View-Based Query Processing for Regular Path Queries with Inverse

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

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

ABSTRACT

View-based query processing is the problem of computing the answer to a query based on a set of materialized views, rather than on the raw data in the database. The problem comes in two different forms, called *query rewriting* and *query answering*, respectively. In the first form, we are given a query and a set of view definitions, and the goal is to reformulate the query into an expression that refers only to the views. In the second form, besides the query and the view definitions, we are also given the extensions of the views and a tuple, and the goal is to check whether the knowledge on the view extensions logically implies that the tuple satisfies the query.

In this paper we address the problem of view-based query processing in the context of semistructured data, in particular for the case of regular-path queries extended with the inverse operator. Several authors point out that the inverse operator is one of the fundamental extensions for making regular-path queries useful in real settings. We present a novel technique based on the use of two-way finite-state automata. Our approach demonstrates the power of this kind of automata in dealing with the inverse operator, allowing us to show that both query rewriting and query answering with the inverse operator has the same computational complexity as for the case of standard regular-path queries.

1. INTRODUCTION

View-based query processing is the problem of computing the answer to a query based on a set of materialized views, rather than on the raw data in the database [32; 2]. It represents an abstraction for various data management problems arising in several contexts, including query optimization, query answering with incomplete information, data warehousing, and data integration.

To appear in Proceedings of the 19th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (**PODS 2000**).

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

Notice the difference between the two approaches. In query rewriting, query processing is divided in two steps, where the first re-expresses the query in terms of a given query language over the alphabet of the view names, and the second evaluates the rewriting over the view extensions. Obviously, it may happen that no rewriting in the target language exists that is equivalent to the original query. In this case, we are interested in computing a so-called *maximally contained rewriting*, i.e., an expression that captures the original query in the best way.

In query answering, we do not pose any limit on how queries are processed, and the only goal is to exploit all possible information, in particular the view extensions, to compute the answer to the query. Since the view extensions provide partial knowledge on the database, the problem is a special case of query answering with incomplete information [33].

A large number of results have been reported for both problems in the last years. Several papers investigate query rewriting for the case of conjunctive queries (with or without arithmetic comparisons) [25; 28], disjunctive views [5], queries with aggregates [30; 15], recursive queries and nonrecursive views [16], queries expressed in Description Logics [6], regular-path queries [10], and in the presence of integrity constraints [22; 17]. Rewriting techniques for query optimization are described, for example, in [13; 4; 31], and in [19; 26; 27] for the case of path queries in semistructured data.

A comprehensive framework for view-based query answering, as well as several interesting results, is presented in [21]. The framework considers various assumptions for interpreting the view extensions with respect to the corresponding definitions (closed, open, and exact view assumptions). In [2], an analysis of the data complexity of the problem under the different assumptions is carried out for the case where the views and the queries are expressed in terms of various languages (conjunctive queries, Datalog, first-order queries, etc.). In [11], a further distinction is proposed for characterizing the domain of the database (open vs. closed domain assumption), and the problem is studied for the case of regular-path queries. In [9], upper bound results are reported for view-based query answering in Description Logics.

In this paper, we address view-based query processing in the context of semistructured data. Semistructured data are modeled by labeled directed graphs, and have been introduced with the aim of capturing data in the web. The main difficulty arising in this context is that languages for querying semistructured data enable expressing regular-path queries [3; 8; 18]. A regular-path query asks for all pairs of nodes in the database connected by a path conforming to a regular expression, and therefore may contain a restricted form of recursion. Note that when the query contains unrestricted recursion, both view-based query rewriting and view-based query answering become undecidable, even when the views are not recursive [16].

To the best of our knowledge, the only known decidability results for view-based query processing in the case where the query and the views may contain recursion are reported in [10; 11].

In [10], a method for rewriting a regular-path query in terms of other regular-path queries is proposed. The method computes the rewriting in 2EXPTIME, and is able to check whether the computed rewriting is exact in 2EXPSPACE. It is also shown that checking whether there is a nonempty rewriting is EXPSPACE-complete, and verifying the existence of an exact rewriting is 2EXPSPACE-complete.

In [11], algorithms and complexity results for view based query answering for regular-path queries are presented. With respect to data complexity, the problem is co-NP-complete under all assumptions, whereas with respect to combined complexity it is co-NP-complete under the closed domain assumption, and PSPACE-complete under the open domain assumption.

While regular-path queries represent the core of any query language for semistructured data, their expressive power is limited. Several authors point out that extensions are required for making them useful in real settings (see for example [7; 8; 26]). One fundamental step in this direction is to enrich regular-path queries with the *inverse* operator. The inverse operator is essential for expressing navigations in the database that traverse the edges both backward and forward [14].

The existing approaches to view-based query processing for regular-path queries do not easily extend to the case with the inverse operator. The goal of this paper is to present a novel technique for this case, based on the use of two-way finite-state automata. Unlike standard finite-state automata, two-way automata are equipped with a head that can move back and forth on the input string. A transition of this kind of automata indicates not only the new state, but also whether the head should move left, right, or stay in place.

Generally speaking, the method presented here works by searching for counterexamples. In the case of rewriting, a counterexample demonstrates that a certain rewriting is not good for the query, whereas in the case of answering, the counterexample demonstrates that a given tuple is not an answer to the query. The basic idea underlying our techniques is to encode candidate counterexamples as words over a suitable alphabet, and then use two-way automata to check that a candidate is indeed a counterexample. The ability of two-way automata to move back and forth on words is essential in capturing the computation done when processing a regular-path query with inverse over a database, and therefore in checking candidate counterexamples. Indeed, the power of two-way automata in dealing with the inverse operator allows us to derive the following results:

- View-based query rewriting can be done with the same computational complexity as for the case without inverse. Indeed, we describe a method based on two-way automata that computes the maximal contained rewriting in 2EXPTIME, and checks whether the rewriting is exact in 2EXPSPACE.

- We provide algorithms for view-based query answering under different assumptions. With respect to data complexity, the algorithm works in co-NP in all cases. With respect to combined complexity, the algorithm works in co-NP under the closed domain assumption, and in PSPACE under the open domain assumption. Once again, by exploiting two-way automata, we are able to show that the complexity of the problem does not increase by adding the inverse operator.

Besides the specific results, our method provides the basis for using two-way automata in reasoning about complex queries. Indeed, the idea of encoding candidate counterexamples as words, and then checking candidates with two-way automata, can be adapted to reasoning about queries of more general forms and in other logical formalisms (e.g., temporal and dynamic logics with a backward modality). A first step in this direction is presented in [12], where the technique is used to show that containment of conjunctive regular-path queries with inverse is EXPSPACE-complete.

The paper is organized as follows. Section 2 defines regular-path queries with inverse, and Section 3 introduces the notion of two-way automata. Sections 4 and 5 present the results for query rewriting and query answering, respectively. Section 6 concludes the paper.

2. REGULAR-PATH QUERIES WITH INVERSE

Following the usual approach in semistructured data, we define a *(semi-structured) database* (DB) as a graph whose nodes represent objects, and whose edges are labeled by elements from a given alphabet Σ', which we assume to be finite [7; 1]. Different nodes represent different objects, and an edge from node x to node y labeled by p, denoted $x \xrightarrow{p} y$, represents the fact that relation p holds between the object represented by x and the object represented by y.

The basic querying mechanism on a DB is that of *regular-path queries* (RPQs). An RPQ E is expressed as a regular expression or a finite-state automaton over Σ', and computes the set of pairs of nodes of the DB connected by a path that conforms (see formal definition below) to the regular language $L(E)$ defined by E.

As we said in the introduction, we consider queries that extend regular-path queries with the *inverse* operator. Formally, let $\Sigma = \Sigma' \cup \{p^- \mid p \in \Sigma'\}$ be the alphabet including a new symbol p^- for each p in Σ'. Intuitively, p^- denotes the inverse of the binary relation p. If $r \in \Sigma$, then we use r^- to mean the *inverse* of r, i.e., if r is p, then r^- is p^-, and if r is p^-, then r^- is p.

Regular-path queries with inverse (RPQIs) are expressed by means of regular expressions or finite-state automata over Σ. Thus, in contrast with RPQs, RPQIs may use also the inverse p^- of p, for each $p \in \Sigma'$.

When evaluated over a DB $\mathcal{B}$, an RPQI E computes the set $ans(E, \mathcal{B})$ of pairs of nodes connected by a semipath that conforms to the regular language $L(E)$ defined by E. A *semipath* in $\mathcal{B}$ from x to y (labeled with $r_1 \cdots r_q$) is a sequence of the form $(y_1, r_1, y_2, \ldots, y_q, r_q, y_{q+1})$, where $q \geq 0$, $y_1 = x$, $y_{q+1} = y$, and for each y_i, r_i, y_{i+1}, $r_i \in \Sigma$, and, if $r_i = p$ then $y_i \xrightarrow{p} y_{i+1}$, and if $r_i = p^-$ then $y_{i+1} \xrightarrow{p} y_i$ is in $\mathcal{B}$. We say that a semipath $(y_1, r_1, \ldots, r_q, y_{q+1})$ *conforms to* E if $r_1 \cdots r_q \in L(E)$. A semipath is said to be *simple* if no node appears more than once in the corresponding sequence.

A word $w = r_1 \cdots r_q$ over Σ *satisfies* E iff $(x, y) \in ans(E, \mathcal{B}_w)$, where $\mathcal{B}_w$ is constituted by a single semipath $(x, r_1, \ldots, r_q, y)$ labeled with w. Note that w may satisfy E even if $w \notin L(E)$, since we may obtain the semipath from x to y conforming to E by moving back and forth on the nodes of $\mathcal{B}_w$.

EXAMPLE 1. *Let us consider a DB of software modules, where the relation* **hassubmodule** *connects m_1 to m_2 if m_2 is a module defined in m_1, and the relation* **containsvar** *connects m to v if v is a variable defined in module m. The RPQI*

$$(\texttt{hassubmodule}^-)^* \cdot (\texttt{containsvar} \cup \texttt{hassubmodule})$$

computes the pairs (m, x) such that x (either a variable, or a module) is visible in module m, according to the usual visibility rules of Algol-like languages.

3. TWO-WAY AUTOMATA

A *two-way automaton* [23] $A = (\Gamma, S, S_0, \rho, F)$ consists of an alphabet Γ, a finite set of states S, a set of initial states $S_0 \subseteq S$, a transition function

$$\rho : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$$

and a set of accepting states $F \subseteq S$.

Intuitively, a transition indicates both the new state of the automaton, and whether the head reading the input should move left (-1), right (1), or stay in place (0). If for all $s \in S$ and $a \in \Gamma$ we have that $\rho(s, a) \subseteq S \times \{1\}$, then the automaton is a traditional nondeterministic finite-state automaton (also called *one-way automaton*).

A *configuration* of A is a pair consisting of a state and a position represented as a natural number. A *run* is a sequence of configurations. The sequence $((s_0, j_0), \ldots, (s_m, j_m))$ is a run of A on a word $w = a_0, \ldots, a_{n-1}$ in Γ^* if $s_0 \in I$, $j_0 = 0$, $j_m \leq n$, and for all $i \in \{0, \ldots, m-1\}$, we have that $0 \leq j_i < n$, and there is some $(t, k) \in \delta(s_i, a_{j_i})$ such that $s_{i+1} = t$ and $j_{i+1} = j_i + k$. This run is *accepting* if $j_m = n$ and $s_m \in F$. *A accepts w* if it has an accepting run on w. The set of words accepted by A is denoted $L(A)$.

It is well known that two-way automata define regular languages, and that, given a two-way automaton A with n states, one can construct a one-way automaton B_1 with $O(2^{n \log n})$ states such that $L(B_1) = L(A)$, and a one-way automaton B_2 with $O(2^n)$ states such that $L(B_2) = \Gamma^* - L(A)$ [23; 35].

The ability of two-way automata to scan words in both directions allows us to use them to evaluate RPQIs over DBs. To gain some intuition on how two-way automata capture computations of RPQIs over DBs, we show how to construct, with linear blow-up, from an RPQI E a two-way automaton A_E that accepts all words $w\$ \in \Sigma^* \cdot \$$ such that w satisfies E. To construct A_E, we assume that E is represented as a finite-state (one-way) automaton $E = (\Sigma, S, I, \delta, F)$ over the alphabet Σ. Then $A_E = (\Sigma_A, S_A, I, \delta_A, \{s_f\})$, where

- $\Sigma_A = \Sigma \cup \{\$\}$,
- $S_A = S \cup \{s_f\} \cup \{s^{\leftarrow} \mid s \in S\}$, and
- δ_A is defined as follows:
    1. $(s^{\leftarrow}, -1) \in \delta_A(s, \ell)$, for each $s \in S$ and $\ell \in \Sigma_A$. Such transition makes the automaton ready to scan one step backward by placing it in "backward mode".
    2. $(s_2, 1) \in \delta_A(s_1, r)$ and $(s_2, 0) \in \delta_A(s_1^{\leftarrow}, r^-)$, for each transition $s_2 \in \delta(s_1, r)$ of E. These transitions correspond to the transitions of E that are performed forward or backward according to the current "scanning mode".
    3. $(s_f, 1) \in \delta_A(s, \$)$, for each $s \in F$.

Notice that the \$ symbol, which acts as a terminator in the words accepted by A_E, is necessary to allow the automaton to continue backwards, even when it has reached the end of the input word.

THEOREM 2. *Let E be an RPQI, and A_E the two-way automaton constructed from E as specified above. Then w satisfies E iff $w\$ \in L(A_E)$.*

4. VIEW-BASED QUERY REWRITING

Let E_0 be an RPQI, and $\mathcal{E} = \{E_1, \ldots, E_k\}$ a finite set of RPQIs over the alphabet Σ. We assume that associated with $\mathcal{E}$ we always have an alphabet $\Sigma'_{\mathcal{E}}$ containing exactly one symbol for each query in $\mathcal{E}$, and we denote the query associated to the symbol $p \in \Sigma'_{\mathcal{E}}$ with $def(p)$.

Intuitively, we want to rewrite E_0 in terms of the views $\mathcal{E} = \{E_1, \ldots, E_k\}$, where p_i is the name of the i-th view, and $def(p_i) = E_i$ is its definition. The only constraint we impose on the rewriting is that it is an expression denoting a set of words in the alphabet $\Sigma_{\mathcal{E}} = \Sigma'_{\mathcal{E}} \cup \{p^- \mid p \in \Sigma'_{\mathcal{E}}\}$, i.e., the target query language for the rewriting is a formalism for specifying languages over $\Sigma_{\mathcal{E}}$. This means that in the rewriting, the inverse operator can be applied to symbols in $\Sigma'_{\mathcal{E}}$, and therefore we have to specify the semantics of such applications.

For this purpose, we extend the function def to all symbols in $\Sigma_{\mathcal{E}}$ as follows: if $p \in \Sigma'_{\mathcal{E}}$, then $def(p^-)$ is the regular expression $inv(def(p))$ over Σ, where inv is defined as follows:

- $inv(a) = a^-$,
- $inv(a^-) = a$,
- $inv(e_1 \cdot e_2) = inv(e_2) \cdot inv(e_1)$,
- $inv(e_1 \cup e_2) = inv(e_1) \cup inv(e_2)$,
- $inv(e^*) = inv(e)^*$.

Given a language ℓ over $\Sigma_{\mathcal{E}}$, we denote by $expand_\Sigma(\ell)$ the language over Σ defined as follows

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

DEFINITION 3. *Let R be any language over $\Sigma_{\mathcal{E}}$. We say that R is a* rewriting of E_0 *wrt $\mathcal{E}$ if $ans(expand_\Sigma(R), \mathcal{B}) \subseteq ans(E_0, \mathcal{B})$, for every DB $\mathcal{B}$.*

The following theorem is a useful characterization of "bad" rewritings, based on the notion of satisfaction of a query by a word, as defined in Section 2.

THEOREM 4. *Let R be a language over $\Sigma_{\mathcal{E}}$. Then R is not a rewriting of E_0 wrt $\mathcal{E}$ iff there exists a word in $expand_\Sigma(R)$ that does not satisfy E_0.*

PROOF. If R is not a rewriting of E_0 wrt $\mathcal{E}$, then there exists a DB $\mathcal{B}$ with nodes x, y such that $(x, y) \in ans(expand_\Sigma(R), \mathcal{B})$ and $(x, y) \notin ans(E_0, \mathcal{B})$. This means that there is a semipath from x to y in $\mathcal{B}$ labeled by a word in $expand_\Sigma(R)$ that does not satisfy E_0.

If there exists a word $r_1 \cdots r_q$ in $expand_\Sigma(R)$ that does not satisfy E_0, then the DB $\mathcal{B}$ constituted by the single semipath $(x, r_1, \ldots, r_q, y)$ is such that $(x, y) \in ans(expand_\Sigma(R), \mathcal{B})$ and $(x, y) \notin ans(E_0, \mathcal{B})$. $\square$

We are interested both in maximal rewritings, i.e., rewritings that capture in the best possible way the query E_0, and in exact rewritings, which capture exactly the query E_0.

DEFINITION 5. *Let R be a rewriting of E_0 wrt $\mathcal{E}$. R is said to be* maximal *if for each rewriting R' of E_0 wrt $\mathcal{E}$ we have $ans(expand_\Sigma(R'), \mathcal{B}) \subseteq ans(expand_\Sigma(R), \mathcal{B})$, for every DB $\mathcal{B}$. R is said to be* exact *if for every DB $\mathcal{B}$, $ans(expand_\Sigma(R), \mathcal{B}) = ans(E_0, \mathcal{B})$.*

Our technique is based on characterizing the words in $\Sigma_{\mathcal{E}}$ that do not belong to a rewriting of E_0 wrt $\mathcal{E}$. By Theorem 4, these are the words $w \in \Sigma^*_{\mathcal{E}}$ such that some word in $expand_\Sigma(\{w\})$ does not satisfy E_0. Based on the use of two-way automata, we define an automaton that accepts exactly such words, and then complement such automaton to get the rewriting.

Let A_0 be the one-way automaton over Σ corresponding to the query E_0, and let $\Sigma_{\mathcal{E}}$ be $\{e_1, \ldots, e_n\}$. We consider words over $\Sigma \cup \Sigma_{\mathcal{E}} \cup \{\$, :\}$ of the form

$$\$e_{i_1}\!:\!w_{i_1}\$\cdots\$e_{i_m}\!:\!w_{i_m}\$$$

with $e_{i_j} \in \Sigma_{\mathcal{E}}$, and $w_{i_j} \in \Sigma^*$.

Following the idea described in Section 3, we construct a two-way automaton A_1 that accepts words of the above form such that $w_{i_1} \cdots w_{i_m}$ satisfies E_0. Let A_2 be a one-way automaton that complements A_1. Let A_3 be a one-way automaton that accepts a word of the form

$$\$e_{i_1}\!:\!w_{i_1}\$\cdots\$e_{i_m}\!:\!w_{i_m}\$$$

iff for every i_j, the word w_{i_j} is in $L(def(e_{i_j}))$, i.e., iff the word $w_{i_1} \cdots w_{i_m}$ is in $expand_\Sigma(\{e_{i_1} \cdots e_{i_m}\})$. Now consider the automaton $A_3 \cap A_2$. A word

$$\$e_{i_1}\!:\!w_{i_1}\$\cdots\$e_{i_m}\!:\!w_{i_m}\$$$

is accepted by this automaton iff the word $w_{i_1} \cdots w_{i_m}$ is in $expand_\Sigma(\{e_{i_1} \cdots e_{i_m}\})$ and does not satisfy E_0. Let A_4 accept all words $e_{i_1} \cdots e_{i_m}$ that are projections on the e_{i_j}'s of the words

$$\$e_{i_1}\!:\!w_{i_1}\$\cdots\$e_{i_m}\!:\!w_{i_m}\$$$

that are accepted by $A_3 \cap A_2$. By construction, A_4 accepts all words $e_{i_1} \cdots e_{i_m}$ such that there is a word in $expand_\Sigma(\{e_{i_1} \cdots e_{i_m}\})$ that does not satisfy E_0. Finally, let $R_{\mathcal{E},E_0}$ be the complement of A_4. By virtue of the above construction, $R_{\mathcal{E},E_0}$ accepts all words $e_{i_1} \cdots e_{i_m}$ such that every word in $expand_\Sigma(\{e_{i_1} \cdots e_{i_m}\})$ satisfies E_0.

THEOREM 6. *$R_{\mathcal{E},E_0}$ is a maximal rewriting of E_0 wrt $\mathcal{E}$.*

PROOF. By exploiting Theorem 4, it is easy to show that $R_{\mathcal{E},E_0}$ is a rewriting of E_0 wrt $\mathcal{E}$. To see that $R_{\mathcal{E},E_0}$ is maximal, consider a word $e_{i_1} \cdots e_{i_m} \in \Sigma_{\mathcal{E}}$ that is not accepted by $R_{\mathcal{E},E_0}$. Obviously, $e_{i_1} \cdots e_{i_m}$ is accepted by A_4, and therefore there is a word $\$e_{i_1}{:}w_{i_1}\$ \cdots \$e_{i_m}{:}w_{i_m}\$$ that is accepted by $A_3 \cap A_2$. It follows that $expand_{\Sigma}(\{e_{i_1} \cdots e_{i_m}\})$ contains a word that does not satisfy E_0. By Theorem 4, this in turn implies that $e_{i_1} \cdots e_{i_m}$ does not belong to a rewriting of E_0 wrt $\mathcal{E}$. □

Observe that the only constraint we put on the rewriting is that it defines a language over $\Sigma_{\mathcal{E}}$. Theorem 6 shows that the language over $\Sigma_{\mathcal{E}}$ (and therefore also the language over Σ) defined by the maximal rewriting is regular (indeed, $R_{\mathcal{E},E_0}$ is a finite-state automaton).

THEOREM 7. *The problem of generating the maximal rewriting of an RPQI wrt a set of RPQIs is in 2EXPTIME.*

PROOF. The size of A_1 is polynomial in the size of E_0, the size of A_2 is exponential in the size of A_1, and the size of A_3 is polynomial in the size of $\mathcal{E}$. Finally, the size of A_4 is polynomial in the size of A_3 and A_2. The claim follows from the fact the size of $R_{\mathcal{E},E_0}$ is exponential in the size of A_4, and therefore double exponential in the size of E_0, and exponential in the size of $\mathcal{E}$. □

By exploiting the techniques and the results in [10], it is easy to prove the following theorem, which shows that our method for computing the maximal rewriting of an RPQI is essentially optimal.

THEOREM 8. *The problem of verifying the existence of a nonempty rewriting of an RPQI E_0 wrt a set $\mathcal{E}$ of RPQIs is EXPSPACE-complete.*

Next we address the problem of verifying whether the rewriting $R_{\mathcal{E},E_0}$ of E_0 wrt $\mathcal{E}$ is exact. We proceed as in [10], and construct a one-way automaton B that accepts $expand_{\Sigma}(R_{\mathcal{E},E_0})$, by replacing each edge labeled by e_i in $R_{\mathcal{E},E_0}$ by a one-way automaton A_i such that $L(A_i) = L(def(e_i))$. We then check whether $ans(L(A_0),\mathcal{B}) \subseteq ans(L(B),\mathcal{B})$, for every DB $\mathcal{B}$. Using the techniques described in [12], the following result can be proven.

THEOREM 9. *The problem of verifying the existence of an exact rewriting of an RPQI E_0 wrt a set $\mathcal{E}$ of RPQIs is 2EXPSPACE-complete.*

5. VIEW-BASED QUERY ANSWERING

As pointed out in [2; 21; 24; 11], the problem of view-based query answering comes in different forms, depending on various assumptions about how accurate is the knowledge on both the objects of the DB, and the pairs satisfying the views.

Consider a DB that is accessible only through a set of views $V_1, \ldots, V_k$, and suppose we want to answer an RPQI only on the basis of our knowledge on the views. Specifically, associated to each view V_i we have

- its definition $def(V_i)$ in terms of an RPQI;
- information about its extension in terms of a set $ext(V_i)$ of pairs of objects[1],
- a specification $as(V_i)$ of which *assumption* to adopt for the view V_i in interpreting $ext(V_i)$ with respect to the answer set of $def(V_i)$.

The possible assumptions on views are:

- *Sound View Assumption* (SVA). We say that a view V_i is *sound* (satisfies SVA) with respect to a DB $\mathcal{B}$, if $ext(V_i) \subseteq ans(def(V_i),\mathcal{B})$.
- *Complete View Assumption* (CVA). We say that a view V_i is *complete*, (satisfies CVA) with respect to a DB $\mathcal{B}$, if $ext(V_i) \supseteq ans(def(V_i),\mathcal{B})$.
- *Exact View Assumption* (EVA). We say that a view V_i is *exact* (satisfies EVA) with respect to a DB $\mathcal{B}$, if $ext(V_i) = ans(def(V_i),\mathcal{B})$.

We say that a DB $\mathcal{B}$ is *consistent with a view* V_i if V_i satisfies the assumption $as(V_i)$ with respect to $\mathcal{B}$.

We observe that, complete views can be reformulated in terms of exact views, by exploiting union in RPQIs [11]. Hence, in the following, we will focus on sound and exact views only.

With respect to the information available on the objects in the DB, one can further distinguish between: *Closed Domain Assumption* (CDA) and *Open Domain Assumption* (ODA). Under CDA, the exact set of objects in the DB coincides with the set of objects that appear in the view extensions, while under ODA the DB may contain additional objects. Formally, we call $\mathcal{D}_V$ the set of objects appearing in $ext(V_1) \cup \cdots \cup ext(V_k)$, and we say that a DB $(\mathcal{D},\mathcal{E})$ is *consistent with $\mathcal{D}_V$ under CDA (resp. ODA)* if $\mathcal{D}_V$ is equal to (resp. is a subset of) $\mathcal{D}$.

DEFINITION 10. *Given (i) $def(V_i)$, $ext(V_i)$, and $as(V_i)$, for $1 \leq i \leq k$; (ii) a query Q; (iii) a pair of objects $c,d \in \mathcal{D}$,* view-based query answering under CDA (resp. ODA) *consists in deciding whether $(c,d) \in ans(Q,\mathcal{B})$, for every $\mathcal{B}$ that is consistent with $\mathcal{D}_V$ under CDA (resp. ODA), and consistent with $V_1, \ldots, V_k$.*

[1]We assume that objects are represented by constants and we adopt the *unique name assumption* [29], i.e., different constants denote different objects, and therefore different nodes.

Assumption on domain	Assumption on views	Complexity		
		data	*expression*	*combined*
closed	all sound	*co-NP*	*co-NP*	*co-NP*
	all exact	*co-NP*	*co-NP*	*co-NP*
	arbitrary	*co-NP*	*co-NP*	*co-NP*
open	all sound	*co-NP*	*PSPACE*	*PSPACE*
	all exact	*co-NP*	*PSPACE*	*PSPACE*
	arbitrary	*co-NP*	*PSPACE*	*PSPACE*

Table 1: Summary of complexity results for view-based query answering (all bounds are tight)

We observe that view-based query answering can be interpreted as checking whether (c, d) is a *certain answer* to Q [2]. On the other hand, we may be interested in checking whether (c, d) is a *possible answer* to Q, i.e., whether $(c, d) \in ans(Q, \mathcal{B})$, for some $\mathcal{B}$ that is consistent with the views. Following the method in [11], it is possible to show that computing possible answers can be reduced to computing certain answers. Therefore, in the following we concentrate on certain answers.

The complexity of the problem can be measured in three different ways [34]:

- *data complexity*, as a function of the size of $ext(V_1) \cup \cdots \cup ext(V_k)$;
- *expression complexity*, as a function of the size of Q and of the expressions $def(V_1), \ldots, def(V_k)$;
- *combined complexity*, as a function of the size of both $ext(V_1) \cup \cdots \cup ext(V_k)$ and the expressions $Q, def(V_1), \ldots, def(V_k)$.

5.1 Closed domain assumption

Under the CDA, the techniques for view-based query answering for RPQs in [11] extend directly to RPQIs. Indeed, such techniques are based on guessing a DB $\mathcal{B}$ and then checking that $\mathcal{B}$ is consistent with the views and does not satisfy the query. Such check basically requires to evaluate each view and the query on $\mathcal{B}$. It is easy to see that the presence of the inverse operator does not influence the complexity of the evaluation.

THEOREM 11. *View-based query answering under* CDA *for RPQIs is co-NP-complete wrt data complexity, expression complexity, and combined complexity.*

5.2 Open domain assumption

Under ODA, the presence of the inverse operator requires introducing novel techniques. As we said in the introduction, the techniques we present are based on searching for a "counterexample DB" represented in a linearized form as a special word, and using two-way automata to check that candidate counterexample DBs satisfy all required conditions.

First of all, it can be shown that it is sufficient to restrict the search for counterexamples on canonical DBs.

DEFINITION 12. *Let $\mathcal{D}_V$ be the set of objects appearing in the extensions of the views. A DB is called* canonical *if it is composed of a set of simple semipaths $a \xrightarrow{r_1} x_1 \cdots x_{n-1} \xrightarrow{r_n} b$, where $a, b \in \mathcal{D}_V$ and $x_1, \ldots, x_{n-1}$ are not in $\mathcal{D}_V$ and do not occur in any other semipath in the set.*

THEOREM 13. *Given an RPQI E, a set of views, and a DB $\mathcal{B}$ that is consistent with the views and such that $(c, d) \notin ans(E, \mathcal{B})$, there exists a canonical DB $\mathcal{B}'$ that is consistent with the views and such that $(c, d) \notin ans(E, \mathcal{B}')$.*

The importance of canonical DBs lies in the fact that they are suitable for being linearized. Each canonical DB $\mathcal{B} = (\mathcal{D}, \mathcal{E})$ can be represented by a word $w_\mathcal{B}$ over the alphabet $\Sigma_A = \Sigma \cup \mathcal{D}_V \cup \{\$\}$, which has the form

$$\$d_1 w_1 d_2 \$ d_3 w_2 d_4 \$ \cdots \$ d_{2m-1} w_m d_{2m} \$$$

where $d_1, \ldots, d_{2m}$ range over $\mathcal{D}_V$, $w_i \in \Sigma^+$, and the \$ acts as a separator. Specifically, $w_\mathcal{B}$ consists of one subword $d_{2i-1} w_i d_{2i}$, for each simple semipath in $\mathcal{B}$ from d_{2i-1} to d_{2i} conforming to w_i. Observe that the same DB can be represented by several words that differ in the direction considered for the semipaths and in the order in which the subwords corresponding to the semipaths appear.

We show now how a two-way automaton $A_{(E,a,b)}$ can be used for the fundamental task of verifying whether a pair (a, b), with $a, b \in \mathcal{D}_V$, is in the answer set of an RPQI E over a linearized DB. The automaton $A_{(E,a,b)}$ accepts a word $w_\mathcal{B}$ over the alphabet Σ_A representing a DB $\mathcal{B}$ iff $(a, b) \in ans(E, \mathcal{B})$. To do so we exploit not only the ability of two-way automata to move on the word both forward and backward (see Section 3), but also the ability to "jump" from one position in the word representing a node to any other position (either preceding or succeeding) representing the same node. These two capabilities ensure that the automaton evaluating the RPQI on the word simulates exactly the evaluation of the query on the DB.

To construct $A_{(E,a,b)}$, we assume that E is represented as a one-way automaton $E = (\Sigma, S, I, \delta, F)$ over the alphabet Σ. Then $A_E = (\Sigma_A, S_A, \{s_0\}, \delta_A, \{s_f\})$, where

- $S_A = S \cup \{s_0, s_f\} \cup \{s^{\leftarrow} \mid s \in S\} \cup S \times \mathcal{D}$, and
- δ_A contains the following transitions[2]:

[2]The transitions defined by items 1 and 2 are the same as those defined by items 1 and 2 in Section 3, considering the current definition of Σ_A.

1. $(s^{\leftarrow}, -1) \in \delta_A(s, \ell)$, for each $s \in S$ and $\ell \in \Sigma \cup \mathcal{D}_V$. At any point such transition makes the automaton ready to scan one step backward by placing it in "backward mode".
2. $(s_2, 1) \in \delta_A(s_1, r)$ and $(s_2, 0) \in \delta_A(s_1^{\leftarrow}, r^-)$, for each transition $s_2 \in \delta(s_1, r)$ of E. These transitions correspond to the transitions of E which are performed forward or backward according to the current "scanning mode".
3. $(s_0, 1) \in \delta_A(s_0, \ell)$, for each $\ell \in \Sigma_A$, and also $(s, 0) \in \delta_A(s_0, a)$ for each $s \in I$. These transitions place the head of the automaton in some randomly chosen occurrence of a in the input string and set the state of the automaton to some randomly chosen initial state of E.
4. for each $s \in S$ and each $d \in \mathcal{D}_V$

$$\begin{array}{rcll} ((s,d),0) & \in & \delta_A(s,d) & \\ ((s,d),0) & \in & \delta_A(s^{\leftarrow},d) & \\ ((s,d),1) & \in & \delta_A((s,d),\ell), & \text{for each } \ell \in \Sigma_A \\ ((s,d),-1) & \in & \delta_A((s,d),\ell), & \text{for each } \ell \in \Sigma_A \\ (s,0) & \in & \delta_A((s,d),d) & \\ (s,1) & \in & \delta_A(s,d) & \end{array}$$

   Whenever the automaton reaches a symbol representing a node d (first and second clause), it may enter into "search (for d) mode" and move to any other occurrence of d in the word. Then the automaton exits search mode (second last clause) and continues its computation either forward (last clause) or backward (see item 2).
5. $(s_f, 0) \in \delta_A(s, b)$, for each $s \in F$ and $(s_f, 1) \in \delta_A(s_f, \ell)$, for each $\ell \in \Sigma_A$. These transitions place the automaton in the final state when it reads b in a final state of E. Then the head moves to the end of the input string to accept.

The following theorem characterizes the relationship between an RPQI and the corresponding two-way automaton.

THEOREM 14. *Let Q be an RPQI, $\mathcal{B}$ a canonical DB, $w_{\mathcal{B}}$ the word representing $\mathcal{B}$, and a, b two objects in $\mathcal{D}_V$. Then $A_{(Q,a,b)}$ accepts $w_{\mathcal{B}}$ iff $(a,b) \in ans(Q,\mathcal{B})$.*

Given a set of views $V_1, \dots, V_n$, an RPQI Q, and objects $c, d \in \mathcal{D}_V$, we can exploit the construction above to build a two-way automaton that checks the existence of a canonical DB $\mathcal{B}$ that is consistent with the views and such that $(c,d) \notin ans(Q,\mathcal{B})$. In particular:

1. We construct the one-way automaton A_0 that accepts the language $(\$ \cdot \mathcal{D}_V \cdot \Sigma^+ \cdot \mathcal{D}_V)^* \cdot \$$, hence enforcing the general structure of words representing canonical databases.
2. For each (sound or exact) view V_i and for each pair $(a,b) \in ext(V_i)$, we construct the automaton $A_{(def(V_i),a,b)}$ as specified above.
3. For each exact view V_i we build a two-way automaton A_{V_i} that checks whether a pair of objects other than those in $ext(V_i)$ is in $ans(def(V_i),\mathcal{B})$. More precisely, A_{V_i} is the union of the following automata, which are all obtained by adapting the construction above:
   - an automaton $A_{(V_i,a)}$, for each object a appearing as the first component in a pair in $ext(V_i)$, which starts by placing the head on some randomly chosen occurrence of a, evaluates $def(V_i)$, and after that makes a transition to the final state only if it does not read a symbol b such that $(a,b) \in ext(V_i)$.
   - an additional automaton $A_{(V_i,other)}$, which starts by placing the head on some randomly chosen symbol in the word different from any a that appears as the first component in a pair in $ext(V_i)$, and accepts if it can evaluate the query $def(V_i)$ from there.
4. Finally, we construct the automaton $A_{(Q,c,d)}$ as specified above.

To check whether $(c,d) \notin ans(Q,\mathcal{B})$ for some DB $\mathcal{B}$ consistent with the views, we check for nonemptiness of the one-way automaton A_{ODA} obtained as the intersection of the one-way automata corresponding to each $A_{(def(V_i),a,b)}$ constructed at point 1, of the one-way automata corresponding to the complement of each A_{V_i} constructed at point 2, and of the one-way automaton corresponding to the complement of $A_{(Q,c,d)}$ constructed at point 3.

By exploiting Theorems 13 and 14 it is possible to prove the following correctness result.

THEOREM 15. *Let $V_1, \dots, V_n$ be a set of views, Q an RPQI, and c, d objects in $\mathcal{D}_V$. Then $(c,d) \notin ans(Q,\mathcal{B})$ for some DB $\mathcal{B}$ consistent with $V_1, \dots, V_n$ iff the one-way automaton A_{ODA} constructed as specified above is nonempty.*

Observe that all two-way automata constructed above are of linear size in the size of Q, $def(V_1), \dots, def(V_k)$, and $ext(V_1), \dots, ext(V_k)$. Hence, the corresponding one-way automata would be exponential. However, we do not need to construct A_{ODA} explicitly. Instead, we can construct it "on the fly" while checking for nonemptiness, by exploiting the fact that the transition function of the one-way automaton corresponding to a two-way automaton or its complement can be computed verifying local conditions only [35], and that the same observation holds for intersection.

Considering the lower bounds for view-based query answering under ODA for RPQs in [11], we obtain the following characterization of computational complexity.

THEOREM 16. *View-based query answering under* ODA *for RPQIs is PSPACE-complete wrt expression complexity and combined complexity.*

As for data complexity, it is known that the lower bound is co-NP [11]. To get a matching upper bound, the construction above cannot be directly applied. To isolate the impact of the extensions of the views on complexity, we have to look into the transformation from two-way to complementary one-way automata. In particular, such transformation is based on searching for the existence of a sequence $T_0 \cdots T_n$ of sets of states of the two-way automaton labeling an input word w of length n, such that the sequence satisfies certain conditions on adjacent sets [35].

For simplicity, we consider only the case where the two-way automaton checks whether there is some word $w_{\mathcal{B}}$ representing a canonical DB $\mathcal{B}$, such that $(a, b) \notin ans(Q, \mathcal{B})$. In our case, the number of states of the two-way automaton depends on the objects in the extensions of the views only due to those states used to implement "search mode". Hence, we need to avoid search mode in the two-way automaton. Therefore, we consider the two-way automaton that computes Q over words representing databases as specified in Section 3, and simulate search mode by ensuring that for each occurrence d_i of an object $d \in \mathcal{D}_V$ in the input word, the set of states in which the two-way automaton may be when the head scans d_i is the same. Then, to check whether the two-way automaton accepts some word $w_{\mathcal{B}}$, we guess a labeling of each object d with a set of states of the two-way automaton, and then check for the existence of the sequence of sets of states that completes such labeling. Such check can be done in polynomial time with respect to the number of objects in the extensions of the views.

THEOREM 17. *View-based query answering under* ODA *for RPQIs is co-NP-complete wrt data complexity.*

The summary of our results on the complexity of view-based query answering for regular-path queries with inverse is reported in Table 1. Entries with "all sound" (resp., "all exact") in the column named "Assumption on views" refer to the case where all views are assumed to be sound (resp., exact), whereas "arbitrary" means that for each view V, $as(V)$ can be either SVA, CVA, or EVA. Each entry of the table referring to a complexity class C means that the corresponding problem is complete with respect to C. The table shows that view-based query answering for regular-path queries with inverse has the same computational complexity as for standard regular-path queries.

6. CONCLUSIONS

We have presented a novel technique for view-based query processing in the context of semistructured data, and in particular for regular-path queries extended with the inverse operator. The technique is based on the idea of encoding candidate counterexamples as words, and then checking candidates with two-way automata.

The ability of two-way automata to move back and forth on words is crucial for checking candidate counterexamples, and allows us to show that the inverse operator does not increase the complexity of view-based query rewriting and query answering.

We believe that, besides the specific results, our method has the merit of showing the power of two-way automata in reasoning on complex queries. Indeed, the techniques described in this paper can be adapted to reasoning about queries of more general forms. First results in this direction are reported in [12] for the problem of checking containment of conjunctive regular-path queries with inverse.

7. ACKNOWLEDGMENTS

This work was supported in part by the NSF grant 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".

8. REFERENCES

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

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

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

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

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

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

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

[9] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 9–13. CEUR Electronic Workshop

Proceedings http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/, 1999.

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

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

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

[13] 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.

[14] J. Clark and S. Deach. Extensible Stylesheet Language (XSL). Technical report, World Wide Web Consortium, 1999. Available at http://www.w3.org/TR/WD-xsl.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[29] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978. Republished in [20].

[30] 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.

[31] 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.

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

[33] R. van der Meyden. Logical approaches to incomplete information. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publishers, 1998.

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

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