An Optimization Technique for Answering Regular Path Queries

Gösta Grahne and Alex Thomo
Concordia University
{grahne, thomo}@cs.concordia.ca

ABSTRACT

Rewriting queries using views is a powerful technique that has applications in data integration, data warehousing and query optimization. Query rewriting in relational databases is by now rather well investigated. However, in the framework of semistructured data the problem of rewriting has received much less attention. In this paper we identify some difficulties with currently known methods for using rewritings in semistructured databases. We study the problem in a realistic setting, proposed in information integration systems such as the Information Manifold, in which the data sources are modelled as sound views over a global schema. We give a new rewriting, which we call the *possibility rewriting*, that can be used in pruning the search space for answering queries using views. The possibility rewriting can be computed in time polynomial in the size of the original query and the view definitions. Finally, we show by means of a realistic example that our method can reduce the search space by an order of magnitude.

1. INTRODUCTION

Semistructured data is a self-describing collection, whose structure can naturally model irregularities that cannot be captured by relational or object-oriented data models [2]. Semi-structured data is usually best formalized in terms of labelled graphs, where the graphs represent data found in many useful applications such web information systems, XML data repositories, digital libraries, communication networks, and so on. Semi-structured data is queried through regular path queries, which are queries represented by regular expressions. The design of the regular path queries is based on the observation that many of the recursive queries that arise in practice amount to graph traversals. These queries are in essence graph patterns and the answers to the query are subgraphs of the database that match the given pattern [21, 16, 9, 10]. For example, the regular path query $(\_^* \cdot article) \cdot (\_^* \cdot ref \cdot \_^* \cdot (ullman + widom))$ specifies all the paths having at some point an edge labelled *article*, followed by any number of other edges then by an edge *ref* and finally by an edge labelled with *ullman* or *widom*.

In semistructured data, as well as in data integration, data warehousing and query optimization the problem of query rewriting using views is well known [20, 25, 9, 19]. Simply stated, the problem is: Given a query Q and a set of views $\{v_1, \ldots, v_n\}$, find a representation of Q by means of the views and then answer the query on the basis of this representation. Several papers investigate this problem for the case of conjunctive queries [20, 25, 11, 24]. Their methods are based on the query containment and the fact that the number of literals in the minimal rewriting is bounded from above by the number of literals in the query.

It is obvious that a method for rewriting of regular path queries requires a technique for the rewriting of regular expressions, i.e. given a regular expression E and a set of regular expressions $E_1, E_2, ..., E_n$ one wants to compute a function $f(E_1, E_2, ..., E_n)$ which approximates E. As far as the authors know, there are two methods for computing such a function f which best approximates E from below. The first one of Conway [12] is based on the derivatives of regular expressions introduced by Brzozowski [7], which provide the ground for the development of an algebraic theory of factorization in the regular algebra [8] which in turn gives the tools for computing the approximating function. The second method by Calvanese et al [9] is automata based. Both methods are equivalent in the sense that they compute the same rewriting, which is the largest subset of the query, that can be represented by the views. Posed in the framework of [17], we show that the rewriting produced by these methods is a (sometimes strict) subset of the *certain rewriting*. If we want to be able to produce the complete certain answer, the only alternative left is then to apply an intractable decision procedure of Calvanese et al [10] for *all* pairs of objects (nodes) found in the views. The contribution of this paper is an algorithm for computing a regular rewriting that will produce a superset of the certain answer. The use of this rewriting in query optimization is that it restricts the space of possible pairs needed to be fed to the decision procedure of Calvanese et al. We show by means of a realistic example that our algorithm can reduce the number of candidate pairs by an order of magnitude.

The outline of paper is as follows. In Section 2 we formalize the problem of query rewriting using views in a realistic framework, proposed in information integration systems, in which the data sources are modelled as sound views over a global schema. We give some results about the applicability of previous work in our setting. At the end of Section 2 we sketch an algorithm for utilizing simultaneously several rewritings in query answering using views. In Section 3 we present our main results and method. First we give an algebraic characterization of a rewriting that we call the *possibility rewriting* and then we prove that the answer computed using this rewriting contains the certain answer of the query, even when algebraically the rewriting does not con-

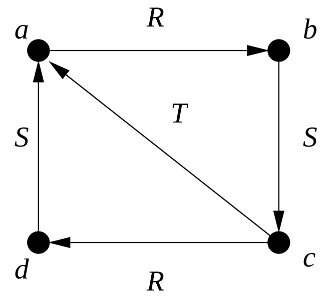

Figure 1: An example of a graph database

tain the query. The computation of the possibility rewriting amounts to finding the transduction of a regular language and we give the appropriate automata-theoretic constructions for these computations.

2. BACKGROUND

Rewriting regular queries. Let Δ be a finite alphabet, called the *database alphabet.* Elements of Δ will be denoted $R, S, T, R', S', \dots, R_1, S_1, \dots$, etc. Let $\mathbf{V} = \{V_1, \dots, V_n\}$ be a set of *view definitions*, with each V_i being a finite or infinite regular language over Δ. We call the set $\Omega = \{v_1, \dots, v_n\}$ the *outer alphabet*, or *view alphabet.* For each $v_i \in \Omega$, we set $def(v_i) = V_i$. The substitution def associates with each "view name" v_i in the view alphabet the language V_i. The substitution def is applied to words, languages, and regular expressions in the usual way (see e. g. [18]).

A *(user) query* Q is a finite or infinite regular language over Δ. A *lower-rewriting* (l-rewriting) of a user query Q using $\mathbf{V}$ is a language Q' over Ω, such that

$$def(Q') \subseteq Q.$$

If for any l-rewriting Q'' of Q using $\mathbf{V}$, it holds that $def(Q'') \subseteq def(Q')$ we say that Q' is *maximal.* If $def(Q') = Q$ we say that the rewriting Q' is *exact.*

Calvanese et al [9] have given a method for constructing an l-rewriting Q' from Q and $\mathbf{V}$. Their method is guaranteed to always find the maximal l-rewriting, and it turns out that the maximal l-rewriting always is regular. An exact rewriting might not exist, while a maximal rewriting always exists, although there is no guarantee on the lower bound. For an extreme example, if $\mathbf{V} = \emptyset$, then the maximal rewriting of any query is $\emptyset$.

Semistructured databases. We consider a database to be an edge labelled graph. This graph model is typical in semistructured data, where the nodes of the database graph represent the objects and the edges represent the attributes of the objects, or relationships between the objects.

Formally, we assume that we have a universe of objects D. Objects will be denoted $a, b, c, a', b', \dots, a_1, b_2, \dots$, and so on. A *database* DB over (D, Δ) is a pair (N, E), where $N \subseteq D$ is a set of nodes and $E \subseteq N \times \Delta \times N$ is a set of directed edges labelled with symbols from Δ. Figure 1 contains an example of a graph database.

If there is a path labelled $R_1, R_2, \dots, R_k$ from a node a to a node b we write $a \xrightarrow{R_1.R_2\dots R_k} b$. Let Q be a query and $DB = (N, E)$ a database. Then the *answer to* Q on DB is defined as

$$ans(Q, DB) = \{(a,b) : \{a,b\} \subseteq N \text{ and } a \xrightarrow{W} b \text{ for some } W \in Q\}.$$

For instance, if DB is the graph in Figure 1, and $Q = \{SR, T\}$, then $ans(Q, DB) = \{(b,d), (d,b), (c,a)\}$

Views and answering queries using views. Let $\Omega = \{v_1, \dots v_n\}$ be the view alphabet and let $\mathbf{V} = \{V_1, \dots, V_n\}$ be a set of view definitions as before. Then a *source collection* $\mathcal{S}$ over $(\mathbf{V}, \Omega)$ is a a database over (D, Ω). A source collection $\mathcal{S}$ defines a set $poss(\mathcal{S})$ of databases over (N, Δ) as follows (cf. [17]):

$$poss(\mathcal{S}) = \{DB : \mathcal{S} \subseteq \bigcup_{i \in \{1,\dots,n\}} \{(a, v_i, b) : (a,b) \in ans(V_i, DB)\}.$$

Suppose now the user gives a query Q in the database alphabet Δ, but we only have a source collection $\mathcal{S}$ available. This situation is the basic scenario in information integration (see e.g. [25, 20, 17]. The best we can do is to approximate Q by

$$\bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB).$$

This approximation is called the *certain answer* for Q using $\mathcal{S}$. Calvanese et al [10], in a follow-up paper to [9] describe an algorithm $\mathcal{A}_{Q,\mathcal{S}}(a,b)$ that returns "yes" or "no" depending on whether given pair (a, b) is in the certain answer for Q or not. This problem is coNP-complete in the number of objects in $\mathcal{S}$ (data complexity), and if we are to compute the certain answer, we need to run the algorithm for *every* pair of objects in the source collection. A brute force implementation of the algorithm runs in time exponential in the number of objects in $\mathcal{S}$. From a practical point of view it is thus important to invoke algorithm $\mathcal{A}_{Q,\mathcal{S}}$ for as few pairs as possible.

Restricting the number of input pairs is not considered by Calvanese et al. Instead they briefly discuss the possibility of using rewritings of regular queries in answering queries using views. Since rewritings have proved to be highly successful in attacking the corresponding problem for relational databases [19], one might hope that the same technique could be used for semistructured databases. Indeed, when the exact rewriting of a query Q using $\mathbf{V}$ exists, Calvanese et al show that, under the "exact view assumption" the rewriting can be used to answer Q using $\mathcal{S}$. Unfortunately, under the more realistic "sound view assumption[1] " adopted in this paper we are only guaranteed to get a subset of the certain answer. The following propositions hold:

[1]If all views are relational projections, the exact view assumption corresponds to the pure universal relation assumption, and the sound view assumption corresponds to the weak instance assumption. For an explanation of the relational assumptions, see [26].

Theorem 1. *Let Q' be an l-rewriting of Q using $\mathbf{V}$. Then for any source collection $\mathcal{S}$ over $\mathbf{V}$,*

$$ans(Q', \mathcal{S}) \subseteq \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB).$$

Proof. Let $(a,b) \in Q'(\mathcal{S})$ and let DB be an arbitrary database in $poss(\mathcal{S})$. Since $(a,b) \in Q'(\mathcal{S})$ there exists objects $c_{i_1} \dots c_{i_k}$ and a path $a\, v_{i_1} c_{i_1} \dots c_{i_k} v_{i_{k+1}} b$ in $\mathcal{S}$ such that $v_{i_1} \dots v_{i_{k+1}} \in Q'$. Since $DB \in poss(\mathcal{S})$, there must be a path $a\, W_{i_1} c_{i_1} \dots c_{i_k} W_{i_{k+1}} b$ in DB, where $W_{i_j} \in def(v_{i_j})$, for $j \in \{1, \dots, k{+}1\}$. Furthermore we have that $W_{i_1} \dots W_{i_k} \in def(Q') \subseteq Q$. In other words, $(a,b) \in ans(Q, DB)$. $\square$

Theorem 2. *There is a query Q and a set of view definitions $\mathbf{V}$, such that there is an exact rewriting Q' of Q using $\mathbf{V}$, but for some source collections $\mathcal{S}$, the set $ans(Q', \mathcal{S})$ is a proper subset of $\bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$.* $\square$

The data-complexity for using the rewriting is NLOGSPACE, which is a considerable improvement from coNP. There is an EXSPACE price to pay though. At the compilation time finding the rewriting requires exponential amount of space measured in the size of the regular expressions used to represent the query and the view definitions (expression complexity). Nevertheless, it usually pays to sacrifice expression complexity for data complexity. The problem is however that the l-rewriting is guaranteed only to produce a subset of the certain answer. We would like to avoid running the testing algorithm $\mathcal{A}_{Q,\mathcal{S}}$ for all other pairs of objects in $\mathcal{S}$.

In the next section we describe a "possibility" rewriting (p-rewriting) Q'' of Q using $\mathbf{V}$, such that for all source collections $\mathcal{S}$:

$$ans(Q'', \mathcal{S}) \supseteq \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB).$$

The p-rewriting Q'' can be used in optimizing the computation of the certain answer as follows:

1. Compute Q' and Q'' from Q using $\mathbf{V}$.
2. Compute $ans(Q', \mathcal{S})$ and $ans(Q'', \mathcal{S})$. Output $ans(Q', \mathcal{S})$
3. Compute $\mathcal{A}_{Q,\mathcal{S}}(a,b)$, for each $(a,b) \in ans(Q'', \mathcal{S}) \setminus ans(Q', \mathcal{S})$. Output those pairs (a,b) for which $\mathcal{A}_{Q,\mathcal{S}}(a,b)$ answers "yes."

3. COMPUTING THE P-REWRITING

As discussed in the previous section, the rewriting Q' of a query Q is only guaranteed to be a contained rewriting. From Propositions 1 and 2 it follows that if we use Q' to evaluate the query, we are only guaranteed to get a subset of the certain answer (recall that the certain answer itself already is an approximation from below). In this section we will give an algorithm for computing a rewriting Q'' that satisfies the relation $ans(Q'', \mathcal{S}) \supseteq \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$. Our rewriting is related to the inverse substitution of regular languages and as consequence it will be a regular language.

Definition 1. *Let L be a language over Ω^*. Then L is a* p-rewriting *of a query Q, using $\mathbf{V}$, if for all $v_{i_1} \dots v_{i_m} \in L$, there exists $W_{i_1} \dots W_{i_m} \in Q$ such that $W_{i_j} \in def(v_{i_j})$, for $j \in \{1, \dots, m\}$, and there are no other words in Ω^* with this property.*

The intuition behind this definition is that we include in the p-rewriting all the words in the view alphabet Ω, such that their substitution by def contains a word in Q. The p-rewriting has the following desirable property:

Theorem 3. *Let Q'' be a p-rewriting of Q using $\mathbf{V}$. Then $ans(Q'', \mathcal{S}) \supseteq \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$, for any source collection $\mathcal{S}$.*

Proof. Assume that there exists a source collection $\mathcal{S}$ and a pair $(a,b) \in \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$, such that $(a,b) \notin ans(Q'', \mathcal{S})$. Since $(a,b) \in \bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$, it follows that for each database $DB \in poss(\mathcal{S})$ there is a path $a \xrightarrow{W} b$, where $W \in Q$. Now, we will construct from $\mathcal{S}$ a database $DB_{\mathcal{S}}$ such that $ans(Q, DB_{\mathcal{S}}) \not\ni (a,b)$. For each edge labelled v_i from one object x to another object y in $\mathcal{S}$ we chose an arbitrary word $W_i \in def(v_i)$ and put in $DB_{\mathcal{S}}$ the "new" objects $c_1, \dots, c_{k-1}$, where k is the length of W_i, and a path $x, c_1, \dots, c_{k-1}, y$ labelled with the word W_i. Each time we introduce "new" objects, so all the constructed paths are disjoint. Obviously, $DB_{\mathcal{S}} \in poss(\mathcal{S})$. It is easy to see that $ans(Q, DB_{\mathcal{S}}) \not\ni (a,b)$ because otherwise there would be a path $v_{i_1} \dots v_{i_m}$ in $\mathcal{S}$ from a to b such that $def(v_{i_1} \dots v_{i_m}) \cap Q \neq \emptyset$, that is $v_{i_1} \dots v_{i_m} \in Q''$ and $(a,b) \in ans(Q'', \mathcal{S})$, From the fact that $ans(Q, DB_{\mathcal{S}}) \not\ni (a,b)$ it then follows that $\bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB) \not\ni (a,b)$; a contradiction. $\square$

It is worth noting here that the Theorem 3 shows that $ans(Q'', \mathcal{S})$ contains the certain answer to the query Q even when algebraically $def(Q'') \not\supseteq Q$.

Recall that the definition of a view name $v_i \in \Omega$ is a regular language $def(V_i)$ over Δ. Thus def is in effect a *substitution* from Ω to 2^{Δ^*}. The *inverse* of this substitution is defined by, for each $W \in \Delta^*$,

$$def^{-1}(W) = \{U \in \Omega^* : W \in def(U)\}.$$

It is now easy to see that a p-rewriting Q'' of of Q using $\mathbf{V}$ equals $def^{-1}(Q)$. This suggests that Q'' can be computed using finite transducers.

A *finite transducer* (see e.g. [27]) $T = (S, I, O, \delta, s, F)$ consists of a finite set of states S, an input alphabet I, and output alphabet O, a starting state s, a set of final states F, and a transition-output function δ from finite subsets of $S \times I^*$ to finite subsets of $S \times O^*$.

An example of a finite transducer $(\{q_0, q_1, q_2\}, \{v_1, v_2\}, \{R, S\}, \delta, \{q_2\})$ is shown in Figure 2.

Intuitively, for instance $(q_1, SRS) \in \delta(q_0, v_2)$ means that if the transducer is in state q_0 and reads word v_2, it can go to state q_1 and emit the word SRS. For a given word $U \in I^*$,

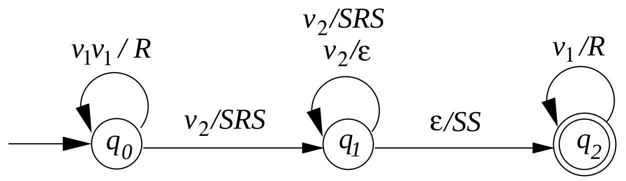

Figure 2: A finite transducer T

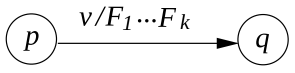

Figure 3:

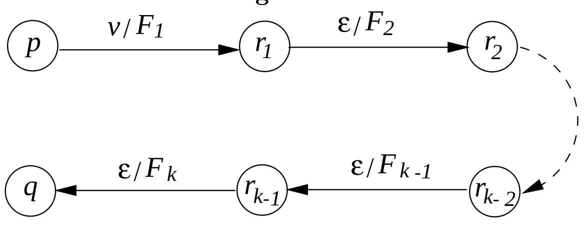

Figure 4:

we say that a word $W \in O^*$ is an *output of T for U* if there exists a sequence $(q_1, W_1) \in \delta(s, U_1)$, $(q_2, W_2) \in \delta(q_1, U_2)$, $\ldots$, $(q_n, W_n) \in \delta(q_{n-1}, U_n)$ of state transitions of T, such that $q_n \in F$, $U = U_1 \ldots U_n \in I^*$, and $W = W_1 \ldots W_n \in O^*$. We write $W \in T(U)$, where $T(U)$ denotes the set of all outputs of T for the input word U. For a language $L \subseteq I^*$, we define $T(L) = \bigcup_{U \in L} T(U)$.

A finite transducer $T = (S, I, O, \delta, s, F)$ is said to be in the *standard form* if δ is a function from $S \times (I \cup \{\epsilon\})$ to $2^{S \times (O \cup \{\epsilon\})}$. Intuitively. the standard form restricts the input and output of each transition to be only a single letter or ϵ. It is known that any finite transducer is equivalent to a finite transducer in standard form (see [27]).

From the above definitions, it is easy to see that a substitution can be characterized by a finite transducer. Start with one node representing both the starting state and the final state. Then build a "macro-transducer" by putting a self-loop corresponding to each $v_i \in \Omega$ on the sole state. In each such self-loop we first have the view symbol v_i as input and a regular expression representing the substitution of v_i as output. After that we transform the "macro-transducer" into an ordinary one in standard form. The transformation is done by applying recursively the following three steps. First, we un-nest all the unions $+$ applying exhaustively the distributive law of concatenation $\cdot$ with respect to the union. In this way a regular expression E is transformed into the form $E_1 + \ldots + E_n$, where each E_i is of the form $F_1 \ldots F_k$ and F_i for $1 \leq i \leq k$ is a single symbol or a regular expression enclosed in a star operation. Then replace the edge $v/(E_1 + \ldots + E_n)$ by the n edges $v/E_1, \ldots, v/E_n$. Second, for each edge of the form $v/F_1 \ldots F_k$ from a node p to a node q (Figure 3), we introduce $k-1$ new nodes $r_1, \ldots\ r_{k-1}$ and replace the edge $v/F_1 \ldots F_k$ by the edges v/F_1 from p to r_1, ϵ/F_2 from r_1 to r_2, $\ldots$, ϵ/F_k from r_{k-1} to q (Figure 4). Third, we get rid of "macro-transitions" of the form v/E^*.

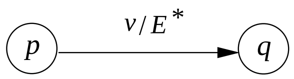

Figure 5:

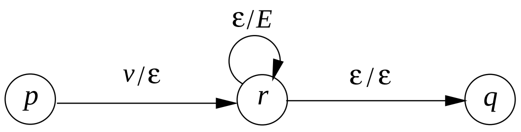

Figure 6:

Suppose we have an edge labelled v/E^* from p to q in the "macro-transducer." (See Figure 5). We introduce a new node r and replace the edge v/E^* by the edges v/ϵ from p to r, ϵ/E from r to r, and ϵ/ϵ from r to q, as shown in Figure 6.

By interchanging the input and output of the finite transducer, we see that the inverse of a substitution can also be characterized by a finite transducer.

We now describe an algorithm that given a regular language L and finite transducer T constructs a finite state automaton that accepts the language $T(L)$. Let $A_L = (S_L, I, \delta_L, s_A, F_L)$ be an ϵ free NFA that accepts L, and let $T = (S_T, I, O, \delta_T, s_T, F_T)$ be the finite transducer in standard form. We construct an NFA $A_{T_L} = (S_{T_L}, O, \delta_{T_L}, s_{T_L}, F_{T_L})$, where $S_{T_L} = S_L \times S_T$, $s_{T_L} = (s_L, s_{T_L})$, $F_{T_L} = F_L \times F_T$, and δ_{T_L} is defined by, for $(p,q) \in S_{T_L}$ and $v \in O \cup \{\epsilon\}$,

$$\begin{aligned} \delta_{T_L}((p,q),v) \quad = \quad & \{(p',q') : \text{ there exists } R \in I \text{ such that} \\ & \delta_L(p,R) = p' \text{ and } (q',v) \in \delta_T(q,R), \\ & \text{or } (q',v) \in \delta_{T_L}(q,\epsilon) \text{ and } p = p'\} \end{aligned}$$

THEOREM 4. *The automaton A_{T_L} accepts exactly the language $T(L)$.* $\Box$

Collecting the results together, we now have the following methodology.

COROLLARY 1. *Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions, such the $def(v_i) = V_i$, for all $v_i \in \Omega$, and let Q be a query over Δ. Then there is an effectively characterizable regular language Q'' over Ω that is the p-rewriting of Q using $\mathbf{V}$.* $\Box$

EXAMPLE 1. *Let the query be $Q = \{(RS)^n : n \geq 0\}$ and the views be v_1, v_2, v_3,and v_4, where $def(v_1) = \{R, SS\}$, $def(v_2) = \{S\}$, $def(v_3) = \{SR\}$, and $def(v_4) = \{RSRS\}$. The DFA A accepting the query Q is given in Figure 7, left, and the transducer characterizing the substitution def is given in Figure 7, right. We transform the transducer into standard form (Figure 8) and then interchange the input with output to get the transducer characterizing the inverse substitution (Figure 9). The constructed automaton A_{T_Q} is shown in Figure 10, where $r_0 = (p_0, q_0)$, $r_1 = (p_1, q_0)$,*

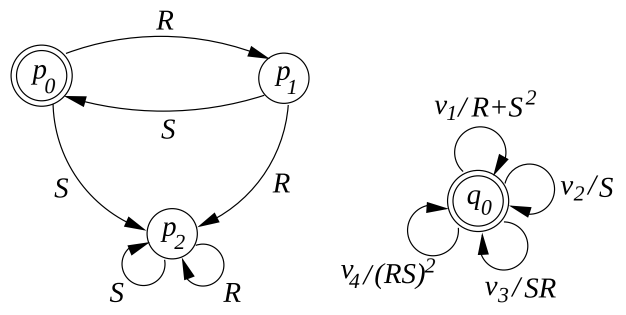

Figure 7:

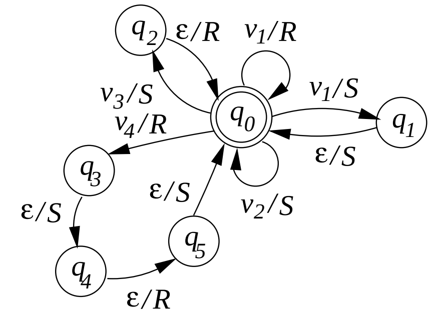

Figure 8:

$r_2 = (p_0, q_2)$ and the inaccessible and garbage states have been removed.

Our algorithm computes the p-rewriting Q'' represented by $(v_4 + v_1 v_3^* v_2)^*$, and the algorithm of Calvanese et al [9] computes the l-rewriting Q' represented by v_4^*. Suppose that the the source collection $\mathcal{S}_n$ is induced by the following set of labelled edges: $\{(i, v_1, a_i) : 1 \leq i \leq n-1\} \cup \{(a_i, v_2, i+1) : 1 \leq i \leq n-1\} \cup \{(a_i, v_3, a_{i+1}) : 1 \leq i \leq n-1\} \cup \{(i, v_4, i+2) : 1 \leq i \leq n-2\}$. We can now compute $ans(Q'', \mathcal{S}_n) = \{(i,j) : 1 \leq i \leq n-1, i \leq j \leq n\}$, and $ans(Q', \mathcal{S}_n) = \{(i, 2k) : 1 \leq i \leq n-1, 0 \leq k \leq n/2\}$. Then we have that the cardinality of $ans(Q'', \mathcal{S}_n)$ is $n + \ldots + 2 = \sum_{i=1}^{n-1}(i+1) = \frac{n(n-1)}{2} - 1 \approx \frac{n^2}{2}$ and the cardinality of $ans(Q', \mathcal{S}_n)$ is $\sum_{i=1}^{n-1}(\lfloor \frac{n-i}{2} \rfloor + 1) \approx \frac{n(n-1)-n}{4} + n \approx \frac{n^2}{4}$. Thus the cardinality of $ans(Q'', \mathcal{S}_n) \setminus ans(Q', \mathcal{S}_n)$ is approximately $n^2/2 - n^2/4 = n^2/4$, that is 16 times better that $(2n)^2$ which the number of all the possible pairs. □

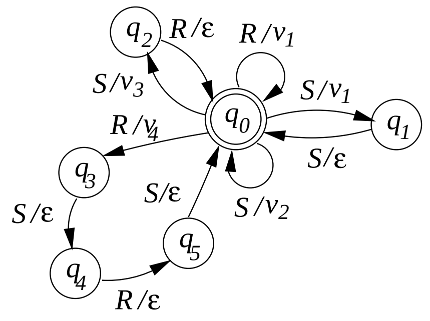

Figure 9:

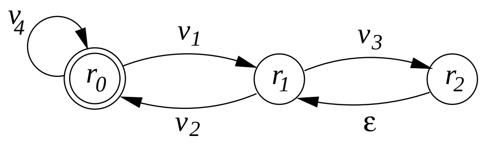

Figure 10:

Now let us calculate the cost of our algorithm for computing the "possibility" regular rewriting.

THEOREM 5. *The automaton A_Q can be built in time polynomial in the size of the regular expression representing Q. The automaton characterizing Q'' can be built in time polynomial in the size of A_Q and the size of the regular expressions representing $\mathbf{V}$.* □

We note that the above analysis is wrt expression and not data complexity. Since the decision procedure of [10] is coNP-complete wrt data complexity, reducing the set of candidate pairs is very desirable.

4. REFERENCES

[1] S. Abiteboul. Querying Semistructured Data. *Proc. of ICDT* 1997 pp. 1–18.

[2] S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and Xml.* Morgan Kaufmann, 1999.

[3] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. L. Wiener. The Lorel Query Language for Semistructured Data. *Int. J. on Digital Libraries* 1997 1(1) pp. 68–88.

[5] P. Buneman. Semistructured Data. *Proc. of PODS* 1997, pp. 117–121.

[6] P. Buneman, S. B. Davidson, M. F. Fernandez and D. Suciu. Adding Structure to Unstructured Data. *Proc. of ICDT* 1997, pp. 336–350.

[7] J. A. Brzozowski. Derivatives of Regular Expressions. *J. ACM 11(4)* 1964, pp. 481–494

[8] J. A. Brzozowski and E. L. Leiss. On Equations for Regular Languages, Finite Automata, and Sequential Networks. *TCS 10* 1980, pp. 19–35

[9] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Proc. of PODS* 1999, pp. 194–204.

[10] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of ICDE* 2000, pp. 389–398

[11] S. Cohen, W. Nutt, A. Serebrenik. Rewriting Aggregate Queries Using Views. *Proc. of PODS* 1999, pp. 155–166

[12] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall 1971.

[13] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, D. Suciu. A Query Language for XML. *WWW8 / Computer Networks 31(11-16)* 1999, pp. 1155–116.

[14] O. Duschka and M. R. Genesereth. Answering Recursive Queries Using Views. *Proc. of PODS* 1997, pp. 109–116.

[15] M. F. Fernandez and D. Suciu. Optimizing Regular path Expressions Using Graph Schemas *Proc. of ICDE* 1998, pp. 14–23.

[16] D. Florescu, A. Y. Levy, D. Suciu Query Containment for Conjunctive Queries with Regular Expressions *Proc. of PODS* 1998, pp. 139–148.

[17] G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. *Proc. of ICDT* 1999 pp. 332–347.

[18] J. E. Hopcroft and J. D. Ullman *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley 1979.

[19] A. Y. Levy. *Answering queries using views: a survey.* Submitted for publication 1999.

[20] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *Proc. of PODS* 1995, pp. 95–104.

[21] A. O. Mendelzon and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp. 24:6,* (December 1995).

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

[23] T. Milo and D. Suciu. Index Structures for Path Expressions. *Proc. of ICDT*, 1999, pp. 277–295.

[24] Y. Papakonstantinou, V. Vassalos. Query Rewriting for Semistructured Data. *Proc. of SIGMOD* 1999, pp. 455–466.

[25] J. D. Ullman. Information Integration Using Logical Views. *Proc. of ICDT* 1997, pp. 19–40.

[26] M. Y. Vardi. The universal-relation model for logical independence. *IEEE Software 5(2)*, 1988, pp. 80–85.

[27] S. Yu. Reqular Languages. In: *Handbook of Formal Languages.* G. Rozenberg and A. Salomaa (Eds.). Springer Verlag 1997, pp. 41–110