Rewriting of Regular Expressions and Regular Path Queries

Diego Calvanese[1], Giuseppe De Giacomo[1], Maurizio Lenzerini[1], Moshe Y. Vardi[2]

[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`

[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 $(\_^* \cdot (\mathit{rome} + \mathit{jerusalem}) \cdot \_^* \cdot \mathit{restaurant})$ specifies all the paths having at some point an edge labeled *rome* or *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, FLS98].

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, CNS99], recursive queries [DG97], and queries expressed in Description Logics [BLR97]. Rewriting techniques for query optimization are described, for example, in [CKPS95, ACPS96, TSI96], 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 2EXPTIME, 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 Σ, 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 $re(e)$. Given any language ℓ over $\Sigma_\mathcal{E}$, we denote by $exp_\Sigma(\ell)$ the language over Σ defined as follows

$$exp_\Sigma(\ell) = \bigcup_{e_1 \cdots e_n \in \ell} \{w_1 \cdots w_n \mid w_i \in L(re(e_i))\}$$

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 $exp_\Sigma(L(R)) \subseteq L(E_0)$. ■

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 Σ-*maximal* if for each rewriting R' of E_0 wrt $\mathcal{E}$ we have that $exp_\Sigma(L(R')) \subseteq exp_\Sigma(L(R))$. 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 Σ-maximal rewritings we look at the languages obtained after substituting each symbol in the rewriting by the corresponding regular expression over Σ, whereas when considering $\Sigma_\mathcal{E}$-maximal rewritings we look at the languages over $\Sigma_\mathcal{E}$. Observe that by definition all Σ-maximal rewritings define the same language (similarly for $\Sigma_\mathcal{E}$-maximal rewritings), and that not all Σ-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 Σ-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 Σ-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 Σ-maximal.*

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

Given E_0 and $\mathcal{E}$, we are interested in deriving a Σ-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 Σ-maximal.

The construction takes E_0 and $\mathcal{E}$ as input, and returns an automaton $R_{\mathcal{E},E_0}$ 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 $\exists w \in L(re(e))$ such that $s_j \in \rho^*(s_i, w)$.
3. $R_{\mathcal{E},E_0} = \overline{A'}$, 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 Σ-words $w_1, \ldots, w_n$ such that $w_i \in L(re(e_i))$ for $i = 1, \ldots, n$ and such that the Σ-word $w_1 \cdots w_n$ is rejected by A_d. On the other hand if there exists n Σ-words $w_1, \ldots, w_n$ such that $w_i \in L(re(e_i))$, 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 Σ-word in $exp_\Sigma(\{e_1 \cdots e_n\})$ that is rejected by A_d. Hence, $R_{\mathcal{E},E_0}$, being the complement of A', accepts a $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n$ if and only if all Σ-words $w = w_1 \cdots w_n$ such that $w_i \in L(re(e_i))$ for $i = 1, \ldots, n$, are accepted by A_d. Hence we can state the following theorem.

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

Proof. It is easy to see that by construction $R_{\mathcal{E},E_0} = \overline{A'}$ 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 $e_1 \cdots e_n \notin L(\overline{A'})$. By definition of rewriting, all Σ-words $w_1 \cdots w_n$ such that $w_i \in L(re(e_i))$ for $i = 1, \ldots, n$, are in $L(E_0) = L(A_d)$. On the other hand, since $e_1 \cdots e_n \notin L(\overline{A'})$, the $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n$ is accepted by A'. Thus there is a Σ-word $w_1 \cdots w_n$, such that $w_i \in L(re(e_i))$ 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 Σ) 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 $\Sigma_{\mathcal{E}}$ 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 $R_{\mathcal{E},E_0}$. ■

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

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

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

1. We construct an automaton $B = (\Sigma, S_B, s_{B0}, \rho_B, F_B)$ that accepts $exp_\Sigma(L(R_{\mathcal{E},E_0}))$, by replacing each edge labeled by e_i in $R_{\mathcal{E},E_0}$ by an automaton A_i such that $L(A_i) = L(re(e_i))$ 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 $R_{\mathcal{E},E_0}$ 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 $R_{\mathcal{E},E_0}$ 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 $R_{\mathcal{E},E_0}$ is a rewriting of E_0 wrt $\mathcal{E}$. Suppose $L(A_d \cap \overline{B}) = \emptyset$. Then any Σ-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 $w_i \in$

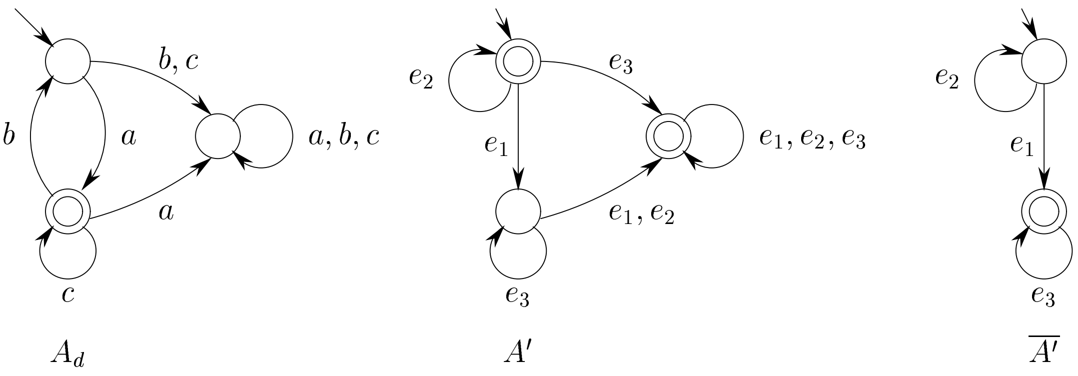

Figure 1: Construction of the rewriting of $a{\cdot}(b{\cdot}a + c)^*$ wrt $\{a,\ a{\cdot}c^*{\cdot}b,\ c\}$

$L(re(e_i))$ for $i = 1, \ldots, n$. Suppose that $L(A_d \cap \overline{B}) \neq \emptyset$. Then there exists a Σ-word $w \in L(E_0) = L(A_d)$ that is rejected by B. Hence by construction of B there is no $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n \in L(\overline{A'})$ such that $w = w_1 \cdots w_n$ and $w_i \in L(re(e_i))$ for $i = 1, \ldots, n$. □

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 $R_{\mathcal{E},E_0} = e_2^*{\cdot}e_1{\cdot}e_3^*$ is exact. Observe that, if $\mathcal{E}$ did not include c, the rewriting algorithm would give us $e_2^*{\cdot}e_1$ as the $\Sigma_{\mathcal{E}}$-maximal rewriting of E_0 wrt $\{a,\ a{\cdot}c^*{\cdot}b\}$, 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 2EXPTIME.*

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. □

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 3EXPTIME, 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. □

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 Σ-*empty* if $exp_\Sigma(L(R)) = \emptyset$. Clearly $\Sigma_{\mathcal{E}}$-emptiness implies Σ-emptiness. The converse also holds except for the non-interesting case where $\mathcal{E}$ contains one or more expressions E such that $L(E) = \emptyset$. Therefore, we will talk about the emptiness of a rewriting R without distinguishing between the two definitions.

Theorem 7 *The problem of verifying the existence of a nonempty rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ of regular expressions is EXPSPACE-complete.*

Proof (sketch). By Theorem 5, we generate the $\Sigma_{\mathcal{E}}$-maximal rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ 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 $\mathcal{E}$ 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 $\mathcal{E}$. We now sketch the reduction.

Let T have an alphabet Γ 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 Δ, where every block of 2^m symbols describes a configuration of T. We take Δ to be $\Sigma_{\mathcal{E}}$. We will define E_0 and $re(e)$ for each letter $e \in \Delta$ such that a word $e_1 \cdots e_l$ describes an accepting computation of T if and only if $exp_{\Sigma}(e_1 \cdots e_l) \subseteq L(E_0)$. 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 Σ to be $\Delta \cup \{0, 1, \$\}$ (so $\Sigma_{\mathcal{E}} \subseteq \Sigma$), and define $re(e) = \$ \cdot (0+1)^{3m+1} \cdot e$; 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 $2m$ 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, $highlite(w)$ to denote the $3m+1$-st bit, and $symbol(w)$ to denote the last symbol, which is e. Consider now a word $e_1 \cdots e_l$ over Δ, and let $w = w_1 \cdots w_l$ be a word in $exp_{\Sigma}(e_1 \cdots e_l)$. 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_1 to be the expression $(\Sigma^{3m+3})^* \cdot \Sigma^{2m} \cdot 0 \cdot \Sigma^{m+2} \cdot (\Sigma^{3m+3})^*$ 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 $w = w_1 \cdots w_l$ in $exp_{\Sigma}(e_1 \cdots e_l)$ are in $L(E_0)$ if and only if $e_1 \cdots e_l$ describes an accepting computation of T. If T has no accepting computation then for every $e_1 \cdots e_l$ we can find a good word $w = w_1 \cdots w_l$ in $exp_{\Sigma}(e_1 \cdots e_l)$ that is not in $L(E_0)$. Thus, E_0 has a nonempty rewriting wrt $\mathcal{E}$ 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 $\mathcal{E}^n$ of regular expressions such that the combined size of E_0^n and $\mathcal{E}^n$ is polynomial in n, but the shortest nonempty rewriting (expressed either as a regular expression or as an automaton) of E_0^n wrt $\mathcal{E}^n$ is of length 2^{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 $\mathcal{E}^n = \{e_0^n, e_1^n\}$ and $\Sigma_{\mathcal{E}} = \{0, 1\}$. We define E_0^n, e_0^n, and e_1^n in such a way that $e_{i_0}^n \dots e_{i_m}^n$ is a rewriting of E_0^n wrt $\mathcal{E}^n$ if and only if the bit vector $i_0 \dots i_m$ is of the form $w_0 \dots w_{2^{2^n}-1}$, where w_i is the 2^n-bit representation of i. Using pumping arguments it can be shown that any regular expression or automaton describing such a rewriting has to be of length at least 2^{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 $\mathcal{E}$ 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 $\mathcal{E}$ 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 $\mathcal{E}$. 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 $\mathcal{E}$ of regular expressions such that the following property hold. For a word w over $\Sigma_{\mathcal{E}}$ we have that $exp_\Sigma(w) \subseteq L(E_0)$ precisely when w is in the form $\Sigma_{\mathcal{E}}^* \cdot a \cdot \Sigma_{\mathcal{E}}^{2^{2^n}} \cdot b \cdot \Sigma_{\mathcal{E}}^*$, where (a, b) are a *special pair* of symbols whose only occurrence in w is as described (we will use a finite set of such pairs). Let Δ be the configuration alphabet of the machine T. We add to E_0 the expression Δ^*, 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^{2^n} apart.

We now add Δ to every regular expression $re(e)$ for $e \in \Sigma_{\mathcal{E}}$ with the exception of the symbols in the special pairs. (We need to extend E_0 is a straightforward manner to ensure that our rewriting is still a rewriting. We also need to extend $\mathcal{E}$ to ensure that our rewriting is exact wrt the "old" part of E_0.) If we added Δ also to the regular expressions of symbols in special pairs, than all words in Δ^* will be contained in $exp_\Sigma(w)$ 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, QRS+95, FFLS97, FFK+98].

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 $\mathcal{D}$ 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[2] first-order theory $\mathcal{T}$ over the domain $\mathcal{D}$. We assume that the language of $\mathcal{T}$ includes one distinct constant for each element of $\mathcal{D}$ (in the following we do not distinguish between constants and elements of $\mathcal{D}$). We further assume that among the predicates of $\mathcal{T}$ we have one unary predicate of the form $\lambda z.z = a$, for each constant a in

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

[2] The theory is complete in the sense that for every closed formula φ, either $\mathcal{T}$ entails φ, or $\mathcal{T}$ entails $\neg\varphi$ [BDFS97].

$\mathcal{D}$. 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, Abi97, 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 $match(w)$, and given a language ℓ over $\mathcal{F}$, we denote $\bigcup_{w \in \ell} match(w)$ by $match(\ell)$.

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

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

∎

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 $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ 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 $\mathcal{Q}$ have an associated alphabet $\Sigma_{\mathcal{Q}}$ containing exactly one symbol for each view in $\mathcal{Q}$. We denote the view associated to the symbol $q \in \Sigma_{\mathcal{Q}}$ with $rpq(q)$.

Given any language ℓ over $\Sigma_{\mathcal{Q}}$, we denote by $exp_{\mathcal{F}}(\ell)$ the language over $\mathcal{F}$ defined as follows

$$exp_{\mathcal{F}}(\ell) = \bigcup_{q_1 \cdots q_n \in \ell} \{w_1 \cdots w_n \mid w_i \in L(rpq(q_i))\}$$

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

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

∎

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

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

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

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

We say that R is $\Sigma_{\mathcal{Q}}$*-maximal* if for each rewriting R' of Q_0 wrt $\mathcal{Q}$ 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 $\Sigma_{\mathcal{Q}}$-maximal rewriting is also maximal.

Next we show how to compute a $\Sigma_{\mathcal{Q}}$-maximal rewriting, by exploiting the construction presented in Section 2. Applying the construction literally, considering $\mathcal{F}$ as the base alphabet Σ, 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 $\mathcal{T} \models \forall x.A(x) \supset B(x)$, $Q_0 = B$, and $\mathcal{Q} = \{A\}$. Then the maximal rewriting of Q_0 wrt $\mathcal{Q}$ 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 $Q \in \{Q_0\} \cup \mathcal{Q}$ we construct an automaton Q^g accepting the language $match(L(Q))$. This can be done by viewing the query Q as a (possibly nondeterministic) automaton $Q = (\mathcal{F}, S, s_0, \rho, F)$ and construct Q^g as $(\mathcal{D}, S, s_0, \rho^g, F)$, where $s_j \in \rho^g(s_i, a)$ if and only if $s_j \in \rho(s_i, \varphi)$ and $\mathcal{T} \models \varphi(a)$. Observe that the set of states of Q and Q^g is the same. We denote $\{Q_1^g, \ldots, Q_k^g\}$ with $\mathcal{Q}^g$. Then we proceed as before:

1. Construct a deterministic automaton $A_d = (\mathcal{D}, S_d, s_0, \rho_d^g, F_d)$ such that $L(A_d) = L(Q_0^g)$.
2. Define the automaton $A' = (\Sigma_{\mathcal{Q}}, S_d, s_0, \rho', S_d - F_d)$, where $s_j \in \rho'(s_i, q)$ if and only if $\exists w \in match(L(rpq(q)))$ such that $s_j \in \rho_d^{g*}(s_i, w)$.
3. Return $R_{\mathcal{Q},Q_0} = R_{\mathcal{Q}^g,Q_0^g} = \overline{A'}$.

Theorem 11 *The automaton $R_{\mathcal{Q},Q_0}$ is a $\Sigma_{\mathcal{Q}}$-maximal rewriting of Q_0 wrt $\mathcal{Q}$.*

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

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

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

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

Observe that both the size of Q_0^g and $\mathcal{Q}^g$ and the time needed to construct them from Q_0 and $\mathcal{Q}$ are linearly related to the size of Q_0 and $\mathcal{Q}$. 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 $\mathcal{Q}^g$ can be avoided in building $R_{\mathcal{Q},Q_0}$, since we can verify whether there exists a $\mathcal{D}$-word $w \in match(L(rpq(q)))$ such that $s_j \in \rho_d^{g*}(s_i, w)$ (required in Step 2 of the algorithm above) as follows. We consider directly the automaton $Q = rpq(q)$ (which is over the alphabet $\mathcal{F}$) and the automaton $A_d^{i,j} = (\mathcal{D}, S_d, s_i, \rho_d^g, \{s_j\})$ obtained from A_d by suitably changing the initial and final states. Then we construct from Q and $A_d^{i,j}$ 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 $Q_{i,j}$, (ii) there is a transition from s_2 to s_2' labeled φ in Q, and (iii) $\mathcal{T} \models \varphi(a)$. Finally, we check whether K accepts a non-empty language. This allows us to instantiate the formulae in $\mathcal{Q}$ only to those constants that are actually necessary to generate the transition function of A'.

With regard to Q_0, instead of constructing Q_0^g, 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_0^g.

4.3 Properties of rewritings

In the case where the rewriting $R_{\mathcal{Q},Q_0}$ 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 $\mathcal{Q}$. However, one may want to know how to get an exact rewriting by adding to $\mathcal{Q}$ suitable views.

Example 3 Let $Q_0 = a \cdot (b + c)$, $\mathcal{Q} = \{a, b\}$, and $\Sigma_{\mathcal{Q}} = \{q_1, q_2\}$, where $rpq(q_1) = a$, and $rpq(q_2) = b$. Then $R_{\mathcal{Q},Q_0} = q_1 \cdot q_2$, which is not exact. On the other hand, by adding c to $\mathcal{Q}$ and q_3 to $\Sigma_{\mathcal{Q}}$, with $rpq(q_3) = c$, 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 $\mathcal{Q}$ are *atomic*, i.e., have the form $\lambda z.P(z)$, where P is a predicate of $\mathcal{T}$. Notice that atomic views include views of the form $\lambda z.z = a$, (abbreviated by a), which we call *elementary*. The intuitive idea is to choose a subset $\mathcal{P}'$ of the set $\mathcal{P}$ of predicates of $\mathcal{T}$, and to construct an exact rewriting of Q_0 wrt $\mathcal{Q}_+$, where $\mathcal{Q}_+$ is obtained by adding to $\mathcal{Q}$ an atomic view for each symbol in $\mathcal{P}'$. An exact rewriting R of Q_0 wrt $\mathcal{Q}_+$ is called a *partial rewriting* of Q_0 wrt $\mathcal{Q}$, provided that $\mathcal{Q}_+ \neq \mathcal{Q}$.

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

Typically, one is interested in using as few symbols of $\mathcal{P}$ as possible to form $\mathcal{Q}_+$, and this corresponds to choose the minimal subsets $\mathcal{P}'$ such that $R_{\mathcal{Q}_+,Q_0}$ 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. $match(exp_{\mathcal{F}}(L(R'))) \subset match(exp_{\mathcal{F}}(L(R)))$,
2. $match(L(R)) = match(L(R'))$ and R uses less additional elementary views than R',
3. $match(L(R)) = match(L(R'))$, R uses the same number of additional elementary views as R', and less additional atomic nonelementary views.
4. $match(L(R)) = match(L(R'))$, 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 [CDGLV99].

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

[Abi97] S. Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 1–18, 1997.

[ACPS96] 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.

[AD98] 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.

[AQM+97] 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.

[AV97] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 122–133, 1997.

[BDFS97] P. Buneman, S. Davidson, M. 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.

[BDHS96] 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.

[BFW98] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-98)*, pages 129–138, 1998.

[BLR97] 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.

[Bun97] 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.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In R. T. Snodgrass and M. Winslett, editors, *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 313–324, Minneapolis (Minnesota, USA), 1994.

[CDGL98] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-98)*, pages 149–158, 1998.

[CDGLV99] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. Technical report, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.

[CKPS95] 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.

[CM90] M. P. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proc. of the 9th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-90)*, pages 404–416, Atlantic City (NJ, USA), 1990.

[CMW87] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 323–330, San Francisco (CA, USA), 1987.

[CNS99] 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)*, 1999.

[DG97] 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.

[FFK+98] 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.

[FFLS97] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[FLS98] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-98)*, pages 139–148, 1998.

[FS98] 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.

[Jon75] N. D. Jones. Space-bounded reducibility among combinatorial problems. *J. of Computer and System Sciences*, 11:68–75, 1975.

[LMSS95] 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.

[MMM97] A. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[MS99] 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.

[QRS+95] 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.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[RSU95] 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.

[Sav70] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. of Computer and System Sciences*, 4:177–192, 1970.

[SDJL96] 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.

[TSI96] 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.

[Ull97] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, number 1186 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 1997.