Rewriting of Regular Path Queries

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

[1] Dip. 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] Dept. of Computer Science,
Rice University
P.O. Box 1892, Houston, TX 77251-1892, U.S.A
vardi@cs.rice.edu
http://www.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. On the other hand, 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 [13, 12]). 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 [1]. 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 [7, 10, 18, 20]. 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 [3, 8, 15]. 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.

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 [24, 2]: Given a query Q and k queries $Q_1, \dots, Q_k$ associated to the symbols $q_1, \dots, q_k$, respectively, generate a new query Q' over the alphabet $q_1, \dots, q_k$ 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) [17, 21], queries with aggregates [22, 11], recursive queries [14], and queries expressed in Description Logics [5]. Rewriting techniques for query optimization are described, for example, in [9, 4, 23], and in [16, 19] 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 $E_1, \dots, E_k$, checks whether we can re-express E_0 by a suitable combination of $E_1, \dots, E_k$. As noted in [19], 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.

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. For the sake of brevity, the proofs of the theorems are only sketched, and appear in the Appendix.

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 $expand_{\Sigma}(\ell)$ the language over Σ defined as follows

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

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 $expand_{\Sigma}(L(R)) \subseteq L(E_0)$.*

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 $expand_{\Sigma}(L(R')) \subseteq expand_{\Sigma}(L(R))$. 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.*

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 $s_j \in \rho'(s_i, e)$ iff $\exists w \in L(re(e))$ 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 a Σ-word $w_1 \cdots w_n$ that is rejected by A_d such that $w_i \in L(re(e_i))$ for $i = 1, \ldots, n$, then the $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n$ is accepted by A'. That is A' accepts a $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n$ iff there is a Σ-word in $expand_{\Sigma}(\{e_1 \cdots e_n\})$ that is rejected by A_d. Hence, $R_{\mathcal{E},E_0}$, being the complement of A', accepts a $\Sigma_{\mathcal{E}}$-word $e_1 \cdots e_n$ iff 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 $\mathcal{E}$.*

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.

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

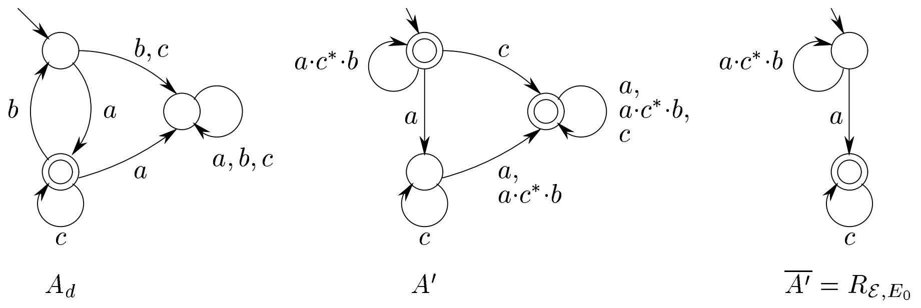

Fig. 1. Construction of the rewriting of $a{\cdot}(b{\cdot}a + c)^*$ wrt $\{a,\ a{\cdot}c^*{\cdot}b,\ c\}$

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 $expand_{\Sigma}(L(R_{\mathcal{E},E_0}))$, 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 $\mathcal{E}$ iff $L(A_d \cap \overline{B}) = \emptyset$.*

Corollary 1. *An exact rewriting of E_0 wrt $\mathcal{E}$ exists iff $L(A_d \cap \overline{B}) = \emptyset$.*

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 4. *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.*

With regard to the cost of verifying the existence of an exact rewriting, Corollary 1 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 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.

Theorem 5. *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.*

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 $expand_{\Sigma}(L(R)) = \emptyset$. 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 6. *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.*

Note that Theorem 6 implies that the upper bound established in Theorem 4 is essentially optimal. If we can generate maximal rewritings in, say, EXPTIME, then we could test emptiness in PSPACE, which is impossible by Theorem 6. We can get, however, an even sharper lower bound on the size of rewritings.

Theorem 7. *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}.*

The technique used in Theorem 6 turns out to be an important building block in the proof that Theorem 5 is also tight.

Theorem 8. *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.*

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.

4.1 Semi-structured data models and queries

All semi-structured data models share the characteristic that data are organized in a labeled graph, where the nodes represent objects, and the edges represent links between objects [7, 6, 1, 20]. 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[1] 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 [7] 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 [8, 1, 16, 19]. 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 $\mathcal{T}$) iff $\mathcal{T} \models \varphi_i(a_i)$, 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 $answer(L(Q), DB)$, where for a language ℓ over $\mathcal{F}$*

$$answer(\ell, DB) = \{(x, y) \mid \text{ there is a path } x \xrightarrow{a_1} x_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} y \text{ in } DB \text{ such that } a_1 \cdots a_n \in match(\ell)\}$$

[1] The theory is complete in the sense that for every closed formula φ, either $\mathcal{T}$ entails φ, or $\mathcal{T}$ entails $\neg\varphi$ [7].

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 $expand_\mathcal{F}(\ell)$ the language over $\mathcal{F}$ defined as follows

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

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, $answer(expand_\mathcal{F}(L(R)), DB) \subseteq answer(L(Q_0), DB)$, and is said to be (i)* maximal *if for each rewriting R' of Q_0 wrt $\mathcal{Q}$ we have that $answer(expand_\mathcal{F}(L(R')), DB) \subseteq answer(expand_\mathcal{F}(L(R)), DB)$, (ii)* exact *if $answer(expand_\mathcal{F}(L(R)), DB) = answer(L(Q_0), DB)$.*

Theorem 9. *R is a rewriting of Q_0 wrt $\mathcal{Q}$ iff $match(expand_\mathcal{F}(L(R))) \subseteq match(L(Q_0))$. Moreover, it is maximal iff for each rewriting R' of Q_0 wrt $\mathcal{Q}$ we have that $match(expand_\mathcal{F}(L(R'))) \subseteq match(expand_\mathcal{F}(L(R)))$, and it is exact iff $match(expand_\mathcal{F}(L(R))) = match(L(Q_0))$.*

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 9, 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 the 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 $s_j \in \rho^g(s_i, a)$ iff $s_j \in \rho(s_i, \varphi)$ 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 $s_j \in \rho'(s_i, q)$ iff $\exists w \in \mathit{match}(L(\mathit{rpq}(q)))$ 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 10. *The automaton $R_{\mathcal{Q},Q_0}$ is 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 $\mathit{expand}_{\mathcal{D}}(L(R_{\mathcal{Q}^g,Q_0^g}))$, 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 $w \in \mathit{match}(L(\mathit{rpq}(q)))$ 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 $Q = \mathit{rpq}(q)$ (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) iff (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'.

On the other hand, the construction of Q_0^g seems unavoidable, since formulae that satisfy more that one constant in $\mathcal{T}$ and that appear as labels of the transitions of Q_0, may hide a nondeterminism that is instead revealed when we consider 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 $\mathit{rpq}(q_1) = a$, and $\mathit{rpq}(q_2) = b$. 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 $\mathit{rpq}(q_3) = c$, 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. $\mathit{match}(\mathit{expand}_{\mathcal{F}}(L(R'))) \subset \mathit{match}(\mathit{expand}_{\mathcal{F}}(L(R)))$,
2. $\mathit{match}(L(R)) = \mathit{match}(L(R'))$ and R uses less additional elementary views than R',
3. $\mathit{match}(L(R)) = \mathit{match}(L(R'))$, R uses the same number of additional elementary views as R', and less additional atomic nonelementary views.
4. $\mathit{match}(L(R)) = \mathit{match}(L(R'))$, 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

We envision several directions for future work. First, we want to extend our rewriting techniques to the case of 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. Such queries ask for all n-tuples $a_1, \ldots, a_n$ of nodes such that, for each i, there is a path from a_i to a_{i+1} that matches Q_i. Second, we aim at investigating possible interesting subcases where the rewriting of regular (and generalized) path queries can be done more efficiently. Finally, we aim at defining cost models for path queries, refining the preference criteria to take into account such cost models, and developing techniques for choosing the best rewriting with respect to the new criteria.

References

1. Serge Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 1–18, 1997.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.

16. 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.
17. 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.
18. Alberto Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.