A_3 that accepts exactly the words of the form (1) such that $e_{i,j} \in L(def(V_{h_{i,j}}))$, for every i, j. The automaton A_3 accepts a word of the form (1) iff each path fragment $e_{i,j}$ is in the language of the view associated to it.

4. Construct the 1NFA $A_2 \cap A_3$. Such an automaton accepts a word of the form (1) iff each path fragment $e_{i,j}$ is in the language of the view associated to it, and the query evaluated over the database corresponding to the word is empty.

5. Construct the 1NFA A_4 that accepts the projections on $\mathcal{V} \cup \{x_1, \ldots, x_n\} \cup \{\$\}$ of the words accepted by $A_2 \cap A_3$, The automaton A_4 accepts a word of the

form $\$\cdot y_1\cdot\mathcal{V}^*\cdot y_2\cdot\$\cdots\$\cdot y_{2s-1}\cdot\mathcal{V}^*\cdot y_{2s}\cdot\$$ if such a word represents view extensions for which there exists a database consistent with the views in which the query is empty.

6. Finally, construct the automaton $A^S_{Q,\mathcal{V}}$ by intersecting the complement of A_4 with an 1NFA that accepts $\$\cdot y_1\cdot\mathcal{V}^*\cdot y_2\cdot\$\cdots\$\cdot y_{2s-1}\cdot\mathcal{V}^*\cdot y_{2s}\cdot\$$. The automaton $A^S_{Q,\mathcal{V}}$ accepts words that represent view extensions for which in all databases consistent with the views the query is empty.

Now, observe that each word accepted by $A^S_{Q,\mathcal{V}}$ can be viewed as a CRPQ with skeleton S of a simple form, where each RPQ is simply a concatenation of symbols in $\mathcal{V}$, we call such CRPQs *simple*. Hence the language accepted by $A^S_{Q,\mathcal{V}}$ denotes a possibly infinite union of simple CRPQs. We show now that we can represent such a possibly infinite union by a finite union of CRPQs.

To this end we construct a new automaton $F^S_{Q,\mathcal{V}}$ with the same set of states as $A^S_{Q,\mathcal{V}}$, same initial and final states, and over an alphabet formed by \$ and symbols that are regular languages $x_i\cdot E\cdot x_j$, with x_i, x_j variables and $E \subseteq \mathcal{V}^*$. The transitions labeled by \$ are as in $A^S_{Q,\mathcal{V}}$, while the other transitions are obtained as follows. For each pair of states s_1, s_2 and each pair of variables x_i, x_j, let $x_i\cdot E\cdot x_j$ be the intersection of $x_i\cdot\mathcal{V}^*\cdot x_j$ with the language accepted by the automaton obtained from $A^S_{Q,\mathcal{V}}$ by changing the initial states to $\{s_1\}$ and the final states to $\{s_2\}$. When such intersection is not empty, then $F^S_{Q,\mathcal{V}}$ contains the transition $(s_1, x_i\cdot E\cdot x_j, s_2)$. Since $A^S_{Q,\mathcal{V}}$ accepts only words that correspond to simple CRPQs with fixed skeleton S, $F^S_{Q,\mathcal{V}}$ accepts only words with a number of symbols of the form $x_i\cdot E\cdot x_j$ equal to the number s of conjuncts determined by the skeleton. Hence the language accepted by $F^S_{Q,\mathcal{V}}$ is finite, and corresponds to a finite union $R^S_{Q,\mathcal{V}}$ of CRPQs, each obtained directly from a word accepted by $F^S_{Q,\mathcal{V}}$.

Theorem 6.1 *$R^S_{Q,\mathcal{V}}$ is the maximal rewriting of Q wrt $\mathcal{V}$ among the rewritings that correspond to (possibly infinite) unions of CRPQs with skeleton S.*

Proof. It suffices to focus on words accepted by $A^S_{Q,\mathcal{V}}$. We prove that every word accepted by $A^S_{Q,\mathcal{V}}$ represents a simple CRPQ such that replacing each $V \in \mathcal{V}$ by $def(V)$ results in a CRPQ Q' contained in Q. It is possible to show that to check that Q' is contained in Q it is sufficient to verify that for each database DB that is canonical for Q', $Q(DB)$ contains the pair (x_1, x_2). A database DB is *canonical* for Q' if: (i) DB constitutes of s paths, one for each conjunct of Q', which are node and edge disjoint; (ii) for $i \in \{1, \ldots, s\}$, the path associated to the conjunct $y_{2i-1}\, E_i\, y_{2i}$ connects the node y_{2i-1} to the node y_{2i} and is labeled by a word in $L(E_i)$. Observe that databases canonical for Q' can be represented by words in $\$\cdot y_1\cdot\Sigma^*\cdot y_2\cdot\$\cdots\$\cdot y_{2s-1}\cdot\Sigma^*\cdot y_{2s}\cdot\$$.

Now consider a word w that is accepted by $A^S_{Q,\mathcal{V}}$. Since w is not accepted by A_4, it follows that each word w' that expands w is either not accepted by A_2 or not accepted by A_3. If w' is not accepted by A_3 then either it does not correspond to a canonical database (and hence needs not to be considered), or it corresponds to a canonical database DB in which subwords in Σ^* are not well annotated with view symbols. In the latter case there exists another word w'' that expands w that corresponds to DB and is well annotated. If w' corresponds to a well annotated canonical database then it is not accepted by A_2. This means that it is accepted by A_1, and hence represents a canonical database on which Q does yield (x_1, x_2).

To see that $\mathcal{A}^S_{Q,\mathcal{V}}$ is maximal among the rewritings that correspond to unions of CRPQs with skeleton S, assume there is a rewriting R containing a simple CRPQ with skeleton S that corresponds to a word w not accepted by $\mathcal{A}^S_{Q,\mathcal{V}}$. Then w is accepted by A_4 and hence there is an expansion w' of w accepted both by A_2 and by A_3. w' corresponds to a canonical database of the CRPQ Q' obtained by replacing each $V \in \mathcal{V}$ by $def(V)$, on which Q does not yield (x_1, x_2), hence contradicting the fact that R is a rewriting. □

In the case where we do not fix the skeleton of the rewriting, but only the number n of variables, the construction above of $F^S_{Q,\mathcal{V}}$ carries through almost unchanged. A *CRPQ Q with n variables* is a CRPQ of the form $Q(x_1, x_2) \leftarrow y_1\, E_1\, y_2,\ \ldots,\ y_{2m-1}\, E_m\, y_{2m}$, where m is any integer and $y_1, \ldots, y_{2m}$ range over $\{x_1, \ldots, x_n\}$. Note that the body of Q may contain an arbitrary number of conjuncts $y_{2h-1}\, E_h\, y_{2h}$ for each pair (x_i, x_j) $1 \leq i, j \leq n$. An UCRPQ with n variables is a union of CRPQs with n variables. To represent such queries we need to consider, instead of words of the form (1), words with an unlimited number of subwords of the form $x_i \ell x_j$. We proceed with steps (1)–(6) as before with this modification. Let us call the resulting automaton $A^n_{Q,\mathcal{V}}$. Consider an accepting run of $A^n_{Q,\mathcal{V}}$ on such a word. Each subword of the form $x_i \ell x_j$ corresponds to a subrun of $A^n_{Q,\mathcal{V}}$. If $A^n_{Q,\mathcal{V}}$ has at most p states, then a simple pumping argument shows that it suffices to consider words with at most p distinct subwords of the form $x_i \ell x_j$ (this is because by deleting subwords we get a "bigger" query). So we can proceed with the construction of the rewriting as before, with the proviso that we consider only words with at most p subwords of the form $x_i \ell x_j$. We still end up with an UCRPQs, which we call $R^n_{Q,\mathcal{V}}$.

Theorem 6.2 *$R^n_{Q,\mathcal{V}}$ is the maximal rewriting of Q wrt $\mathcal{V}$ among the rewritings that correspond to (possibly infinite) unions of CRPQs with n variables.*

Note that an UCRPQ with n variables can be rewritten as an n-Datalog program. Thus, by Theorem 5.2, $R^n_{Q,\mathcal{V}}$ is contained in Q_B^n. However, a nice property of $R_{Q,\mathcal{V}}^n$ is that we know how to test whether it is an exact rewriting. Indeed, by using techniques similar to those in [14], we can check effectively if an UCRPQ rewriting is exact. Unfortunately, as for the case of Q_B^n, we do not know how to check whether $R_{Q,\mathcal{V}}^n$ is a perfect rewriting. Observe also that it is easier to evaluate UCRPQs than Datalog, since the data complexity of UCRPQs is NLOGSPACE, while that of Datalog is PTIME.

7. Conclusions

We have studied view-based query processing in semistructured data. We have set up a framework that clarifies the relationships between the two approaches to the problem, namely, query rewriting and query answering. Based on such framework, we have first shown that the perfect rewriting is in general a co-NP function wrt to the size of view extensions. We have then turned our attention to the problem of characterizing which instances of query rewriting admit a rewriting that is PTIME. Based on a tight connection between view-based query answering and constraint-satisfaction problems, we have shown that the above characterization is going to be difficult. Finally, we have proposed two methods for computing PTIME rewritings. In the first method, rewritings are expressed in Datalog, whereas in the second method, rewritings are formulated as unions of conjunctive regular-path queries. In both methods the rewritings are parametrized by the number of variables used.

There are several interesting open problems to investigate. One is to study the impact of extending the language of RPQs with the inverse operator, in the line of [15]. It also remains open to devise a method that allows us to compute the maximal PTIME rewriting, independently of the language used to express such a rewriting.

Acknowledgements

The last author was supported in part by NSF grant CCR-9700061.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 1–18, 1997.

[2] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[5] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 137–148, 1996.

[6] F. N. Afrati, M. Gergatsoulis, and T. Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 435–452. Springer-Verlag, 1999.

[7] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, pages 99–108, 1997.

[8] P. Buneman. Semistructured data. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, pages 117–121, 1997.

[9] P. Buneman, S. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 336–350, 1997.

[10] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516, 1996.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting regular expressions in semi-structured data. In *Proc. of ICDT'99 Workshop on Query Processing for Semi-Structured Data and Non-Standard Data Formats*, 1999.

[13] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[14] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2000)*, 2000. To appear.

[15] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 2000)*, 2000. To appear.

[16] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, Taipei (Taiwan), 1995.

[17] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 155–166, 1999.

[18] A. Deutsch, M. F. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. Submission to the World Wide Web Consortium, Aug. 1998. Available at http://www.w3.org/TR/NOTE-xml-ql.

[19] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.

[20] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

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

[22] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 414–425, 1998.

[23] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 14–23, 1998.

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

[25] M. L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, Los Altos, 1987.

[26] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1999.

[27] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 174–184, 1999.

[28] J. Gryz. Query folding with inclusion dependencies. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 126–133, 1998.

[29] P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. of Combinatorial Theory, Series B*, 48:92–110, 1990.

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

[31] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 205–213, 1998.

[32] R. E. Ladner. On the structure of polynomial time reducibility. *J. of the ACM*, 22:155–171, 1975.

[33] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 402–412, 1996.

[34] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[35] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295. Springer-Verlag, 1999.

[36] Y. Papakonstantinou and V. Vassalos. Query rewriting using semistructured views. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999.

[37] D. Quass, A. Rajaraman, I. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*, pages 319–344. Springer-Verlag, 1995.

[38] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, 1995.

[39] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978. Republished in [25].

[40] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of the 10th ACM Sym. on Theory of Computing (STOC'78)*, pages 216–226, 1978.

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

[42] D. Srivastava, S. Dar, H. V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 318–329, 1996.

[43] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for phyisical data independence. *Very Large Database J.*, 5(2):101–118, 1996.

[44] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, Potomac, Maryland, 1988.

[45] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[46] M. Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[47] M. Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, 1989.