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], disjunctive views [DG98, AGK99], non-recursive queries and views for semi-structured data [PV99], 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. Thus, $exp_{\Sigma}(\ell)$ denotes all the words obtained from a word $e_1 \cdots e_n \in \ell$ by substituting for each e_i any word of the regular language associated to e_i.

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

Note that we do not constrain in any way the form of the rewritings, which, a priori, need not even be recursive. 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 a maximal rewriting always exists (although it may be empty). 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 exist 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 $w_1 \cdots w_n$ 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, 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 nonfinal 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.

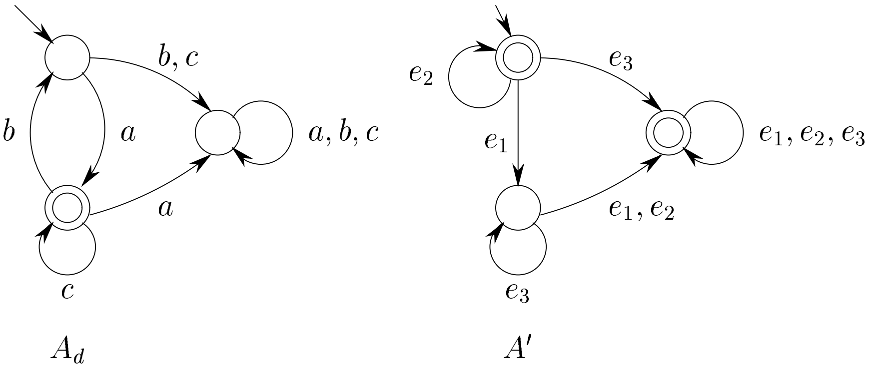

Figure 1: Construction of the rewriting of $a{\cdot}(b{\cdot}a + c)^*$ wrt $\{a,\ a{\cdot}c^*{\cdot}b,\ c\}$

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 over Σ that accepts $exp_\Sigma(L(R_{\mathcal{E},E_0}))$ as follows. We first construct an automaton A_i such that $L(A_i) = L(re(e_i))$ for $i = 1, \ldots, k$. We assume, without loss of generality, that A_i has unique start state and accepting state, and that the start state has no incoming edges and the accepting state no outgoing edges. We then obtain B by replacing each edge labeled by e_i in $R_{\mathcal{E},E_0}$ by a fresh copy of A_i, identifying the start state of the fresh copy with the source of the edge, and the accepting state with the target of the edge. 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 $w_i \in L(re(e_i))$ 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. 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, and 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 upper bounds established in Section 3.1 are essentially optimal. To prove the matching lower bounds we exploit variants of tiling problems (see e.g., [vEB82, vEB97, Ber66]). A *tile* is a unit square of one of several types and a *tiling system* is specified by means of a finite set Δ of tile types and two binary relations H and V over Δ, representing horizontal and vertical adjacency relations, respectively. A *generic tiling problem* consists in determining whether there exists a mapping τ (called *tiling*) from a given region R of the integer plane to Δ which is consistent with H and V. That is, if $(i,j),(i,j+1) \in R$ then $(\tau(i,j),\tau(i,j+1)) \in H$ and if $(i,j),(i+1,j) \in R$ then $(\tau(i,j),\tau(i+1,j)) \in V$. We get a specific tiling problem by imposing additional conditions on the region to be tiled and on the tile types that can be placed in certain positions of the region, such as the first/last row/column, or the borders.

Different tiling problems have been shown to be complete for various complexity classes [vEB82, vEB97]. We will use EXPSPACE and 2EXPSPACE-complete tiling problems.

3.2.1 Existence of a nonempty rewriting

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.

We consider the tiling problem $T = (\Delta, H, V, t_S, t_F, C_{ES})$, where t_S and t_F are two distinguished tile types in Δ, and for a given number n in unary, C_{ES} requires to tile a region of size $O(2^n) \times k$, for some constant c and some number k, in such a way that the left bottom tile of the region (i.e., the one in position $(0, 0)$) is of type t_S and the right upper tile (i.e., the one in position $(c2^n - 1, k - 1)$, for some constant c) is of type t_F. Using a reduction from acceptance of EXPSPACE Turing machines analogous to the one in [vEB97], it can be shown that this variant of tiling problem is EXPSPACE-complete.

We exploit such a tiling problem to prove the EXPSPACE lower bound of the problem of verifying the existence of a nonempty rewriting. That is, given an instance T of the above tiling problem and a number n, we construct a regular expression E_0 and a set $\mathcal{E}$ of regular expressions such that a tiling corresponding to T (a T-tiling) exists if and only if there is a nonempty rewriting of E_0 wrt $\mathcal{E}$.

Let $m = cn$ for some constant c. A tiling of a region of size $2^m \times k$ can be described as a word over Δ of length $k2^m$, where every block of 2^m symbols describes a row of the tiling. We take $\Sigma_{\mathcal{E}}$ to be Δ. We will define E_0 and $re(e)$ for each letter $e \in \Delta$ such that a Δ-word $e_1 \cdots e_\ell$ describes a T-tiling if and only if $exp_{\Sigma}(e_1 \cdots e_\ell) \subseteq L(E_0)$. E_0 will be defined as the sum $E_{bad} + E_{good}$ of two regular expressions E_{bad} and E_{good}, which are in turn defined as sums of regular expressions.

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 column of a tile (m bits are needed to describe the column in a row of length 2^m), and the next $2m$ bits encode bookkeeping information. The $3m + 1$-st bit is a *highlight*. As will become clear shortly, highlights are used to identify either a tile not in the last column or a pair of vertically adjacent tiles. Given a word $w \in L(re(e))$, we use

- $position(w)$ to denote the first m bits after the \$ marker,
- $carry(w)$ to denote the second m bits after the \$ marker, and
- $next(w)$ to denote the third m bits after the \$ marker.

Also, we use $position(w, i)$, $carry(w, i)$ and $next(w, i)$, for $0 \leq i < m$ to denote the $i + 1$-st bit in $position(w)$, $carry(w)$, and $next(w)$, respectively. This means that we count bits starting from 0 and consider the least significant bit to be the one in position 0.

Consider now a word $e_0 \cdots e_\ell$ over Δ, and let $w = w_0 \cdots w_\ell$ be a word in $exp_{\Sigma}(e_0 \cdots e_\ell)$. We call each w_j, which is a word of length $3m + 3$, a *block*. We classify such words w into two classes. Our intention is that $position(w_j)$ describes an m-bit counter, and that precisely one or two highlight bits be on. When only one highlight bit is on it is located in a block w_h such that $position(w_h) \neq 1^m$, and when two highlight bits are on, they are located in blocks w_h and w_k such that $position(w_h) = position(w_k)$ and for at most one j, $h < j < k$, we have $position(w_j) = 0^m$. Requiring $position(w_j)$ to be an m-bit counter means that we expect $position(w_0) = 0^m$ and $position(w_\ell) = 1^m$, and we expect $carry(w_j)$ to be the sequence of m carry bits when $position(w_j)$ is incremented to yield $next(w_j)$, which is equal to $position(w_{j+1})$. If the intended conditions do not hold, then w is a *bad* word. More precisely, a word $w = w_0 \cdots w_\ell$ is bad if one of the following holds:

1. $\mathit{position}(w_0, i) = 1$, for some i, $0 \le i < m$;
2. $\mathit{position}(w_\ell, i) = 0$, for some i, $0 \le i < m$;
3. $\mathit{carry}(w_j, 0) = 0$, for some j, $0 \le j \le \ell$;
4. $\mathit{carry}(w_j, i) \ne \mathit{carry}(w_j, i-1)\,\textbf{and}\,\mathit{position}(w_j, i-1)$, for some j and i, $0 \le j \le \ell$, $1 \le i < m$;
5. $\mathit{next}(w_j, i) \ne \mathit{position}(w_j, i)\,\textbf{xor}\,\mathit{carry}(w_j, i)$, for some j and i, $0 \le j \le \ell$, $0 \le i < m$;
6. $\mathit{position}(w_j, i) \ne \mathit{next}(w_{j-1}, i)$, for some j and i, $1 \le j \le \ell$, $0 \le i < m$;
7. conditions on the highlight bits, which are:
   (a) no highlight bit in w is 1;
   (b) only one highlight bit in w is 1 and it is located in a block w_h such that $\mathit{position}(w_h) = 1^m$;
   (c) at least three highlight bits in w are 1;
   (d) the two highlight bits that are 1 are located in two blocks w_h and w_k and there are at least two blocks w_{j_1} and w_{j_2} between w_h and w_k such that $\mathit{position}(w_{j_1}) = \mathit{position}(w_{j_2}) = 0^m$;
   (e) the two highlight bits that are 1 are located in two blocks w_h and w_k and $\mathit{position}(w_h, i) \ne \mathit{position}(w_k, i)$ for some i, $0 \le i < m$.

We define E_{bad} in such a way that all bad words belong to $L(E_{bad})$. Each of the above conditions can be "detected" by a regular expression of size $O(m)$, which contributes to E_{bad} (and hence to E_0). To illustrate the idea, we provide the regular expressions for some of the conditions above.

Condition (1) is detected by the regular expression

$$(\sum_{i=0}^{m-1} \$\cdot(0+1)^i\cdot 1\cdot(0+1)^{3m-i}\cdot\Delta)\cdot B^*$$

where B stands for the regular expression $\$\cdot(0+1)^{3m+1}\cdot\Delta$.

Condition (4) is detected by the sum of four regular expressions

$$B^*\cdot(\sum_{i=1}^{m-1} \$\cdot(0+1)^{i-1}\cdot p\cdot(0+1)^{m-i}\cdot (0+1)^{i-1}\cdot c\cdot c'(0+1)^{m-1-i}\cdot(0+1)^{m+1}\cdot\Delta)\cdot B^*$$

one for each choice of 0 or 1 for p, c, and c' such that $c' \ne c\,\textbf{and}\,p$.

Condition (6) is detected by the sum of two regular expressions

$$B^*\cdot(\sum_{i=0}^{m-1} \$\cdot(0+1)^{2m}\cdot(0+1)^i\cdot b\cdot(0+1)^{m-1-i}\cdot(0+1)\cdot\Delta\cdot \$\cdot(0+1)^i\cdot\bar{b}\cdot(0+1)^{m-1-i}\cdot(0+1)^{2m}\cdot(0+1)\cdot\Delta)\cdot B^*$$

one for $b = 0$ and $\bar{b} = 1$, and one for $b = 1$ and $\bar{b} = 0$.

Condition (7b) is detected by the regular expression

$$(\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^*\cdot\$\cdot 1^m\cdot(0+1)^{2m}\cdot 1\cdot\Delta\cdot(\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^*$$

Condition (7e) is detected by the sum of two regular expressions

$$B^* \cdot (\sum_{i=0}^{m-1} \begin{array}{l} \$\cdot(0+1)^i\cdot b\cdot(0+1)^{3m-1-i}\cdot 1\cdot\Delta \cdot B^*\cdot \\ \$\cdot(0+1)^i\cdot\bar{b}\cdot(0+1)^{3m-1-i}\cdot 1\cdot\Delta) \cdot B^* \end{array}$$

one for $b=0$ and $\bar{b}=1$, and one for $b=1$ and $\bar{b}=0$.

Words that satisfy none of the above conditions are *good* words, and will be handled differently. In such words either one or two highlight bits are on. When one highlight bit is on, it is located at a block that corresponds to a tile not in the last row in a tiling of the region. The types of this tile and of the one immediately to the right have to be related in a way that depends on the horizontal adjacency relation H of T. When two highlight bits are on, they are located at two positions that are precisely 2^m blocks apart, and these blocks correspond to vertically adjacent tiles. The types of these tiles have to be related in a way that depends on the vertical adjacency relation V of T. We can use regular expressions of size $O(m)$ to force such blocks to be related in the right way, and also to force the tiling to satisfy the additional conditions on the left bottom and right upper tiles. E_{good} is the sum of all such regular expressions.

For example, the following regular expression ensures that the horizontal adjacency relation is respected in the case where the highlight bit is on at a block that is neither the first nor the last one:

$$\begin{array}{l} \$\cdot(0+1)^{3m}\cdot 0\cdot t_S \cdot (\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^* \cdot \\ (\sum_{(t_1,t_2)\in H} \$\cdot(0+1)^{3m}\cdot 1\cdot t_1 \cdot \$\cdot(0+1)^{3m}\cdot 0\cdot t_2) \cdot \\ (\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^* \cdot \$\cdot(0+1)^{3m}\cdot 0\cdot t_F \end{array}$$

The following regular expression ensures that the vertical adjacency relation is respected in the case where the two highlight bits are on at blocks that are neither the first nor the last one:

$$\begin{array}{l} \$\cdot(0+1)^{3m}\cdot 0\cdot t_S \cdot (\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^* \cdot \\ (\sum_{(t_1,t_2)\in V} \$\cdot(0+1)^{3m}\cdot 1\cdot t_1 \cdot (\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^* \cdot \$\cdot(0+1)^{3m}\cdot 1\cdot t_2) \cdot \\ (\$\cdot(0+1)^{3m}\cdot 0\cdot\Delta)^* \cdot \$\cdot(0+1)^{3m}\cdot 0\cdot t_F \end{array}$$

Similar regular expressions can be provided for the cases where the highlight bits are on at the first or last block.

Thus, all the good words $w = w_0 \cdots w_\ell$ in $exp_\Sigma(e_0 \cdots e_\ell)$ are in $L(E_{good})$ if and only if $e_0 \cdots e_\ell$ describes a T-tiling. If no T-tiling exists then for every $e_0 \cdots e_\ell$ we can find a good word $w = w_0 \cdots w_\ell$ in $exp_\Sigma(e_0 \cdots e_\ell)$ that is not in $L(E_{good})$ and hence not in $L(E_0)$. Thus, E_0 has a nonempty rewriting wrt $\mathcal{E}$ if and only if a T-tiling exists.

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. 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). The lower bound follows from the reduction from the EXPSPACE complete tiling problem described above, by observing that E_0 and all regular expressions in $\mathcal{E}$ are of size polynomial in T and n. $\square$

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^n and a set $\mathcal{E}^n$ of regular expressions such that the combined size of E^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^n wrt $\mathcal{E}^n$ is of length 2^{2^n}.*

Proof. We use the encoding technique of Theorem 7. Instead, however, of encoding tiling problems, we directly encode a 2^n-bit counter using an alphabet $\Sigma_{\mathcal{E}} = \{b^n_{000}, b^n_{001}, \dots, b^n_{111}\}$ of 8 symbols representing the 8 possible combinations of a *position*, a *carry*, and a *next* bit. For a symbol b^n_{pcx}, where $p, c, x \in \{0, 1\}$, we say that p is the position-component, c the carry-component, and x the next-component of b^n_{pcx}. In a word over $\Sigma_{\mathcal{E}}$ representing the evolution of the 2^n-bit counter, the three components of symbols that are exactly 2^n positions apart will represent the position, carry, and next bits in the same position of two successive configurations of the counter. By using the highlight bits of the encoding technique of Theorem 7 we can enforce the correct relationships between such symbols. Hence, we can define $\mathcal{E}^n = \{B^n_{000}, \dots, B^n_{111}\}$ and E^n_0 in such a way that a word $w = b^n_{p_0c_0x_0} \cdots b^n_{p_mc_mx_m}$ is a rewriting of E^n_0 wrt $\mathcal{E}^n$ if and only if the bit vector $p_0 \cdots p_m$ represented by the position-components of w is of the form $w_0 \cdots w_{2^{2^n}-1}$, where w_j is the 2^n-bit representation of j.

Using pumping arguments it is easy to see that any regular expression or automaton describing such a rewriting has to be of length at least 2^{2^n}. Indeed, assume there is a regular expression or automaton R of size less than 2^{2^n} describing the rewriting. Then, since any nonempty regular expression or automaton accepts at least one word of length less than or equal to its size, R accepts also a word w' of length less than 2^{2^n}, contradicting the hypothesis that R is a correct rewriting of E^n_0 wrt $\mathcal{E}^n$. □

3.2.2 Existence of an exact rewriting

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

We consider the tiling problem $T = (\Delta, H, V, t_S, t_F, t_L, t_R, C_{2ES})$, where t_S, t_F, t_L, and t_R are distinguished tile types in Δ such that $(t_R, t_L) \in H$, and for a given number n in unary, C_{2ES} requires to tile a region of size $O(2^{2^n}) \times k$, for some number k, in such a way that: (i) the left bottom tile of the region is of type t_S, (ii) all other tiles on the left border are of type t_L, (iii) the right upper tile is of type t_F, and (iv) all other tiles on the right border are of type t_R. Using a reduction from acceptance of 2EXPSPACE Turing machines analogous to the one in [vEB97], it can be shown that this tiling problem is 2EXPSPACE-complete.

We exploit such a tiling problem to prove the 2EXPSPACE lower bound of the problem of verifying the existence of an exact rewriting. That is, given an instance T of the above tiling problem and a number n, we construct a regular expression E_0 and a set $\mathcal{E}$ of regular expressions such that a T-tiling exists if and only if there is an exact rewriting of E_0 wrt $\mathcal{E}$. Each row of a T-tiling is of doubly exponential length in n. We describe such a tiling as a word over Δ, and to "check" the vertical adjacency conditions we need to compare the types of tiles that are a doubly exponential distance apart, which requires "yardsticks" of such length. Fortunately, we have seen in the proofs of Theorems 7 and 8 how to construct such yardsticks.

We directly exploit the construction described in Theorem 8 to encode a 2^n bit counter, and obtain a regular expression E_0^C and a set $\mathcal{E}^C$ of regular expressions, all over an alphabet $\Sigma^C = \{0, 1, \$\} \cup \Delta^C$. Let $re^C(\cdot)$ be the mapping that associates to each symbol in a suitable alphabet $\Sigma_\mathcal{E}^C$ a regular expression in $\mathcal{E}^C$. Then for a word w over $\Sigma_\mathcal{E}^C$ we have that $exp_\Sigma(w) \subseteq L(E_0^C)$ precisely when $w = w_C$, where w_C is the word that describes the 2^{2^n} successive bit configurations (for the position, the carry and the next bits) of the 2^n bit counter. In particular, since each bit configuration is of length 2^n, we have that w_C is of length $2^n \cdot 2^{2^n}$, which is precisely what we need. We will use E_0^C and $\mathcal{E}^C$ to construct regular expressions that detect errors in T-tilings with rows of length exactly $1 + 2^n \cdot 2^{2^n}$.

Let $\tilde{\Delta} = \{\tilde{t} \mid t \in \Delta\}$, where Δ is the set of tile types of T. We take Σ to be $\Sigma^C \cup \tilde{\Delta} \cup \Delta$ and $\Sigma_\mathcal{E}$ to be $\Sigma_\mathcal{E}^C \cup \tilde{\Delta}$. The set $\mathcal{E}$ of regular expressions used for the rewriting is obtained by taking $re(e) = re^C(e) + \Delta$, for each $e \in \Sigma_\mathcal{E}^C$, and $re(\tilde{t}) = \tilde{t} + t$, for each $\tilde{t} \in \tilde{\Delta}$. Thus each symbol in $\Sigma_\mathcal{E}^C$ generates also all possible tile types in Δ, while each symbol in $\tilde{\Delta}$ generates itself and only the corresponding tile type.

We construct regular expressions E_0^V, E_0^H, E_0^S, E_0^F, E_0^L, and E_0^R, which are used to detect errors in candidate tilings. E_0^V is used to detect conflicts with respect to the vertical adjacency relation V, which arise between tiles that are $1 + 2^n \cdot 2^{2^n}$ symbols apart. E_0^H is used to detect conflicts with respect to the horizontal adjacency relation H, which arise between tiles that are directly adjacent. Note that since $(t_R, t_L) \in H$, also the last tile of a row and the first tile of the next row have to respect the horizontal adjacency condition. E_0^S, E_0^F, E_0^L, and E_0^R are used to detect tiles of the wrong type at the beginning and end, and on the left and right border respectively. All such tiles are at a known distance from the left bottom tile. The regular expressions are constructed in such a way that for a word w over $\Sigma_\mathcal{E}$ we have that:

- $exp_\Sigma(w) \subseteq L(E_0^V)$ precisely when w is in the form

  $$\Sigma_\mathcal{E}^{C*} \cdot \Big(\sum_{(t_1,t_2)\in\overline{V}} \tilde{t_1} \cdot \Sigma_\mathcal{E}^C \cdot w_C \cdot \tilde{t_2}\Big) \cdot \Sigma_\mathcal{E}^{C*}$$

  where $\overline{V}$ is the set of pairs of tiles that are not in V.

- $exp_\Sigma(w) \subseteq L(E_0^H)$ precisely when w is in the form

  $$\Sigma_\mathcal{E}^{C*} \cdot \Big(\sum_{(t_1,t_2)\in\overline{H}} \tilde{t_1} \cdot \tilde{t_2}\Big) \cdot \Sigma_\mathcal{E}^{C*}$$

  where $\overline{H}$ is the set of pairs of tiles that are not in H.

- $exp_\Sigma(w) \subseteq L(E_0^S)$ precisely when w is in the form

  $$\Big(\sum_{t\in\Delta\setminus\{t_S\}} \tilde{t}\Big) \cdot \Sigma_\mathcal{E}^{C*}$$

- $exp_\Sigma(w) \subseteq L(E_0^F)$ precisely when w is in the form

  $$(\Sigma_\mathcal{E}^C \cdot w_C)^* \cdot w_C \cdot \Big(\sum_{t\in\Delta\setminus\{t_F\}} \tilde{t}\Big)$$

- $exp_\Sigma(w) \subseteq L(E_0^L)$ precisely when w is in the form

  $$(\Sigma_\mathcal{E}^C \cdot w_C)^* \cdot \Sigma_\mathcal{E}^C \cdot w_C \cdot \Big(\sum_{t\in\Delta\setminus\{t_L\}} \tilde{t}\Big) \cdot \Sigma_\mathcal{E}^{C*}$$

- $exp_\Sigma(w) \subseteq L(E_0^R)$ precisely when w is in the form

$$(\Sigma_\mathcal{E}^C \cdot w_C)^* \cdot w_C \cdot (\sum_{t \in \Delta \setminus \{t_R\}} \tilde{t}) \cdot \Sigma_\mathcal{E}^C \cdot \Sigma_\mathcal{E}^{C*}$$

The construction of E_0^H and E_0^S is immediate. For the other regular expressions we need to construct a regular expression $E_0^{C\Delta}$, of size polynomial in n, whose rewriting is w_C. We make use of E_0^C and $\mathcal{E}^C$, but need to take into account that, wrt the construction in Theorem 8, now a symbol e in $\Sigma_\mathcal{E}^C$ generates not only all possible sequences of type $\$\cdot(0+1)^{3n+1}\cdot e$ (and hence of length $3n+3$) but also all symbols in Δ. We can however exploit the fact that E_0^C is composed of subexpressions that generate words of length $3n+3$ and thus obtain $E_0^{C\Delta}$ from E_0^C by simply adding the expression Δ to each such subexpression. Then we have for example that

$$E_0^V = (B^C + \Delta)^* \cdot (\sum_{(t_1,t_2)\in\overline{V}} (\tilde{t_1} + t_1) \cdot (B^C + \Delta) \cdot E_0^{C\Delta} \cdot (\tilde{t_2} + t_2)) \cdot (B^C + \Delta)^*$$

where B^C stands for the regular expression $\$\cdot(0+1)^{3n+1}\cdot\Delta^C$. The regular expressions E_0^F, E_0^L, and E_0^R are constructed in a similar way.

The regular expression $E_0^1 = E_0^V + E_0^H + E_0^S + E_0^F + E_0^L + E_0^R$ is such that a rewriting of E_0^1 generates only candidate tilings with some error (in addition to words containing also \$, 0, 1, the symbols in Δ^C, and at most two symbols in $\tilde{\Delta}$).

To encode the problem of the existence of an *exact* rewriting, we take E_0 to be $E_0^1 + \Delta^*$, i.e., E_0 expresses also all "candidate" tilings using the tile types in Δ. If no T-tiling exists, then every candidate tiling will have an error, and thus will already be generated by a rewriting of E_0^1. If, on the other hand, a T-tiling exists, such a tiling does not have an error and will not be generated by the rewriting of E_0^1, resulting in a non-exact rewriting. Notice that we cannot attempt to construct a rewriting of Δ^* separately, and the only way to get one is via the rewriting of E_0^1. This is due to the fact that, from the symbols in $\Sigma_\mathcal{E} = \Sigma_\mathcal{E}^C \cup \tilde{\Delta}$, each symbol e in $\Sigma_\mathcal{E}^C$ generates not only all symbols in Δ, but also sequences of type $\$\cdot(0+1)^{3n+1}\cdot e$, while each symbol $\tilde{t}$ in $\tilde{\Delta}$ generates besides t also $\tilde{t}$.

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. The upper bound proof is given in Theorem 6. The lower bound follows from the reduction from the 2EXPSPACE complete tiling problem described above, by observing that E_0 and all regular expressions in $\mathcal{E}$ are of size polynomial in T and n. □

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 $\mathcal{D}$. 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 $\mathcal{F}$

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

∎

[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].

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, we are interested in extending regular path queries to the so-called 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 [CDGLV99b, CDGLV99a].

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] Serge 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] Serge Abiteboul and Oliver 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.

[AGK99] Foto N. Afrati, Manolis Gergatsoulis, and Theodoros Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 435–452. Springer-Verlag, 1999.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[AV97] Serge Abiteboul and Victor 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] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 336–350, 1997.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan 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.

[Ber66] R. Berger. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, 66:1–72, 1966.

[BFW98] Peter Buneman, Wenfei Fan, and Scott 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] Catriel Beeri, Alon Y. Levy, and Marie-Christine 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] Peter 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] Diego Calvanese, Giuseppe De Giacomo, and Maurizio 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.

[CDGLV99a] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Vardi. Answering regular path queries using views. Technical Report 20-99, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.

[CDGLV99b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Vardi. Query answering using views for data integration over the web. In *2nd Int. Workshop on the Web and Databases (WebDB'99)*, pages 73–78, 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] Sara Cohen, Werner Nutt, and Alexander 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] Oliver M. Duschka and Michael 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.

[DG98] Oliver M. Duschka and Michael R. Genesereth. Query planning with disjunctive sources. In *Proc. of the AAAI-98 Workshop on AI and Information Integration*, 1998.

[FFK+98] Mary F. Fernandez, Daniela Florescu, Jaewoo Kang, Alon Y. Levy, and Dan 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] Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[FLS98] Daniela Florescu, Alon Levy, and Dan 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] Mary F. Fernandez and Dan 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] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh 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] Alberto Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[MS99] Tova Milo and Dan 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.

[PV99] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting using semistructured views. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 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] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey 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] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[vEB82] Peter van Emde Boas. Dominoes are forever. In *Proc. of 1st GTI Workshop*, Rheie Theoretische Informatik UGH Paderborn, pages 75–95, Paderborn (Germany), 1982.

[vEB97] Peter van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and Recursion Theory*, volume 187 of *Lecture notes in pure and applied mathematics*, pages 331–363. Marcel Dekker Inc., 1997.