Rewriting of Regular Expressions and Regular Path Queries

Diego Calvanese\textsuperscript{1}, Giuseppe De Giacomo\textsuperscript{1}, Maurizio Lenzerini\textsuperscript{1}, Moshe Y. Vardi\textsuperscript{2}

\medskip
\textsuperscript{1} Dipartimento di Informatica e Sistemistica  
Università di Roma “La Sapienza”  
Via Salaria 113, I-00198 Roma, Italy  
lastname@dis.uniroma1.it  
http://www.dis.uniroma1.it/~lastname
\textsuperscript{2} Department of Computer Science  
Rice University, P.O. Box 1892  
Houston, TX 77251-1892, U.S.A.  
vardi@cs.rice.edu  
http://www.cs.rice.edu/~vardi

\abstract
Recent work on semi-structured data has revitalized the interest in path queries, i.e., queries that ask for all pairs of objects in the database that are connected by a path conforming to a certain specification, in particular to a regular expression. Also, in semi-structured data, as well as in data integration, data warehousing, and query optimization, the problem of query rewriting using views is receiving much attention: Given a query and a collection of views, generate a new query which uses the views and provides the answer to the original one.

In this paper we address the problem of query rewriting using views in the context of semi-structured data. We present a method for computing the rewriting of a regular expression \( E \) in terms of other regular expressions. The method computes the exact rewriting (the one that defines the same regular language as \( E \)) if it exists, or the rewriting that defines the maximal language contained in the one defined by \( E \), otherwise. We present a complexity analysis of both the problem and the method, showing that the latter is essentially optimal. Finally, we illustrate how to exploit the method to rewrite regular path queries using views in semi-structured data. The complexity results established for the rewriting of regular expressions apply also to the case of regular path queries.

1 \ (Introduction)

Database research has often shown strong interest in path queries, i.e., queries that ask for all pairs of objects in the database that are connected by a specified path (see for example [CMW87, CM90]). Recent work on semi-structured data has revitalized such interest. Semi-structured data are data whose structure is irregular, partially known, or subject to frequent changes [Abi97]. They are usually formalized in terms of labeled graphs, and capture data as found in many application areas, such as web information systems, digital libraries, and data integration [BDFS97, CACS94, MMM97, QRS+95].

The basic querying mechanism over such graphs is the one that retrieves all pairs of nodes connected by a path conforming to a given pattern. Since a user may ignore the precise structure of the graph, the mechanism for specifying path patterns should be flexible enough to allow for expressing regular path queries, i.e., queries that provide the specification of the requested paths through a regular language [AQM+97, BDHS96, FFK+98]. For example, the regular path query \((\cdot\cdot(rome+jerusalem)\cdot\cdot\cdot\text{restaurant})\) specifies all the paths having at some point an edge labeled \textit{rome} or \textit{jerusalem}, followed by any number of other edges and by an edge labeled with a restaurant.

Methods for reasoning about regular path queries have been recently proposed in the literature. In particular, [AV97, BFW98] investigate the decidability of the implication problem for path constraints, which are integrity constraints that are exploited in the optimization of regular path queries. Also, containment of conjunctions of regular path queries has been addressed and proved decidable in [CDGL98, FL98].

In semi-structured data, as well as in data integration, data warehousing, and query optimization, the problem of query rewriting using views is receiving much attention [Ull97, AD98]: Given a query \( Q \) and \( k \) queries \( Q_1, \ldots, Q_k \) associated to the symbols \( q_1, \ldots, q_k \), respectively, generate a new query \( Q' \) over the alphabet \( q_1, \ldots, q_k \) such that, first interpreting each \( q_i \) as the result of \( Q_i \), and then evaluating \( Q' \) on the basis of such interpretation, provides the answer to \( Q \).

Several papers investigate this problem for the case of conjunctive queries (with or without arithmetic comparisons) [LMSS95, RSU95], queries with aggregates [SDJL96, CS99], recursive queries [DG97],
and queries expressed in Description Logics [BLR97]. Rewriting techniques for query optimization are described, for example, in [CKPS95, ACPS96, TS96], and in [FS98, MS99] for the case of path queries in semi-structured data.

None of the above papers provides a method for rewriting regular path queries. Observe that such a method requires a technique for the rewriting of regular expressions, i.e., the problem that, given a regular expression \( E_0 \), and other \( k \) regular expressions \( E_1, \ldots, E_k \), checks whether we can re-express \( E_0 \) by a suitable combination of \( E_1, \ldots, E_k \). As noted in [MS99], such a problem is still open.

In this paper we present the following contributions:

- We describe a method for computing the rewriting of a regular expression \( E_0 \) in terms of other regular expressions. The method computes the exact rewriting (the one that defines the same regular language as \( E_0 \)) if it exists, or the rewriting that defines the maximal language contained in the one defined by \( E_0 \), otherwise.

- We provide a complexity analysis of the problem of rewriting regular expressions. We show that our method computes the rewriting in 2EXP-TIME, and is able to check whether the computed rewriting is exact in 2EXPSPACE. We also show that the problem of checking whether there is a nonempty rewriting is EXPSPACE-complete, and demonstrate that our method for computing the rewriting is essentially optimal. Finally, we show that the problem of verifying the existence of an exact rewriting is 2EXPSPACE-complete.

- We illustrate how to exploit the above mentioned method in order to devise an algorithm for the rewriting of regular path queries for semi-structured databases. The complexity results established for the rewriting of regular expressions apply to the new algorithm as well. Also, we show how to adapt the method in order to compute rewritings with specific properties. In particular, we consider partial rewritings (which are rewritings that, besides \( E_1, \ldots, E_k \), may use also symbols in \( E_0 \)), in the case where an exact one does not exist.

We point out that the results established in this work provide the first decidability results for rewriting recursive queries using recursive views. Indeed, in our context, both the query and the views may contain a form of recursion due to the presence of transitive closure. Observe that the case where the query contains unrestricted recursion has been shown undecidable, even when the views are not recursive [DG97]. More precisely, the authors in [DG97] present a method that computes the maximally contained rewriting of a datalog query in terms of a set of conjunctive queries, and show that it is undecidable to check whether the rewriting is equivalent to the original query.

The paper is organized as follows. Section 2 presents the method for rewriting regular expressions. Section 3 describes the complexity analysis of both the method and the problem. Section 4 illustrates the use of the technique to rewrite path queries for semi-structured databases. Finally, Section 5 describes possible developments of our research.

## 2 Rewriting of regular expressions

In this section, we present a technique for the following problem: Given a regular expression \( E_0 \) and a (finite) set \( \mathcal{E} = \{E_1, \ldots, E_k\} \) of regular expressions over an alphabet \( \Sigma \), re-express, if possible, \( E_0 \) by a suitable combination of \( E_1, \ldots, E_k \).

We assume that associated to \( \mathcal{E} \) we always have an alphabet \( \Sigma_{\mathcal{E}} \) containing exactly one symbol for each regular expression in \( \mathcal{E} \), and we denote the regular expression associated to the symbol \( e \in \Sigma_{\mathcal{E}} \) with \( \text{re}(e) \). Given any language \( \ell \) over \( \Sigma_{\mathcal{E}} \), we denote by \( \exp_{\Sigma_{\mathcal{E}}}(\ell) \) the language over \( \Sigma \) defined as follows

\[
\exp_{\Sigma_{\mathcal{E}}}(\ell) = \bigcup_{e_1, \ldots, e_n \in \ell \ell} \{w_1 \cdots w_n \mid w_i \in L(\text{re}(e_i))\}
\]

where \( L(e) \) is the language defined by the regular expression \( e \).

**Definition 1** Let \( R \) be any formalism for defining a language \( L(R) \) over \( \Sigma_{\mathcal{E}} \). We say that \( R \) is a rewriting of \( E_0 \) wrt \( \mathcal{E} \) if \( \exp_{\Sigma_{\mathcal{E}}}(L(R)) \subseteq L(E_0) \).

We are interested in maximal rewritings, i.e., rewritings that capture in the best possible way the language defined by the original regular expression \( E_0 \).

**Definition 2** A rewriting \( R \) of \( E_0 \) wrt \( \mathcal{E} \) is \( \Sigma\)-maximal if for each rewriting \( R' \) of \( E_0 \) wrt \( \mathcal{E} \) we have that \( \exp_{\Sigma_{\mathcal{E}}}(L(R')) \subseteq \exp_{\Sigma_{\mathcal{E}}}(L(R)) \). A rewriting \( R \) of \( E_0 \) wrt \( \mathcal{E} \) is \( \Sigma_{\mathcal{E}}\)-maximal if for each rewriting \( R' \) of \( E_0 \) wrt \( \mathcal{E} \) we have that \( L(R') \subseteq L(R) \).

Intuitively, when considering \( \Sigma \)-maximal rewritings we look at the languages obtained after substituting each symbol in the rewriting by the corresponding regular expression over \( \Sigma \), whereas when considering \( \Sigma_{\mathcal{E}} \)-maximal rewritings we look at the languages over \( \Sigma_{\mathcal{E}} \). Observe that by definition all \( \Sigma \)-maximal rewritings define the same language (similarly for \( \Sigma_{\mathcal{E}} \)-maximal rewritings), and that not all \( \Sigma \)-maximal rewritings are \( \Sigma_{\mathcal{E}} \)-maximal, as shown by the following example.
Example 1 Let $E_0 = a^*$, $\mathcal{E} = \{a^*\}$, and $\Sigma_\mathcal{E} = \{e\}$, where $re(e) = a^*$. Then both $R_1 = e^*$ and $R_2 = e$ are $\Sigma_\mathcal{E}$-maximal rewritings of $E_0$ wrt $\mathcal{E}$, but $R_1$ is also $\Sigma_\mathcal{E}$-maximal while $R_2$ is not.

However, it turns out that $\Sigma_\mathcal{E}$-maximality is a sufficient condition for $\Sigma$-maximality.

**Theorem 1** Let $R$ be a rewriting of $E_0$ wrt $\mathcal{E}$. If $R$ is $\Sigma_\mathcal{E}$-maximal then it is also $\Sigma$-maximal.

**Proof.** Assume by contradiction that $R$ is a $\Sigma_\mathcal{E}$-maximal rewriting of $E_0$ wrt $\mathcal{E}$ that is not $\Sigma$-maximal. Then there is a rewriting $R'$ of $E_0$ wrt $\mathcal{E}$, a $\Sigma_\mathcal{E}$-word $u' \in L(R')$, and a $\Sigma$-word $w \in L(\exp_2(\{u'\}))$ such that for no $\Sigma_\mathcal{E}$-word $u \in L(R)$, it holds that $w \in L(\exp_2(\{u\}))$. Hence $u' \not\in L(R)$ and $L(R') \not\subseteq L(R)$. Contradiction. □

Given $E_0$ and $\mathcal{E}$, we are interested in deriving a $\Sigma_\mathcal{E}$-maximal rewriting of $E_0$ wrt $\mathcal{E}$. We show that such maximal rewriting always exists. In fact, we provide a method that, given $E_0$ and $\mathcal{E}$, constructs a $\Sigma_\mathcal{E}$-maximal rewriting of $E_0$ wrt $\mathcal{E}$. By Theorem 1 the constructed rewriting is also $\Sigma$-maximal.

The construction takes $E_0$ and $\mathcal{E}$ as input, and returns an automaton $R_{E_0,\mathcal{E}}$ built as follows:

1. Construct a deterministic automaton $A_d = (\Sigma, S, s_0, \rho, F)$ such that $L(A_d) = L(E_0)$.

2. Define the automaton $A' = (\Sigma_\mathcal{E}, S, s_0, \rho', S - F)$, where $s_j \in \rho'(s_i, e)$ if and only if $\exists w \in L(\exp_2(e))$ such that $s_j \in \rho''(s_i, w)$.

3. $R_{E_0,\mathcal{E}} = \overline{A'}$, i.e., the complement of $A'$.

Observe that, if $A'$ accepts a $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n$, then there exist $n$ $\Sigma$-words $w_1, \ldots, w_n$ such that $w_i \in L(\exp_2(e_i))$ for $i = 1, \ldots, n$ and such that the $\Sigma$-word $w_1 \cdots w_n$ is rejected by $A_d$. On the other hand if there exists $n$ $\Sigma$-words $w_1, \ldots, w_n$, such that $w_i \in L(\exp_2(e_i))$, for $i = 1, \ldots, n$, and $w_1 \cdots w_n$ is rejected by $A_d$, then the $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n$ is accepted by $A'$. That is $A'$ accepts a $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n$ if and only if there is a $\Sigma$-word in $\exp_2(e_1 \cdots e_n)$ that is rejected by $A_d$. Hence, $R_{E_0,\mathcal{E}}$, being the complement of $A'$, accepts a $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n$ if and only if all $\Sigma$-words $w = w_1 \cdots w_n$ such that $w_i \in L(\exp_2(e_i))$ for $i = 1, \ldots, n$, are accepted by $A_d$. Hence we can state the following theorem.

**Theorem 2** The automaton $R_{E_0,\mathcal{E}}$ is a $\Sigma_\mathcal{E}$-maximal rewriting of $E_0$ wrt $\mathcal{E}$. 

**Proof.** It is easy to see that by construction $R_{E_0,\mathcal{E}} = \overline{A'}$ is a rewriting of $E_0$ wrt $\mathcal{E}$. We prove by contradiction that it is $\Sigma_\mathcal{E}$-maximal. Let $R$ be a rewriting of $E_0$ wrt $\mathcal{E}$ such that $L(R) \not\subseteq L(\overline{A'})$. Let $e_1 \cdots e_n$ be a $\Sigma_\mathcal{E}$-word such that $e_1 \cdots e_n \in L(R)$ but $e_1 \cdots e_n \not\in L(\overline{A'})$. By definition of rewriting, all $\Sigma$-words $w_1 \cdots w_n$ such that $w_i \in L(\exp_2(e_i))$ for $i = 1, \ldots, n$, are in $L(E_0) = L(A_d)$. On the other hand, since $e_1 \cdots e_n \not\in L(\overline{A'})$, the $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n$ is accepted by $A'$. Thus there is a $\Sigma$-word $w_1 \cdots w_n$, such that $w_i \in L(\exp_2(e_i))$ for $i = 1, \ldots, n$, that is rejected by $A_d$. Contradiction.

Notably, although Definition 1 does not constrain in any way the form of the rewritings, which, a priori, need not even be recursive, Theorem 2 shows that the language over $\Sigma_\mathcal{E}$ (and therefore also the language over $\Sigma$) defined by the $\Sigma_\mathcal{E}$-maximal rewritings is in fact regular (indeed, $\overline{A'}$ is a finite automaton).

We illustrate the algorithm that computes a $\Sigma_\mathcal{E}$-maximal rewriting by means of the following example.

**Example 2** Let $E_0 = a\cdot(b\cdot a + c)^*$, and let $\mathcal{E}$ and $\Sigma$ be such that $re(e_1) = a$, $re(e_2) = a\cdot c^*\cdot b$, and $re(e_3) = c$. The deterministic automaton $A_d$ shown in Figure 1 accepts $L(E_0)$, while $A'$ is the corresponding automaton constructed in Step 2 of the rewriting algorithm. Since $A'$ is deterministic, by simply exchanging final and non-final states we obtain its complement $\overline{A'}$, which is the rewriting $R_{E_0,\mathcal{E}}$.

Next we address the problem of verifying whether the rewriting $R_{E_0,\mathcal{E}}$ captures exactly the language defined by $E_0$.

**Definition 3** A rewriting $R$ of $E_0$ wrt $\mathcal{E}$ is exact if $\exp_2(L(R)) = L(E_0)$.

To verify whether $R_{E_0,\mathcal{E}}$ is an exact rewriting of $E_0$ wrt $\mathcal{E}$ we proceed as follows:

1. We construct an automaton $B = (\Sigma, S_B, s_{B_0}, \rho_B, F_B)$ that accepts $\exp_2(L(R_{E_0,\mathcal{E}}))$, by replacing each edge labeled by $e_i$ in $R_{E_0,\mathcal{E}}$ by an automaton $A_i$ such that $L(A_i) = L(\exp_2(e_i))$ for $i = 1, \ldots, k$. (Each edge labeled by $e_i$ is replaced by a fresh copy of $A_i$: We assume, without loss of generality, that $A_i$ has unique start state and accepting state, which are identified with the source and target of the edge, respectively.) Observe that, since $R_{E_0,\mathcal{E}}$ is a rewriting of $E_0$, $L(B) \subseteq L(A_d)$.

2. We check whether $L(A_d) \subseteq L(B)$, that is, we check whether $L(A_d \cap \overline{B}) = \emptyset$.

**Theorem 3** The automaton $R_{E_0,\mathcal{E}}$ is an exact rewriting of $E_0$ wrt $\mathcal{E}$ if and only if $L(A_d \cap \overline{B}) = \emptyset$.

**Proof.** By Theorem 2 the automaton $R_{E_0,\mathcal{E}}$ is a rewriting of $E_0$ wrt $\mathcal{E}$. Suppose $L(A_d \cap \overline{B}) = \emptyset$. Then any $\Sigma$-word $w \in L(E_0) = L(A_d)$ is also accepted by $B$. Hence by construction of $B$ there is a $\Sigma_\mathcal{E}$-word $e_1 \cdots e_n \in L(\overline{A'})$ such that $w = w_1 \cdots w_n$ and $w_i \in \exp_2(e_i)$ for $i = 1, \ldots, n$, that is rejected by $A_d$. Contradiction.
Corollary 4 An exact rewriting of \( E_0 \) wrt \( \mathcal{E} \) exists if and only if \( L(A_d \cap \overline{B}) = \emptyset \).

Example 2 (cont.) One can easily verify that \( R_{\mathcal{E},E_0} = e_2^* e_1 e_3^* \) is exact. Observe that, if \( \mathcal{E} \) did not include \( c \), the rewriting algorithm would give us \( e_2^* e_1 \) as the \( \Sigma_\mathcal{E} \)-maximal rewriting of \( E_0 \) wrt \( \{ a, a c^* b \} \), which however is not exact.

3 Complexity analysis

In this section we analyze the computational complexity of both the problem of rewriting regular expressions, and the method described in Section 2.

3.1 Upper bounds

Let us analyze the complexity of the algorithms presented above for computing the maximal rewriting of a regular expression. By considering the cost of the various steps in computing \( R_{\mathcal{E},E_0} \), we immediately derive the following theorem.

Theorem 5 The problem of generating the \( \Sigma_\mathcal{E} \)-maximal rewriting of a regular expression \( E_0 \) wrt a set \( \mathcal{E} \) of regular expressions is in 2EXP TIME.

Proof. We refer to the algorithm that computes \( R_{\mathcal{E},E_0} \), and we observe that: (i) Generating the deterministic automaton \( A_d \) from \( E_0 \) is exponential. (ii) Building \( A' \) from \( A_d \) and the expressions \( E_1, \ldots, E_k \) is polynomial. (iii) Complementing \( A' \) is again exponential. \( \square \)

With regard to the cost of verifying the existence of an exact rewriting, Corollary 4 ensures us that we can solve the problem by checking \( L(A_d \cap \overline{B}) = \emptyset \). Observe that, if we construct \( L(A_d \cap \overline{B}) \), we get a cost of 3EXP TIME, since \( \overline{B} \) is of triply exponential size with respect to the size of the input. However, we can avoid the explicit construction of \( \overline{B} \), thus getting the following result.

Theorem 6 The problem of verifying the existence of an exact rewriting of a regular expression \( E_0 \) wrt a set \( \mathcal{E} \) of regular expressions is in 2EXPSPACE.

Proof (sketch). We refer to the algorithm that verifies whether the automaton \( R_{\mathcal{E},E_0} \) is an exact rewriting of \( E_0 \) wrt \( \mathcal{E} \), and we observe that: (i) By Theorem 5, the automaton \( R_{\mathcal{E},E_0} \) is of doubly exponential size. (ii) Building the automaton \( B \) from \( R_{\mathcal{E},E_0} \) is polynomial. (iii) Complementing \( B \) to get \( \overline{B} \) is exponential. (iv) Verifying the emptiness of the intersection of \( A_d \) and \( \overline{B} \) can be done in nondeterministic logarithmic space [RS59, Jon75]. Combining (i)–(iv), we get a nondeterministic 2EXPSPACE bound, using Savitch’s Theorem [Sav70], we get a deterministic 2EXPSPACE bound.

Some care, however, is needed to getting the claimed space bound. We cannot simply construct \( \overline{B} \), since it is of triply exponential size. Instead, we construct \( \overline{B} \) “on-the-fly”; whenever the nonemptiness algorithm wants to move from a state \( s_1 \) of the intersection of \( A_d \) and \( \overline{B} \) to a state \( s_2 \), the algorithm guesses \( s_2 \) and checks that it is directly connected to \( s_1 \). Once this has been verified, the algorithm can discard \( s_1 \). Thus, at each step the algorithm needs to keep in memory at most two states and there is no need to generate all of \( \overline{B} \) at any single step of the algorithm. \( \square \)

3.2 Lower bounds

We show that the bounds established in Section 3.1 are essentially optimal.

We say that a rewriting \( R \) is \( \Sigma_\mathcal{E} \)-empty if \( L(R) = \emptyset \). We say that it is \( \Sigma_\mathcal{E} \)-empty if \( \exp_{\Sigma_\mathcal{E}}(L(R)) = \emptyset \). Clearly \( \Sigma_\mathcal{E} \)-emptiness implies \( \Sigma \)-emptiness. The converse also
Theorem 7 The problem of verifying the existence of a nonempty rewriting of a regular expression $E_0$ wrt a set $E$ of regular expressions is EXPSPACE-complete.

Proof (sketch). By Theorem 5, we generate the $\Sigma_E$-maximal rewriting of a regular expression $E_0$ wrt a set $E$ of regular expressions in 2EXPTIME. Checking whether a given finite-state automaton in non-empty can be done in NLOGSPACE. The upper bound follows (see comments in the proof of Theorem 6).

To prove the lower bound we describe a reduction from an EXPSPACE Turing machine. That is, given an EXPSPACE Turing machine $T$ we construct a regular expression $E_0$ and a set $E$ of regular expressions such that $T$ accepts an empty tape of length $n$ if and only if there is a nonempty rewriting of $E_0$ wrt $E$. We now sketch the reduction.

Let $T$ have an alphabet $\Gamma$ and a set $Q$ of states. Then configurations of $T$ can be represented as words of length $2^m$ over the configuration alphabet $\Delta = \Gamma \cup (\Gamma \times Q)$, where $m = cn$ for some constant $c$. A computation of $T$ can be described as a word over $\Delta$, where every block of $2^m$ symbols describes a configuration of $T$. We take $\Delta$ to be $\Sigma_e$. We will define $E_0$ and $re(e)$ for each letter $e \in \Delta$ such that a word $e_1 \cdots e_i$ describes an accepting computation of $T$ if and only if $exp_{\Sigma}(e_1 \cdots e_i) \subseteq L(E_0)$. $E_0$ will be defined as a sum of regular expressions $E_i$'s.

The construction of $re(e)$ for $e \in \Delta$ is uniform: we take the alphabet $\Sigma$ to be $\Delta \cup \{0, 1, \$\}$ (so $\Sigma_e \subseteq \Sigma$), and define $re(e) = \$ \cdot (0 + 1)^{3m+3} \cdot e$; that is, the language associated with $e$ consists of $e$ prefixed with a $\$ sign and all binary words of length $3m + 1$. Intuitively, the $\$ sign is a marker, the first $m$ bits encode the position of a symbol in a configuration (m bits are needed to describe the position in a configuration of length $2^m$), and the next 2$m$ bits encode bookkeeping information. The $3m + 1$-st bit is a highlight whose function will become clear shortly. Given a word $w \in L(re(e))$, we use $position(w)$ to denote the first $m$ bits, $carry(w)$ to denote the second $m$ bits, $next(w)$ to denote the third $m$ bits, $highlight(w)$ to denote the $3m + 1$-st bit, and $symbol(w)$ to denote the last symbol, which is $c$. Consider now a word $e_1 \cdots e_i$ over $\Delta$, and let $w = w_0 \cdots w_i$ be a word in $exp_{\Sigma}(e_1 \cdots e_i)$. We call each $w_i$, which is a word of length $3m + 3$, a block.

We classify such words $w$ into two classes. Our intention is that $position(w_i)$ describes an $m$-bit counter, that precisely two highlight bits be on, and that these two highlight bits be located in blocks $w_i$ and $w_j$ such that $position(w_i) = position(w_j)$ and for at most one $k, i < k < j$, we have $position(w_k) = 0^m$. Requiring $positions(w)$ to be an $m$-bit counter means that we expect $position(w_0) = 0^m$, and we expect $carry(w_i)$ to be the sequence of $m$ carry bits when $position(w_i)$ is incremented to yield $next(w_i)$, which is equal to $position(w_{i+1})$. If the intended conditions do not hold, then $w$ is a bad word. We define $E_0$ in such a way that all bad words belong to $L(E_0)$. Every violated condition can be "detected" by a regular expression $E_i$ of size $O(m)$. For example, the last carry bit need always to be 1. Thus, by taking $E_i$ to be the expression $(\Sigma_3^{m+3})^* \cdot \Sigma_2^m \cdot 0^m \cdot \Sigma_3^{m+2} \cdot (\Sigma_3^{m+3})^*$ we guarantee that words that have carry whose last bit is not 1 will be included in $L(E_0)$.

Words that satisfy these conditions are good words, and will be handled differently. In such words the two highlight bits are on at two positions that are precisely $2^m$ blocks apart. These blocks correspond to identically located cells of two adjacent configurations of the machine $T$. These cells, and their neighboring cells have to be related in a way that depends on the transition table of $T$. (Generally, cell $i$ in a configuration of a Turing machine depends only on cells $i - 1$, $i$, and $i$ + 1 in the previous configuration.) We can use regular expressions of size $O(m)$ to force such blocks to be related in the right way. Thus, all the good words $w = w_0 \cdots w_i$ in $exp_{\Sigma}(e_1 \cdots e_i)$ are in $L(E_0)$ if and only if $e_1 \cdots e_i$ describes an accepting computation of $T$. If $T$ has no accepting computation then for every $e_1 \cdots e_i$ we can find a good word $w = w_0 \cdots w_i$ in $exp_{\Sigma}(e_1 \cdots e_i)$ that is not in $L(E_0)$. Thus, $E_0$ has a nonempty rewriting wrt $E$ if and only if $T$ has an accepting computation. $\square$

Note that Theorem 7 implies that the upper bound established in Theorem 5 is essentially optimal. If we can generate maximal rewritings in, say, EXPTIME, then we could test emptiness in PSPACE, which is impossible by Theorem 7. We can get, however, an even sharper lower bound on the size of rewritings.

Theorem 8 For each $n > 0$ there is a regular expression $E_0^n$ and a set $E^n$ of regular expressions such that the combined size of $E_0^n$ and $E^n$ is polynomial in $n$, but the shortest nonempty rewriting (expressed either as a regular expression or as an automaton) of $E_0^n$ wrt $E^n$ is of length $2^n$.

Proof (sketch). We use the encoding technique of Theorem 7. Instead, however, of encoding Turing machine computations, we encode a $2^n$-bit counter. We take $E^n = \{e_0^n, e_1^n\}$ and $\Sigma_e = \{0, 1\}$. We define $E_0^n, e_0^n,$ and $e_1^n$ in such a way that $e_0^n, \ldots, e_m^n$ is a rewriting of $E_0^n$ wrt $E^n$ if and only if the bit vector $i_0 \ldots i_m$ is of the form $w_0 \ldots w_{2^m-1}$, where $w_i$ is the $2^n$-bit representation of $i$. Using pumping arguments it can be shown that any reg-
ular expression or automaton describing such a rewriting has to be of length at least $2^n$. □

The technique used in Theorem 7 turns out to be an important building block in the proof that Theorem 6 is also tight.

**Theorem 9** The problem of verifying the existence of an exact rewriting of a regular expression $E_0$ wrt a set $E$ of regular expressions is 2EXPSPACE-complete.

**Proof (sketch).** The upper bound proof is given in Theorem 6.

To prove the lower bound we describe a reduction from an 2EXPSPACE Turing machine. That is, given an 2EXPSPACE Turing machine $T$ we construct a regular expression $E_0$ and a set $E$ of regular expressions such that $T$ accepts an empty tape of length $n$ if and only if there is an exact rewriting of $E_0$ wrt $E$. Computations of 2EXPSPACE machines are sequences of configurations each of which is doubly exponentially long. Thus, to “check” such computations one needs to compare cells that are doubly exponential distance apart, which requires “yardsticks” of such length. Fortunately, we have seen in the proof of Theorem 7 how to construct such yardsticks.

Using a Turing machine $T'$ that emulates a $2^n$-bit counter (this machine is different than the 2EXPSPACE machine $T$), we use the construction described in Theorem 7 to construct a regular expression $E_0$ and a set $E$ of regular expressions such that the following property hold. For a word $w$ over $\Sigma_E$ we have that $\exp_E(w) \subseteq L(E_0)$ precisely when $w$ is in the form $\Sigma_E \cdot a \cdot \Sigma_E^2 \cdot b \cdot \Sigma_E$, where $(a, b)$ is a special pair of symbols whose only occurrence in $w$ is as described (we will use a finite set of such pairs). Let $\Delta$ be the configuration alphabet of the machine $T$. We add to $E_0$ the expression $\Delta^*$, i.e., $E_0$ expresses also all “candidate” computations of $T$. If $T$ does not have an accepting computation, then every candidate computation will have an error. We focus here on errors that arise from mismatch of symbols that are $2^n$ apart.

We now add $\Delta$ to every regular expression $re(e)$ for $e \in \Sigma_E$ with the exception of the symbols in the special pairs. (We need to extend $E_0$ in a straightforward manner to ensure that our rewriting is still a rewriting. We also need to extend $E$ to ensure that our rewriting is exact wrt the “old” part of $E_0$.) If we added $\Delta$ also to the regular expressions of symbols in special pairs, then all words in $\Delta^*$ will be contained in $\exp_E(w)$ for some word $w$ in the rewriting. Instead, for each special pair $(a, b)$ we add to $re(a)$ and $re(b)$, respectively, a pair of symbols that correspond to a possible mismatch of symbols in a candidate computation. (The finite number of such possible errors correspond to the finite number of special pairs). Thus, the rewriting generates only candidate computations with errors. Thus, if all candidate computations of $T$ have an error, the rewriting is exact. If, on the other hand, $T$ does have an accepting computation, such a computation does not have an error and will not be generated by the rewriting, resulting in a non-exact rewriting. □

### 4 Query rewriting in semi-structured data

In this section we show how to apply the results presented above to query rewriting in semi-structured data.

All semi-structured data models share the characteristic that data are organized in a labeled graph [Bun97, Abi97]. Following this idea two different approaches have been proposed:

1. The first approach associates data both to the nodes and to the edges. Specifically, nodes represent objects, and edges represent relations between objects [Abi97, QRS95, FFLS97, FFK98].

2. The second approach associates data to the edges only [BDFS97, BDHS96, FS98], but queries are not expressed directly over the constants labeling the edges of databases, but over formulae describing the properties of such edges.

An answer to a regular path query is a set of pairs of nodes connected in the database through a path conforming to the query. In the first approach the rewriting techniques proposed in Section 2 can be directly applied to rewrite regular path queries. It is sufficient to show that $R$ is a rewriting of a query $Q$ if and only if $R$ (considered as a mechanism to define a language) is a rewriting of the regular expression $Q^1$. In the second approach more care is required. In the rest of the section we concentrate on this case.

#### 4.1 Semi-structured data models and queries

From a formal point of view we can consider a (semi-structured) database as a graph $DB$ whose edges are labeled by elements from a given domain $D$ which we assume finite. We denote an edge from node $x$ to node $y$ labeled by $a$ with $x \xrightarrow{a} y$. Typically, a database will be a rooted connected graph, however in this paper we do not need to make this assumption.

In order to define queries over a semi-structured database we start from a decidable, complete first-order theory $T$ over the domain $D$. We assume that the language of $T$ includes one distinct constant for each element of $D$ (in the following we do not distinguish between constants and elements of $D$). We further assume that among the predicates of $T$ we have one unary predicate of the form $\lambda z. z = a$, for each constant $a$ in $D$.

---

1 The proof is similar to the one for Theorem 10.

2 The theory is complete in the sense that for every closed formula $\phi$, either $T$ entails $\phi$, or $T$ entails $\neg \phi$ [BDFS97].
D. We use simply a as an abbreviation for such predicate. Finally, we follow [BDFS97] and consider both the size of $\mathcal{T}$, and the time needed to check validity of any formula in $\mathcal{T}$ to be constant.

In this paper we consider regular path queries (which we call simply queries) i.e., queries that denote all the paths corresponding to words of a specified regular language. The regular language is defined over a (finite) set $\mathcal{F}$ of formulae of $\mathcal{T}$ with one free variable. Such formulae are used to describe properties that the labels of the edges of the database must satisfy. Regular path queries are the basic constituents of queries in semi-structured data, and are typically expressed by means of regular expressions [BDHS96, Ali97, FS98, MS99]. Another possibility to express regular path queries is to use finite automata.

When evaluated over a database, a query $Q$ returns the set of pairs of nodes connected by a path that conforms to the regular language $L(Q)$ defined by $Q$, according to the following definitions.

**Definition 4** Given an $\mathcal{F}$-word $\varphi_1 \cdots \varphi_n$, a $\mathcal{D}$-word $a_1 \cdots a_n$ matches $\varphi_1 \cdots \varphi_n$ (wrt $\mathcal{T}$) if and only if $\mathcal{T} \models \varphi_i(a_i)$, for $i = 1, \ldots, n$.

We denote the set of $\mathcal{D}$-words that match an $\mathcal{F}$-word $w$ by $\text{match}(w)$, and given a language $\ell$ over $\mathcal{F}$, we denote $\bigcup_{w \in \ell} \text{match}(w)$ by $\text{match}(\ell)$.

**Definition 5** The answer to a query $Q$ over a database $DB$ is the set $\text{ans}(L(Q), DB)$, where for a language $\ell$ over $\mathcal{F}$

$$\text{ans}(\ell, DB) = \{(x, y) \mid \text{there is a path } x \xrightarrow{a_1} \cdots \xrightarrow{a_n} y \text{ in } DB \text{ s.t. } a_1 \cdots a_n \in \text{match}(\ell)\}$$

### 4.2 Rewriting regular path queries

In order to apply the results on rewriting of regular expressions to query rewriting in semi-structured data we need to take into account that the alphabet over which queries (the one we want to rewrite and the views to use in the rewriting) are expressed, is the set $\mathcal{F}$ of formulae of the underlying theory $\mathcal{T}$, and not the set of constants that appear as edge labels in graph databases.

Let $Q_0$ be a regular path query and $Q = \{Q_1, \ldots, Q_k\}$ be a finite set of views, also expressed as regular path queries, in terms of which we want to rewrite $Q_0$. Let $\mathcal{F}$ be the set of formulae of $\mathcal{T}$ appearing in $Q_0, Q_1, \ldots, Q_k$, and let $Q$ have an associated alphabet $\Sigma_Q$ containing exactly one symbol for each view in $Q$. We denote the view associated to the symbol $q \in \Sigma_Q$ with $\text{rpq}(q)$.

Given any language $\ell$ over $\Sigma_Q$, we denote by $\text{exp}_\mathcal{F}(\ell)$ the language over $\mathcal{F}$ defined as follows

$$\text{exp}_\mathcal{F}(\ell) = \bigcup_{\ell \in \ell} \{w_1 \cdots w_n \mid w_i \in L(\text{rpq}(q_i))\}$$

**Definition 6** Let $R$ be any formalism for defining a language $L(R)$ over $\Sigma_Q$. $R$ is a rewriting of $Q_0$ wrt $Q$ if for every database $DB$, $\text{ans}(\text{exp}_\mathcal{F}(L(R)), DB) \subseteq \text{ans}(L(Q_0), DB)$, and is said to be

- maximal if for each rewriting $R'$ of $Q_0$ wrt $Q$ we have that $\text{ans}(\text{exp}_\mathcal{F}(L(R')), DB) \subseteq \text{ans}(\text{exp}_\mathcal{F}(L(R)), DB)$,
- exact if $\text{ans}(\text{exp}_\mathcal{F}(L(R)), DB) = \text{ans}(L(Q_0), DB)$.

**Theorem 10** $R$ is a rewriting of $Q_0$ wrt $Q$ if and only if $\text{match}(\text{exp}_\mathcal{F}(L(R))) \subseteq \text{match}(L(Q_0))$. Moreover, it is maximal if and only if for each rewriting $R'$ of $Q_0$ wrt $Q$ we have that $\text{match}(\text{exp}_\mathcal{F}(L(R'))) \subseteq \text{match}(\text{exp}_\mathcal{F}(L(R)))$, and it is exact if and only if $\text{match}(\text{exp}_\mathcal{F}(L(R))) = \text{match}(L(Q_0))$.

**Proof.** We prove only that $R$ is a rewriting of $Q_0$ wrt $Q$ iff $\text{match}(\text{exp}_\mathcal{F}(L(R))) \subseteq \text{match}(L(Q_0))$. The other assertions follow immediately.

$\Rightarrow$ By contradiction. Assume there exists a $\mathcal{D}$-word $a_1 \cdots a_n \in \text{match}(\text{exp}_\mathcal{F}(L(R)))$ such that $a_1 \cdots a_n \notin \text{match}(L(Q_0))$. Then for the database $DB$ consisting of a single path $x \xrightarrow{a_1} \cdots \xrightarrow{a_n} y$ it holds that $(x, y) \in \text{ans}(\text{exp}_\mathcal{F}(L(R)), DB)$ but $(x, y) \notin \text{ans}(L(Q_0), DB)$. Contradiction.

$\Leftarrow$ Again by contradiction. Assume there exists a database $DB$ and two nodes $x$ and $y$ in $DB$ such that $(x, y) \in \text{ans}(\text{exp}_\mathcal{F}(L(R)), DB)$ and $(x, y) \notin \text{ans}(L(Q_0), DB)$. Then there exists a path $x \xrightarrow{a_1} \cdots \xrightarrow{a_n} y$ in $DB$ such that $a_1 \cdots a_n \in \text{match}(\text{exp}_\mathcal{F}(L(R)))$. Hence $a_1 \cdots a_n \in \text{match}(L(Q_0))$ and thus $(x, y) \in \text{ans}(L(Q_0), DB)$. Contradiction.

We say that $R$ is $\Sigma_Q$-maximal if for each rewriting $R'$ of $Q_0$ wrt $Q$ we have that $L(R') \subseteq L(R)$. By arguing as in Theorem 1, and exploiting Theorem 10, it is easy to show that a $\Sigma_Q$-maximal rewriting is also maximal.

Next we show how to compute a $\Sigma_Q$-maximal rewriting, by exploiting the construction presented in Section 2. Applying the construction literally, considering $\mathcal{F}$ as the base alphabet $\Sigma$, we would not take into account the theory $\mathcal{T}$, and hence the construction would not give us the maximal rewriting in general. As an example, suppose that $\mathcal{T} \models \forall x.A(x) \lor B(x)$, $Q_0 = B$, and $Q = \{A\}$. Then the maximal rewriting of $Q_0$ wrt $Q$ is $A$, but the algorithm would give us the empty language.
In order to take the theory into account, we can proceed as follows: For each query $Q \in \{Q_0\} \cup Q$ we construct an automaton $Q^g$ accepting the language $\text{match}(L(Q))$. This can be done by viewing the query $Q$ as a (possibly nondeterministic) automaton $Q = (F, S, s_0, \rho, F)$ and construct $Q^g$ as $(D, S, s_0, \rho^g, F)$, where $s_j \in \rho(s_i, a)$ if and only if $s_j \in \rho(s_i, \varphi)$ and $T \models \varphi(a)$. Observe that the set of states of $Q$ and $Q^g$ is the same. We denote $(Q^g, \ldots, Q^g_k)$ with $Q^g$. Then we proceed as before:

1. Construct a deterministic automaton $A_d = (D, S_d, s_0, \rho_d^g, F_d)$ such that $L(A_d) = L(Q^g_0)$.
2. Define the automaton $A' = (\Sigma, S_d, s_0, \rho', F_d)$, where $s_j \in \rho(s_i, q)$ if and only if $\exists w \in \text{match}(L(\text{rpq}(q)))$ such that $s_j \in \rho_{d}^{\varphi}(s_i, w)$.
3. Return $R_{Q^g_0} = R_{Q^g, Q^g_0} = A'$.

**Theorem 11** The automaton $R_{Q^g_0}$ is a $\Sigma_Q$-maximal rewriting of $Q_0 \text{ wrt } Q$.

**Proof.** First we show that every rewriting $R$ of $Q^g_0 \text{ wrt } Q^g$ is also a rewriting of $Q_0 \text{ wrt } Q$, and vice-versa. If $R$ is a rewriting of $Q^g_0 \text{ wrt } Q^g$, then by definition $\text{exp}_D(L(R)) \subseteq L(Q^g_0)$, which implies that $\text{match}(\text{exp}_D(L(R))) \subseteq \text{match}(L(Q_0))$, i.e., $R$ is a rewriting of $Q_0 \text{ wrt } Q$. On the converse, if $R$ is a rewriting of $Q_0 \text{ wrt } Q$, then by definition $\text{match}(\text{exp}_D(L(R))) \subseteq \text{match}(L(Q_0))$ which implies that $\text{exp}_D(L(R)) \subseteq L(Q^g_0)$, i.e., $R$ is a rewriting of $Q^g_0 \text{ wrt } Q^g$.

Now, by Theorem 2 we know that $R_{Q^g_0} = R_{Q^g, Q^g_0}$ is a $\Sigma_Q$-maximal rewriting of $Q^g_0 \text{ wrt } Q^g$. Hence it is a rewriting of $Q_0 \text{ wrt } Q$.

As $R_{Q^g_0} = R_{Q^g, Q^g_0}$ is a $\Sigma_Q$-maximal rewriting of $Q^g_0 \text{ wrt } Q^g$, we have that, for each rewriting $R$ of $Q^g_0 \text{ wrt } Q^g$, and hence for each rewriting $R$ of $Q_0 \text{ wrt } Q$, $L(R) \subseteq L(R_{Q^g_0})$, which implies that $R_{Q^g_0}$ is a $\Sigma_Q$-maximal rewriting of $Q_0 \text{ wrt } Q$. \hfill \square

To check that $R_{Q^g_0}$ is an exact rewriting of $Q_0 \text{ wrt } Q$ we can proceed as in Section 2, by constructing an automaton $B$ that accepts $\text{exp}_D(L(R_{Q^g_0}))$, and checking for the emptiness of $L(A_d \cap B)$.

Observe that the size of $Q^g$ and $Q^g$ and the time needed to construct them from $Q_0$ and $Q$ are linearly related to the size of $Q_0$ and $Q$. It follows that the same upper bounds as established in Section 3.1 hold for the case of regular path queries.

In fact, the construction of $Q^g$ can be avoided in building $R_{Q^g_0}$, since we can verify whether there exists a $\mathcal{P}$-word $w \in \text{match}(L(\text{rpq}(q)))$ such that $s_j \in \rho_{d}^{\varphi}(s_i, a)$ (required in Step 2 of the algorithm above) as follows. We consider directly the automaton $Q = \text{rpq}(q)$ (which is over the alphabet $F$) and the automaton $A_d^{F} = (D, S_d, s_0, \rho_d^{\varphi}, \{s_j\})$ obtained from $A_d$ by suitably changing the initial and final states. Then we construct from $Q$ and $A_d^{F}$ the product automaton $K$, with the proviso that $K$ has a transition from $(s_1, s_2)$ to $(s'_1, s'_2)$ (whose label is irrelevant) if and only if (i) there is a transition from $s_1$ to $s'_1$ labeled $a$ in $Q_1$, (ii) there is a transition from $s_2$ to $s'_2$ labeled $\varphi$ in $Q$, and (iii) $T \models \varphi(a)$. Finally, we check whether $K$ accepts a non-empty language. This allows us to instantiate the formulae in $Q$ only to those constants that are actually necessary to generate the transition function of $A$.

With regard to $Q_0$, instead of constructing $Q^g_0$, we can build an automaton based on the idea of separating constants into suitable equivalence classes according to the formulae in the query they satisfy. The resulting automaton still describes the query $Q_0$, and its alphabet is generally much smaller than that of $Q^g_0$.

### 4.3 Properties of rewritings

In the case where the rewriting $R_{Q^g_0}$ is not exact, the only thing we know is that such rewriting is the best one we can obtain by using only the views in $Q$. However, one may want to know how to get an exact rewriting by adding to $Q$ suitable views.

**Example 3** Let $Q_0 = a \cdot (b + c)$, $Q = \{a, b, c\}$, and $\Sigma_Q = \{q_1, q_2\}$, where $\text{rpq}(q_1) = a$, and $\text{rpq}(q_2) = b$. Then $R_{Q^g_0} = q_1 \cdot q_2$, which is not exact. On the other hand, by adding to $Q$ and $q_3$ to $Q$, with $\text{rpq}(q_3) = c$, we obtain $q_1 \cdot (q_2 + q_3)$ as an exact rewriting of $Q_0$.

Here we consider the case where the views added to $Q$ are atomic, i.e., have the form $\lambda z. P(z)$, where $P$ is a predicate of $T$. Notice that atomic views include views of the form $\lambda z. z = a$, which we call elementary. The intuitive idea is to choose a subset $P'$ of the set $P$ of predicates of $T$, and to construct an exact rewriting of $Q_0 \text{ wrt } Q_+$, where $Q_+$ is obtained by adding to $Q$ an atomic view for each symbol in $P'$. An exact rewriting $R$ of $Q_0 \text{ wrt } Q_+$ is called a partial rewriting of $Q_0 \text{ wrt } Q$, provided that $Q_+ \neq Q$.

The method we have presented can be easily adapted to compute partial rewritings. Indeed, if we compute $R_{Q^g_0}$, we obtain a partial rewriting of $Q_0 \text{ wrt } Q$, provided that $R_{Q^g_0}$ is an exact rewriting of $Q_0 \text{ wrt } Q_+$. Observe that it is always possible to choose a subset $P'$ of $P$ in such a way that $R_{Q^g_0}$ is exact (e.g., by choosing the set of all elementary views).

Typically, one is interested in using as few symbols of $P$ as possible to form $Q_+$, and this corresponds to choosing the minimal subsets $P'$ such that $R_{Q^g_0}$ is exact. More generally, one can establish various preference criteria for choosing rewritings. For instance, we may say that a (partial) rewriting $R$ is preferable to a (partial) rewriting $R'$ if one of the following holds:
1. $\text{match}(\exp_\mathcal{F}(L(R')) \subset \text{match}(\exp_\mathcal{F}(L(R)))$,

2. $\text{match}(L(R)) = \text{match}(L(R'))$ and $R$ uses less additional elementary views than $R'$,

3. $\text{match}(L(R)) = \text{match}(L(R'))$, $R$ uses the same number of additional elementary views as $R'$, and less additional atomic nonelementary views.

4. $\text{match}(L(R)) = \text{match}(L(R'))$, $R$ uses the same number of additional atomic views as $R'$, and less views than $R'$.

Under this definition an exact rewriting is preferable to a nonexact one. Moreover, the definition reflects the fact that the cost of materializing additional atomic views (in particular the elementary ones) is higher than the cost of using the available ones. Finally, since a certain cost is associated to the use of each view, when comparing two rewritings defining the same language and using (if any) the same number of additional atomic views, then the one that uses less views is preferable.

The rewriting algorithm presented above can be immediately exploited to compute the most preferable rewritings according to the above criteria. It easy to see that the problem of computing the most preferable rewritings remains in the same complexity class.

5 Conclusions

In this paper we have studied the problem of query rewriting using views in the case where both the query and the views are expressed as regular path queries. We have shown the decidability of the problem of computing the maximal rewriting and checking whether it is exact. We have characterized the computational complexity of the problem and have provided algorithms that are essentially optimal. We envision several directions for extending the present work.

First, in this paper we focused on the problem of computing the maximal contained rewriting, i.e., the best rewriting that is guaranteed to provide only answers contained in those of the original query. Also of interest is the dual approach, i.e., computing the minimal containing rewritings (in general not unique), which guarantee to provide all the answers of the original query, and possibly more.

Second, an extension of regular path queries are generalized path queries, i.e., queries of the form $x_1 Q_1 x_2 \cdots x_{n-1} Q_{n-1} x_n$, where each $Q_i$ is a regular path query [FS98]. Such queries ask for all $n$-tuples $o_1, \ldots, o_n$ of nodes such that, for each $i$, there is a path from $o_i$ to $o_{i+1}$ that satisfies $Q_i$. Computing the rewriting of a generalized path query requires to take into account that each rewritten subpath appears in a given context formed by a suitable prefix and a suitable suffix.

A further generalization would be to consider conjunctions of regular path queries, where the context in which a certain subpath appears is even more complex.

Third, one can investigate possible interesting subcases where the rewriting of regular (and generalized) path queries can be done more efficiently. Additionally, cost models for path queries and preference criteria that take into account such cost models can be defined, leading to the development of techniques for choosing the best rewriting with respect to the new criteria.

Finally, it is interesting to investigate the relationships to query answering using views in semi-structured data, i.e., the problem of answering a regular path query on the basis of a set of materialized views. One relevant aspect is to verify whether the technique we have developed for query rewriting can be exploited for query answering using views. First results in this direction are reported in [CDGLV96].

Acknowledgments

This work was supported in part by the NSF grants CCR-9628400 and CCR-9700061, by MURST, by ESPRIT LTR Project No. 22469 DWQ (Foundations of Data Warehouse Quality), and by the Italian Space Agency (ASI) under project “Integrazione ed Accesso a Basi di Dati Eterogenee”. Part of this work was done when the last author was a Varon Visiting Professor at the Weizmann Institute of Science.

References


