INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

QUERY PROCESSING USING VIEWS IN SEMISTRUCTURED DATABASES

Alex-Imir Thomo

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

December 2000

Abstract

Query Processing Using Views in Semistructured Databases

Alex-Imir Thomo

Since its introduction, XML, the eXtensible Markup Language, has quickly emerged as the universal format for publishing and exchanging data in the World Wide Web. As a result, data sources, including object-relational databases, are now faced with a new class of users: clients and customers who would like to deal directly with XML data rather than being forced to deal with the data source particular schema and query languages. XML is also rapidly becoming popular for representing web data as it brings a finely granulated structure to the web information and exposes the semantics of the web content. In all these web applications including electronic commerce and intelligent agents, view mechanisms are recognized as critical and are being widely employed to represent users' specific interests. Rewriting the user queries using views is a powerful technique in the above described applications, which can be categorized as data integration, data warehousing and query optimization. In this study we identify some difficulties with currently known methods for using rewritings in XML-like "semistructured" databases. We study the problem in two realistic scenarios. The first one is related to information integration systems such as the Information Manifold, in which the data sources are modelled as sound views over a global schema. The second scenario, is query optimization using cached views. In this setting we propose two kinds of algebraic rewritings that focus on extracting as much information as possible from the views for the purpose of optimizing regular path queries, which are the building block of all the query languages for semistructured data.

To my mother Lida

Acknowledgments

I would like to thank my supervisor Dr. Gösta Grahne for his guidance, enthusiasm and encouragement. He opened a new world to me. Also, I am very grateful to Professor Jaroslav Opatrny for his exceptional way of teaching me and the other students of the database group the Theory of Computing course. Finally, I wish to thank Dr. Laks Lakshmanan for his fast but deep treatment of the most recent topics in databases that he taught us.

Contents

List of Figures

Chapter 1

Introduction

1.1 Preamble

Until a few years ago the publication of electronic data was limited to a few scientific and technical areas. It is now becoming universal. Most people see such data as Web documents, but these documents, rather than being manually composed, are increasingly generated automatically from databases. It is possible to publish enormous volumes of data in this way, and we are now starting to see the development of software that extracts structured data from Web pages that were generated to be readable by humans. The emergence of XML (Extended Markup Language) as a standard for data representation on the Web is expected greatly to facilitate the publication of electronic data by providing a simple syntax for data that is both human and machine-readable.

Since its introduction, XML, the eXtended Markup Language, has quickly emerged as the universal format for publishing and exchanging data in the World Wide Web. As a result, data sources, including object-relational databases, are now faced with a new class of users: clients and customers who would like to deal directly with XML data rather than being forced to deal with the data source particular schema and query languages. XML is also rapidly becoming popular for representing web data as it brings a finely granulated structure to the web information and exposes the

semantics of the web content.

XML versus HTML. Consider the following example of a common situation in the data exchange in the Web. An organisation publishes data about books, articles and software. The source for this data is a relational database, and the Web pages are generated on demand by invoking an SQL query and formatting its output into HTML. A second organisation wants to obtain some product analyses of this data but has only access to the HTML page(s). Here, the only solution is to write software to parse the HTML and convert it into a structure suitable for the analysis software. This solution has a serious defect: It is brittle, since a minor (text) formatting change in the source could break the parsing program.

In XML, the schema information is stored with the data. Structured values are called elements and attributes, or element names, are called tags. For instance

```
<person><name>Ullman</name><address>Stanford</address></person>
```

is well-formed XML. Thus, XML data is self-describing and can naturally model irregularities that cannot be modeled by relational or object-oriented data. For example, data items may have missing elements or multiple occurrences of the same element; elements may have atomic values in some data items and structured values in others; and collections of elements can have heterogeneous structure.

XML as a graph. We can consider an XML database to be an edge labelled graph. In this graph model we view the nodes of the database graph to represent the objects and the edges to represent the attributes of the objects, or relationships between the objects.

For an example suppose we are given the following XML file.

```
<BOOK bookId="ds">
        <TITLE>Distributed Systems</TITLE>
        <AUTHOR authorId="smith">
                <NAME>Dan Smith</NAME>
                <EMAIL>ds@cs.mcgill.ca</EMAIL>
        </AUTHOR>
        <AUTHOR authorId="bouret">
                <NAME>Emil Bouret</NAME>
                <EMAIL>bouret@alpha.net</EMAIL>
        </AUTHOR>
</BOOK>

<ARTICLE articleId="xml">
        <JOURNAL>ACM J. 2000-12</JOURNAL>
        <TITLE>XML Programming</TITLE>
        <AUTHOR authorId="smith"/>
        <REF refId="ds"/>
</ARTICLE>

<ARTICLE articleId="vb">
        <JOURNAL>PCWorld 2000-12</JOURNAL>
        <TITLE>VBScript Integration</TITLE>
        <AUTHOR authorId="Bouret"/>
        <REF refId="xml"/>
</ARTICLE>

<SOFTWARE>
        <COMPANY>Microsoft</COMPANY>
        <PRODUCT>MsOffice</PRODUCT>
        <SOFTWARE>
                <PRODUCT>MS Access 97</PRODUCT>
        </SOFTWARE>
        <SOFTWARE>
                <PRODUCT>MS PowerPoint 97</PRODUCT>
        </SOFTWARE>
        <SOFTWARE>
                <PRODUCT>Word</PRODUCT>
                <SOFTWARE>
                        <PRODUCT>MS Equations</PRODUCT>
                        <CATEGORY>Mathematics</CATEGORY>
                </SOFTWARE>
        </SOFTWARE>
</SOFTWARE>

<SOFTWARE>
        <COMPANY>Microsoft</COMPANY>
        <PRODUCT>MsPaint</PRODUCT>
        <CATEGORY>Images</CATEGORY>
</SOFTWARE>

<SOFTWARE>
        <COMPANY>Borland</COMPANY>
        <PRODUCT>Borland C++</PRODUCT>
        <CATEGORY>Programming</CATEGORY>
</SOFTWARE>
```

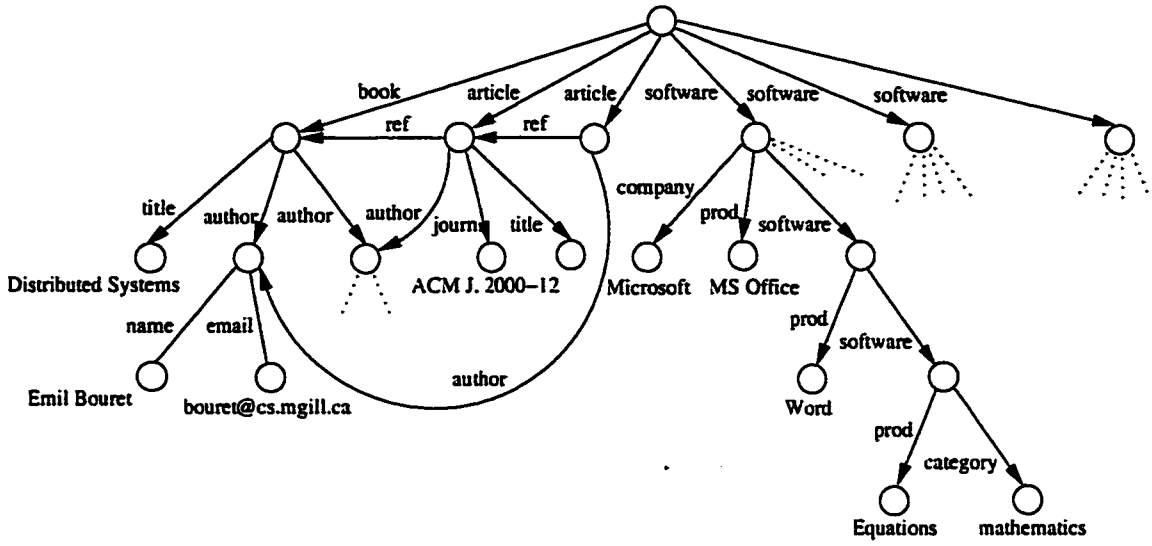


Figure 1: XML as a graph.

Intuitively, this XML database can be represented as the graph of Figure 1. Considering the graph abstraction of XML, instead of XML text, makes it more convenient to study the theoretical database relevant properties and to establish powerful query languages.

1.2 Semistructured Databases and Regular Path Queries

As mentioned before, we abstract the XML data by a data-graph which is the visual representation of the so called *semistructured data model.* Semistructured data is a self-describing collection, whose structure can naturally model irregularities that cannot be captured by relational or object-oriented data models [ABS99]. Semistructured data is usually best formalized in terms of labelled graphs, where the graphs represent data found in many useful applications such as web information systems, XML data repositories, digital libraries, communication networks, and so on.

Formally, let Δ be a finite alphabet, called the *database alphabet.* Elements of Δ will be denoted $R, S, T, R', S', \ldots, R_1, S_1, \ldots$, etc.

Now, assume that we have a universe of objects D. Objects will be denoted $a, b, c, a', b', \ldots, a_1, b_2, \ldots$, 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 2 contains an example of a graph database.

In order to traverse arbitrarily long paths in graph databases, almost all the query languages for semistructured data provide a facility to the user to query through regular path queries, which are queries represented by regular expressions. The design of 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 [MW95, FLS98, CGLV99, CGLV2000]. For example, the regular path query $(\_^* \cdot \mathit{article}) \cdot (\_^* \cdot \mathit{ref} \cdot \_^* \cdot (\mathit{ullman} + \mathit{widom}))$ specifies all the paths having at some point an edge labelled *article*, followed by any number of other edges then by an edge labelled *ref* and finally by an edge labelled with *ullman* or *widom*.

Formally, we consider a *(user) query* Q to be a finite or infinite regular language over Δ. We denote by $re(Q)$ a regular expression describing the regular language Q.

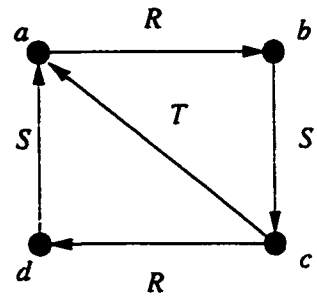


Figure 2: An example of a graph database

If there is a path labelled $R_1, R_2, \ldots, R_k$ from a node a to a node b we write

$$a \xrightarrow{R_1.R_2...R_k} b.$$

Let Q be a query and $DB = (N, E)$ a database. Then the *answer to* Q on DB is defined as

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

Example 1 *For instance, if DB is the graph in Figure 2, and $Q = \{SR, T\}$, then $ans(Q, DB) = \{(b,d), (d,b), (c,a)\}$.*

In semistructured data, as well as in data integration, data warehousing and query optimization, the problem of query rewriting using views is well known [LMSS95, Ull97, CGLV99, Lev99]. 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.

Query rewriting in relational databases is by now rather well investigated. Several papers investigate this problem for the case of conjunctive queries [LMSS95, Ull97, CSS99, PV99]. These methods are based on query containment and the fact that the number of subgoals in the minimal rewriting is bounded from above by the number of subgoals in the query.

However, in the framework of semistructured data the problem of rewriting has received much less attention. In this thesis we identify some difficulties with currently

known methods for using rewritings in semistructured databases and deal with the problem in two realistic scenarios.

The first one is related to information integration systems such as the Information Manifold, in which the data sources are modelled as sound views over a global schema. We give in this setting a new rewriting, which we call the *possibility rewriting*, that can be used in pruning the search space when 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.

The second scenario, is query optimization using cached views. In this setting we propose two kinds of algebraic rewritings that focus on extracting as much information as possible from the views for the purpose of optimizing regular path queries. The cases when we can find a complete exact rewriting of a query using a set a views are very "ideal." However, there is always information available in the views, even if this information is only partial. We introduce "lower" and "possibility" partial rewritings and provide algorithms for computing them. These rewritings are algebraic in their nature, i.e. we use only the algebraic view definitions for computing the rewritings. This fact makes them a main memory product which can be used for reducing secondary memory and remote access. We give two algorithms for utilizing the partial lower and partial possibility rewritings in the context of query optimization.

Chapter 2

Rewriting of Regular Path Queries

2.1 Introduction

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.

Example 2 *Let* $E = (R + S)^*$ *and* $E_1 = RS$, $E_2 = SR$, $E_3 = R$. *Then the best approximation of* E *using* E_1, E_2 *and* E_3 *is*

$$E' = (E_1 + E_2 + E_3)^*.$$

As far as the author knows, there are two methods for computing such a function f which best approximates E from below. The first one of Conway [Con71] is based on the derivatives of regular expressions introduced by Brzozowski [Brzo64], which provide the ground for the development of an algebraic theory of factorization in the regular algebra [BL80] which in turn gives the tools for computing the approximating function. The second method by Calvanese *et. al.* [CGLV99] 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.

In the next section we formalize the problem of query rewriting using views and shortly present the query rewriting proposed by Calvanese *et. al.* [CGLV99], which is called l-rewriting (lower-rewriting) in the sequel. The complexity of computing the l-rewriting is double exponential in the worst case and is discussed at the end of the section.

2.2 L-Rewriting of Regular Path Queries.

Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of *view definitions*, with each V_i being a finite or infinite regular language over Δ. Associated with each view definition V_i is a view name v_i. We call the set $\Omega = \{v_1, \ldots, v_n\}$ 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 Ω alphabet the language V_i. The substitution *def* is applied to words, languages, and regular expressions in the usual way (see e. g. [HU79]).

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.* [CGLV99] 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 is always 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 $\emptyset$.

Their algorithm is:

1. Construct a DFA $A_Q(\Delta, S, s_0, \rho, F)$ such that $Q = L(A_Q)$.

2. Construct automaton $B = (\Omega, S, s_0, \rho', S - F)$, where $s_j \in \rho'(s_i, v)$ iff $\exists W \in V$ such that $s_j \in \rho^*(s_i, W)$.

3. The maximal l-rewriting is the Ω language accepted by complementing the B automaton.

Step 2, says: *Consider each pair of states. If they are connected in A_Q by a* **walk** *labeled with a word in V_i, put an* **edge** *v_i between them in B.*

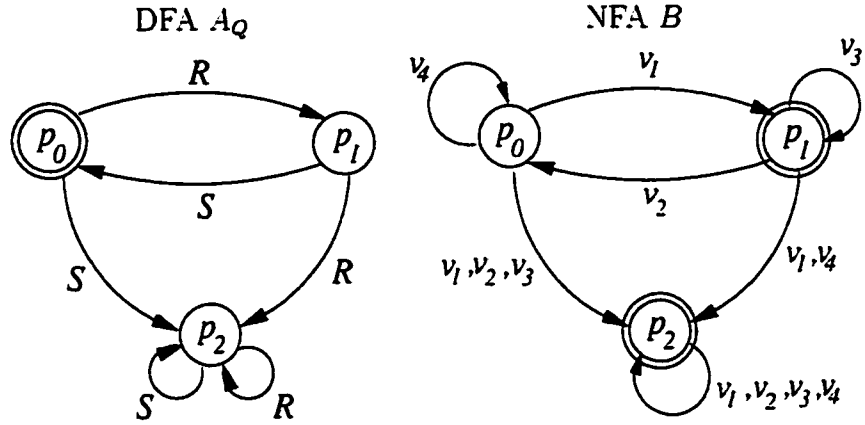
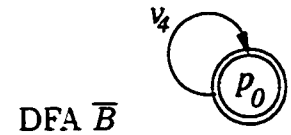


Figure 3: The DFA for the query and the corresponding "view" automaton.

Example 3 *Let $Q = (RS)^*$ be the regular path query and $V_1 = R + S^2$, $V_2 = S$, $V_3 = SR$, $V_4 = (RS)^2$ the regular path views. Then the minimal[1] DFA A_Q for the query Q is given in Figure 3, left and the corresponding B automaton with view symbols is given in in Figure 4, right. The resulting complement automaton $\overline{B}$ is given in Figure 4. Note that the "garbage" and unreachable states have been removed for clearness.*



DFA $\overline{B}$

Figure 4: The resulting l-rewriting given by DFA $\overline{B}$.

[1]The constructed DFA for the query does not need to be minimal.

Observe that, if B accepts an Ω-word $v_1 \cdots v_n$, then there exist n Δ-words $W_1, \ldots, W_n$ such that $W_i \in V_i$ for $i = 1, \ldots, n$ and such that the Δ-word $W_1 \ldots W_n$ is rejected by A_Q. On the other hand if there exists a Δ-word $W_1 \ldots W_n$ that is rejected by A_Q such that $W_i \in V_i$ for $i = 1, \ldots, n$, then the Ω-word $v_1 \cdots v_n$ is accepted by B. That is B accepts an Ω-word $v_1 \cdots v_n$ if and only if there is a Δ-word in $def(\{v_1 \cdots v_n\})$ that is rejected by A_Q. Hence, $\overline{B}$ being the complement of B accepts an Ω-word if and only if all Δ-words $W = W_1 \ldots W_n$ such that $W_i \in V_i$ for $i = 1, \ldots, n$, are accepted by A_Q.

2.3 Complexity of Computing the L-Rewriting

Unfortunately, the complexity of computing the l-rewriting of a regular path query Q given a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions is very high as the following results in [CGLV99] show.

Theorem 1 *The problem of generating the Ω-maximal rewriting of a regular path query with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of regular path view definitions is in EXPTIME.*

In order to use the l-rewriting Q' of a query Q alone for query answering, the rewriting should be algebraically exact (see [CGLV99]), i.e. $def(Q') = Q$. So before talking how the rewritings can be utilized for answering the query (Chapter 3), let us shortly show an optimal algorithm for exactness testing, which is presented in [CGLV99].

Their algorithm for testing if an Ω rewriting of query Q is exact is as follows.

1. Construct an automaton $B = (\Delta, S_B, s_{B0}, \rho_B, F_B)$ that accepts $def(Q')$, by replacing each edge labeled by v_i in the automaton for Q', say $A_{Q'}$, by an automaton A_i such that $L(A_i) = def(v_i)$ for $i = 1, \ldots, n$. Each edge labeled by v_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 Q' is an l-rewriting of Q, $L(B) \subseteq Q = L(A_Q)$.

2. Check whether $L(A_Q) \subseteq L(B)$, that is, check whether $L(A_Q \cap \overline{B}) = \emptyset$.

Theorem 2 *The l-rewriting Q' is an exact rewriting of the query Q with respect to a set $\mathbf{V}$ of regular view definitions, if and only if $L(A_Q \cap \overline{B}) = \emptyset$.*

Corollary 1 *An exact rewriting of Q with respect to $\mathbf{V}$ exists if and only if $L(A_Q \cap \overline{B}) = \emptyset$.*

Theorem 3 *The problem of verifying the existence of an exact rewriting of a regular path query Q with respect to a set $\mathbf{V}$ of regular view definitions, is in 2EXPSPACE.*

Observe that, if we construct $L(A_Q \cap \overline{B}) = \emptyset$., 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 non-emptiness algorithm wants to move from a state s_1 of the intersection of A_Q 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.

In [CGLV99] it is shown that the complexity bounds established in the previous theorems are essentially optimal.

Theorem 4 *The problem of checking whether there is a non-empty rewriting of a regular path query Q with respect to a set $\mathbf{V}$ of regular view definitions, is EXPSPACE-complete.*

Note that Theorem 4 implies that the upper bound established in Theorem 1 is essentially optimal. If we can generate maximal rewritings in, say, EXPTIME, then we could test emptiness in PSPACE, which impossible by Theorem 4.

Chapter 3

Query Processing in Information Integration Systems

3.1 Introduction

Much of the work on answering queries using views has been spurred because of its application to data integration systems. A data integration system provides a uniform query interface to a multitude of autonomous hetereogeneuos data sources. Prime examples of data integration systems include enterprise integration, querying multiple sources in the World Wide Web, and integration of data from distributed scientific experiments. The sources in such an application may be traditional databases, legacy systems, or even structured files. The goal of data integration is to free the user from having to find the data sources relevant to the query, interact with each source in isolation, and manually combine data from the different sources.

To provide a uniform interface, a data integration system exposes to the user a *mediated schema*. A mediated schema is set of virtual relations, in the sense that they are not stored anywhere. The mediated schema is designed manually for a particular data integration application. To be able to answer the queries the system must also contain a set of *source descriptions* that specify the contents of the data sources.

One of the approaches that has been adopted in several systems, is to describe the contents of a source as a *view* over the mediated schema. In order to answer a query, a data integration system needs to translate a query formulated on the mediated schema into one that refers directly to the schemas in data sources. Since the contents of the data sources are described as views, the translation problem amounts to finding a way to answer a query using a set of views.

We illustrate the problem with the following example, where the mediated schema exposed to the user is our bookstore database graph of Figure 1 with the binary relations specified by the edge labels in the graph.

Suppose, we have the following three data sources. The first provides us a listing of each pair (x, y) of objects such that x is an article that refers to the article or book y. This source can be described by the following view definition.

$$\text{ArticleRefArticle: } ref.$$

The second source is supposed to contain all the pairs (x, y) of objects such that x is a book and y is either a book. article or software referred to, directly or indirectly, in the book. This source can be described by the following view definition.

$$\text{BookRefs: } ref^* + ref^*.software.$$

And the third source contains all the pairs (x, y) of objects such that y is a sub-software of x. This source can be described by the following view definition.

$$\text{SoftwareAndSubs: } software^*$$

If we were to ask now, "which objects are somehow related to some other objects", and we have only the contents of the above data sources available then we would be able to answer this query using the following regular expression.

$$\text{Q: } ArticleRefArticle^* + BookRefs.SoftwareAndSubs + SoftwareAndSubs.$$

It is important here to note that this formulation of the query is to be answered against the so called "view graph" which consists of nodes representing the objects

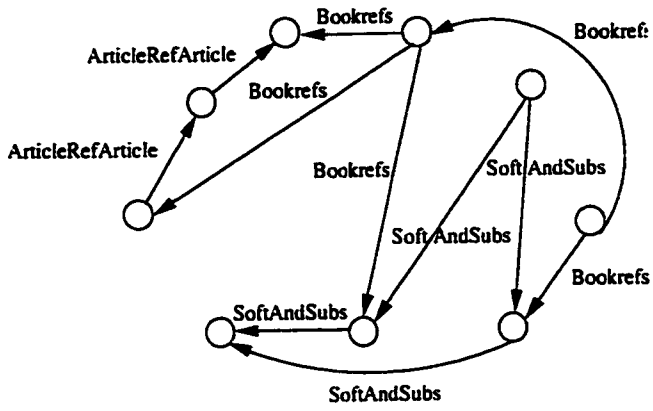


Figure 5: An example of a view graph

found in the data sources and edges labeled with view names. An edge labeled for example with *BookRef* between the objects x and y shows that the pair (x, y) is in the data source described by the second view definition. An example of a view graph is given in Figure 5.

In the previous chapter we showed what is a lower or contained rewriting of a regular path query with respect to a set of regular view definitions, and how to compute a maximal l-rewriting. Posed in the framework of [GM99], we show that the answer to the query that we get if we use the (maximal) l-rewriting only, 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 extremely intractable decision procedure of Calvanese *et. al.* [CGLV2000] for *all* pairs of objects (nodes) found in the views. One of the contributions of this thesis 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 the chapter is as follows. In Section 3.3 we formalize the problem of query rewriting using views in a commonly occurring setting, 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 this setting, and discuss further possibilities of optimization. At the end of Section 3.3 we

give an algorithm for utilizing simultaneously the "subset" and a "superset" rewriting in query answering using views. In Section 3.4 we present our main results. 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 is a superset of the certain answer of the query, even when algebraically the rewriting does not contain the query. Section 3.5 is devoted to the computation of the possibility rewriting. It amounts to finding the transduction of a regular language and we give the appropriate automata-theoretic constructions for these computations.

3.2 Warm-Up

Let the user query be $Q = R_3R_4 + R_3R_5 + R_1R_4 + R_2R_5$ and suppose that we have the following data sources available.

$$\begin{array}{lcllcl} V_1 & = & R_1 & & ext(V_1) & = & \{(a,b)\} \\ V_2 & = & R_2 + R_3 & \text{with} & ext(V_2) & = & \{(a,b)\} \\ V_3 & = & R_4 + R_5 & & ext(V_3) & = & \{(b,c)\} \end{array}$$

These data sources can graphically represented as the view-graph[1] $\mathcal{S}$ in Figure 6. It is easy to see that, the only *contained rewriting* of Q using the views is $Q' = \emptyset$, and

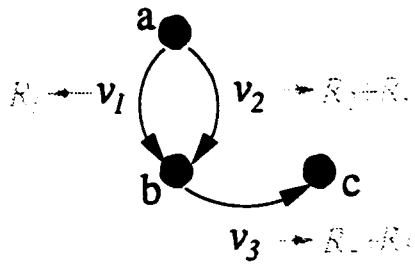


Figure 6: View graph $\mathcal{S}$

therefore $ans(Q', \mathcal{S}) = \emptyset$.

[1]We will use interchangebly the terms "view graph" and "source collection" throughout this thesis.

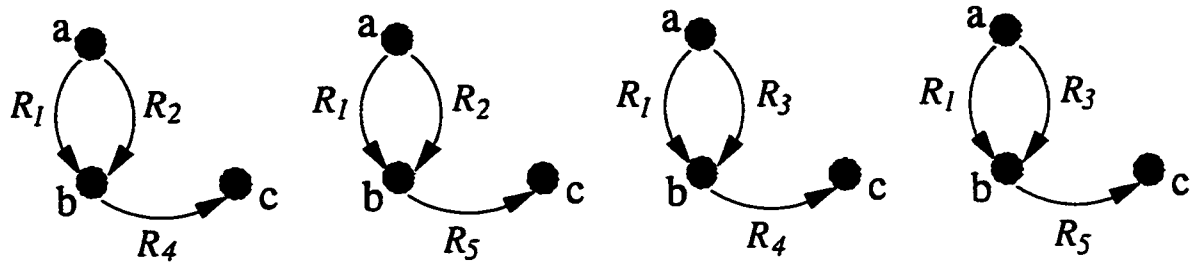


Figure 7: The smallest possible databases

However, what about the rewriting

$$\begin{aligned} Q'' &= V_1V_3 + V_2V_3, \text{ with} \\ ans(Q'', \mathcal{S}) &= \{(a, c)\}. \end{aligned}$$

If we examine all the *possible databases* we can see that the pair (a, c) belongs always to the answer of the query (see Figure 7). We say that the pair (a, c) is a *certain answer* to the query, and observe that it is contained in $ans(Q'', \mathcal{S})$.

3.3 Formal Background

Views and answering queries using views. Let $\Omega = \{v_1, \ldots v_n\}$ be the view alphabet and let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions as before. Then a *source collection* $\mathcal{S}$ over $(\mathbf{V}, \Omega)$ is a database over $(D.\Omega)$. A source collection $\mathcal{S}$ defines a set $poss(\mathcal{S})$ of databases over (D, Δ) as follows (cf. [GM99]):

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

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. [Ull97, LMSS95, GM99]). The best we can do is to approximate Q by

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

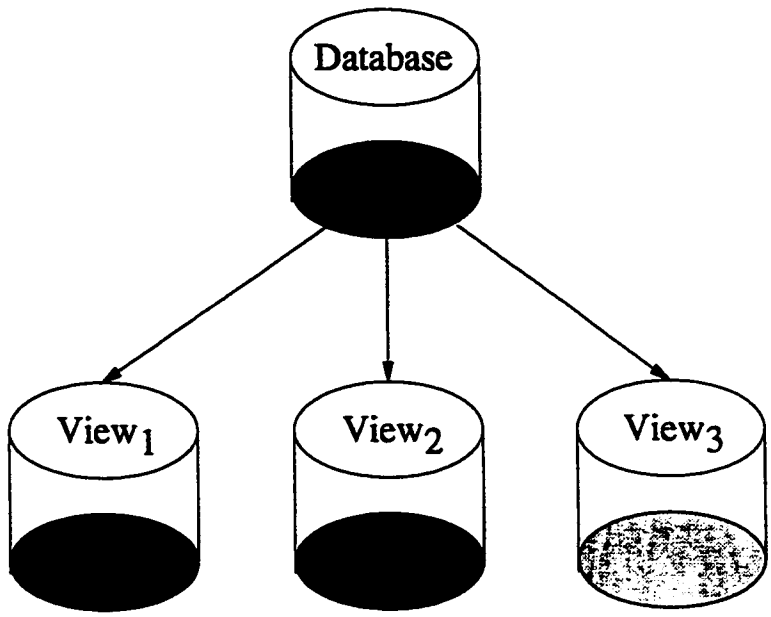


Figure 8: A query and a source collection.

This approximation is called the *certain answer for Q using $\mathcal{S}$*. Calvanese *et. al.* [CGLV2000], in a follow-up paper to [CGLV99] 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 [Lev99], one might hope that the same technique could be used for semistructured databases. Indeed, when the exact rewriting of a query Q using 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[2] " adopted in this

[2]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 [Var88].

chapter we are only guaranteed to get a subset of the certain answer. The following propositions hold:

Theorem 5 *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 exist objects $c_{i_1} \ldots c_{i_k}$ and a path $a\, v_{i_1} c_{i_1} \ldots c_{i_k} v_{i_{k+1}} b$ in $\mathcal{S}$ such that $v_{i_1} \ldots v_{i_{k+1}} \in Q'$. Since $DB \in poss(\mathcal{S})$, there must be a path $a\, W_{i_1} c_{i_1} \ldots c_{i_k} W_{i_{k+1}} b$ in DB, where $W_{i_j} \in def(v_{i_j})$, for $j \in \{1, \ldots, k+1\}$. Furthermore we have that $W_{i_1} \ldots W_{i_k} \in def(Q') \subseteq Q$. In other words, $(a, b) \in ans(Q, DB)$. ■

Theorem 6 *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 $\bigcap_{DB \in poss(\mathcal{S})} ans(Q, DB)$.*

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."

3.4 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 5 and 6 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 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 7 *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

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

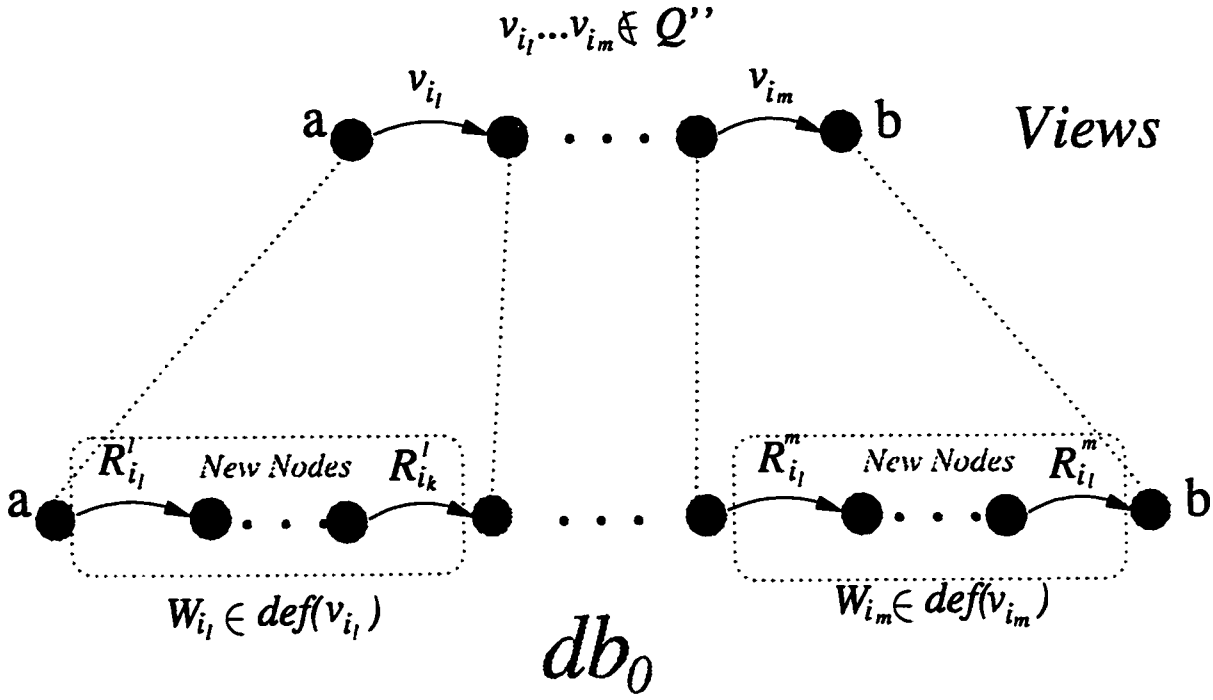


Figure 9: Visualisation of the proof for Theorem 7.

such that $(a, b) \notin ans(Q'', \mathcal{S})$. Since the pair (a, b) is in the certain answer of the query Q, 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 choose an arbitrary word $W_i \in def(v_i)$ and put in $DB_{\mathcal{S}}$ the "new" objects $c_1, \ldots, c_{k-1}$, where k is the length of W_i, and a path $x, c_1, \ldots, c_{k-1}, y$ 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 $v_{i_1} \ldots v_{i_m}$ in $\mathcal{S}$ from a to b such that $def(v_{i_1} \ldots v_{i_m}) \cap Q \neq \emptyset$, that is $v_{i_1} \ldots v_{i_m} \in Q''$ 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. For a visulalisation of the proof see Figure 9. ■

It is worth noting here that the Theorem 7 shows that $ans(Q'', \mathcal{S})$ contains the certain answer to the query Q even when algebraically $def(Q'') \not\supseteq Q$.

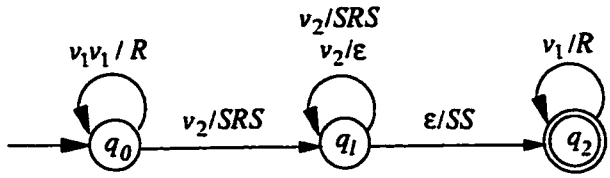


Figure 10: A finite transducer T

3.5 Computing the P-Rewriting

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 Q using $\mathbf{V}$ equals $def^{-1}(Q)$. This suggests that Q'' can be computed using finite transducers.

A *finite transducer* (see e.g. [Yu97]) $T = (S, I, O, \delta, s, F)$ 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 relation $\delta \subseteq S \times I^* \times S \times O^*$. 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 10.

Intuitively, for instance $(q_0, v_2, q_1, SRS) \in \delta$ 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 $U \in I^*$, we say that a word $W \in O^*$ is an *output of T for U* if there exists a sequence $(s, U_1, q_1, W_1) \in \delta$, $(q_1, U_2, q_2, W_2) \in \delta$, ..., $(q_{n-1}, U_n, q_n, W_n) \in \delta$ 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 relation in $S \times (I \cup \{\epsilon\}) \times S \times (O \cup \{\epsilon\})$. 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 [Yu97]).

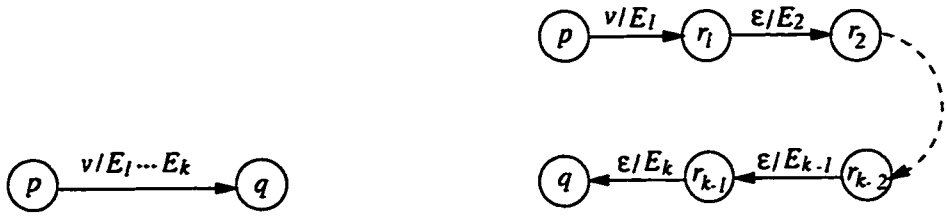


Figure 11: Decomposing a "macro" transducer I.

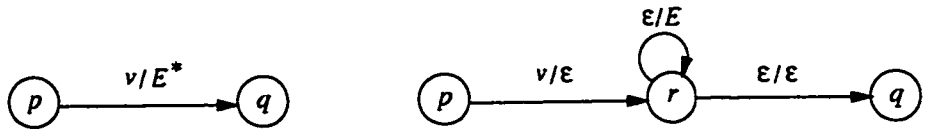


Figure 12: Decomposing a "macro" transducer II.

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, replace each edge $v/(E_1 + \ldots + E_n)$, $n \geq 1$, by the n edges $v/E_1, \ldots, v/E_n$. Second, for each edge of the form $v/E_1 \ldots E_k$ from a node p to a node q (Figure 11, left), we introduce $k-1$ new nodes $r_1, \ldots r_{k-1}$ and replace the edge $v/E_1 \ldots E_k$ by the edges v/E_1 from p to r_1, ϵ/E_2 from r_1 to r_2, ..., ϵ/E_k from r_{k-1} to q (Figure 11, right). Third, we get rid of "macro-transitions" of the form v/E^*. Suppose we have an edge labelled v/E^* from p to q in the "macro-transducer." (See Figure 12, left). 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 12, right.

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 $A = (P, I, \delta_A, s_0, F_A)$ be an ϵ-free NFA that accepts L, and let $T = (S, I, O, \delta_T, p_0, F_T)$

be a transducer in standard form. We construct an NFA:

$$\mathcal{A} = (P \times S, O, \delta, (p_0, q_0), F_A \times F_T),$$

where δ is defined by,

$$\begin{aligned} \delta &= \{((p,q), v, (p',q')) : (p, R, p') \in \delta_A \text{ and } (q, R, q', v) \in \delta_T\} \\ &\cup \{((p,q), v, (p,q')) : (q, \epsilon, q', v) \in \delta_T\} \end{aligned}$$

Theorem 8 *The automaton $\mathcal{A}$ accepts exactly the language $T(L)$.*

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

Corollary 2 *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 $\mathbf{V}$.* ■

Example 4 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[3] A accepting the query Q is given in Figure 13 (left), and the transducer characterizing the substitution *def* is given in Figure 13 (right). We transform the transducer into standard form (Figure 14, left), and then interchange the input with output to get the transducer characterizing the inverse substitution (Figure 14, right). The constructed automaton $\mathcal{A}$ is shown in Figure 15, where $r_0 = (p_0, q_0)$, $r_1 = (p_1, q_0)$, $r_2 = (p_0, q_2)$ 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.* [CGLV99] 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:

$$\{(i, v_1, a_i) : 1 \leq i \leq n-1\} \cup$$

[3] An ϵ-free NFA would do as well.

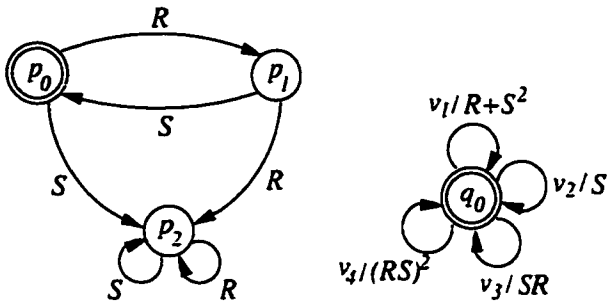


Figure 13: Query automaton and macro transducer.

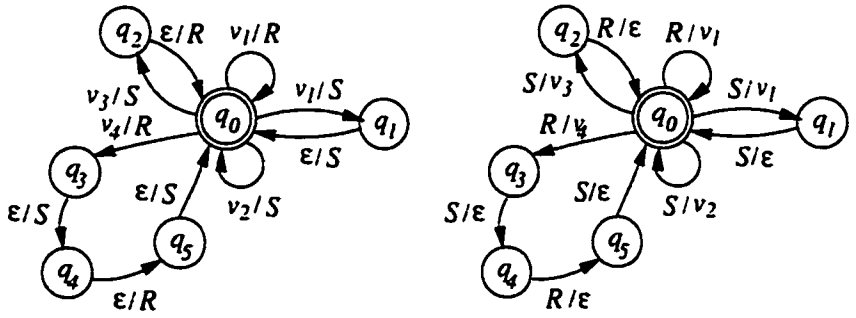


Figure 14: Transducers for the substitution and inverse substitution.

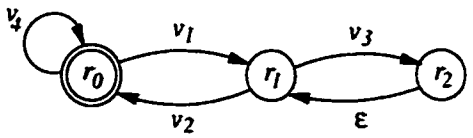


Figure 15: Automaton of the p-rewriting.

$$\{(a_i, v_2, i+1) : 1 \leq i \leq n-1\} \cup$$
$$\{(a_i, v_3, a_{i+1}) : 1 \leq i \leq n-1\} \cup$$
$$\{(i, v_4, i+2) : 1 \leq i \leq n-2\}.$$

The above source collection is visualised in Figure 16.

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

$$\begin{aligned} || ans(Q'', \mathcal{S}_n) || &= n + \ldots + 2 \\ &= \sum_{i=1}^{n-1} (i+1) \\ &= \frac{n(n-1)}{2} - 1 \\ &\approx \frac{n^2}{2} \end{aligned}$$

and the cardinality of

$$\begin{aligned} || ans(Q', \mathcal{S}_n) || &= \sum_{i=1}^{n-1} (\left\lfloor \frac{n-i}{2} \right\rfloor + 1) \\ &\approx \frac{n(n-1) - n}{4} + n \\ &\approx \frac{n^2}{4}. \end{aligned}$$

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. ■

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

Theorem 9 *The automaton characterizing Q'' can be built in time polynomial in the size of the regular expression representing Q and the size of the regular expressions representing* $\mathbf{V}$. ■

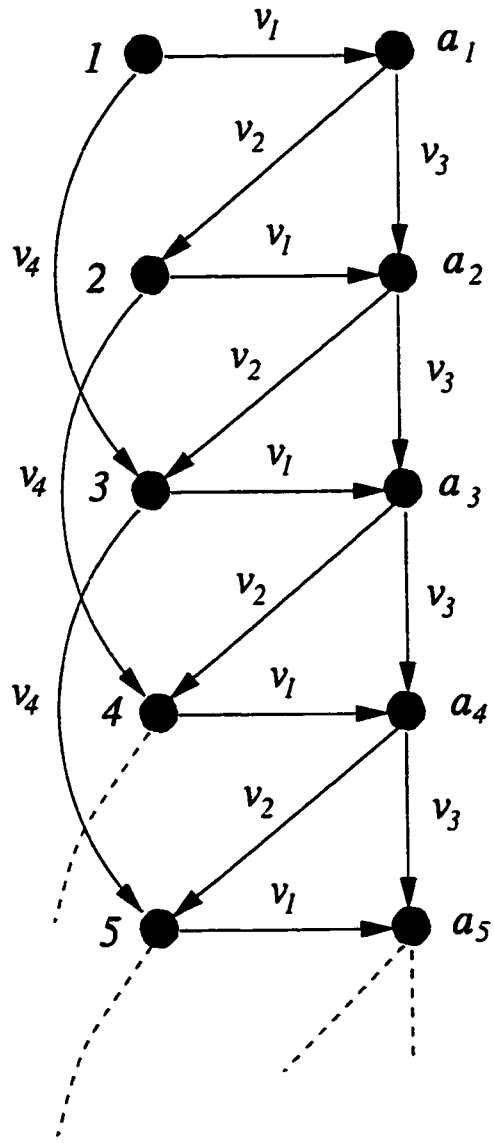


Figure 16: Source collection for Example 4

We note that the above analysis is wrt expression and not data complexity. Since the decision procedure of [CGLV2000] is coNP-complete wrt data complexity, reducing the set of candidate pairs is very desirable.

Chapter 4

Cached View Query Processing Through Partial Rewritings

4.1 Preamble

As mentioned before, rewriting queries using views is a powerful technique that has applications in query optimization, data integration, data warehousing etc. In this chapter we focus on extracting as much information as possible from algebraic rewritings for the purpose of optimizing regular path queries. The cases when we can find a complete exact rewriting of a query using a set of views are very "ideal." However, there is always information available in the views, even if this information is only partial. We introduce "lower" and "possibility" partial rewritings and provide algorithms for computing them. These rewritings are algebraic in their nature, i.e. we use only the algebraic view definitions for computing the rewritings. This fact makes them a main memory product which can be used for reducing secondary memory and remote access. We give two algorithms for utilizing the partial lower and partial possibility rewritings in the context of query optimization.

4.2 Introduction

Almost all the query languages for semi-structured data provide the possibility for the user to query the database through regular expressions. The design of query languages using regular path expressions 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 [MW95, FLS98, CGLV99, CGLV2000].

For example, for answering a query containing in it the regular expression $(\_^* \cdot article) \cdot (\_^* \cdot ref \cdot \_^* \cdot (ullman + widom))$ one should find 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*.

Based on practical observations, the most expensive part of answering queries on semistructured data is finding these graph patterns described by regular expressions. This is, because a regular expression can describe arbitrary long paths in the database which means in turn an arbitrary number of physical accesses. Hence it is clear that, having a good optimizer for answering regular path (sub)queries is very important. This optimizer will take into consideration the already answered queries and try to answer the new query using the information available from the "old" queries or views. It is clear that such an optimizer can be used for the broader class of full fledged query languages for semistructured data.

In Chapter 2 we discussed the l-rewritings and introduced the p-rewritings in Chapter 3. The maximal l-rewriting Q' of a query Q with respect to a set of views, in a cached views scenario, can be used for query answering, only when it is an exact rewriting (see [CGLV2000]). On the other hand the $p - rewriting\ Q''$ can be used always to produce a superset of the answer to the query. We also gave in Chapters 2 and 3 optimal methods to compute the maximal l-rewriting and the p-rewriting respectively, of a query.

However, these methods model –using views– only full paths of the database, i.e. paths whose labels spell a word belonging to the regular language of the query. But

in practice, the cases in which we can infer from the views full paths for the query are very "ideal." The views can cover partial paths which can be satisfactory long for using them in optimization but if they are not complete paths, they are ignored by the above mentioned methods. So, it would probably be better to give a partial rewriting in order to capture all the information provided by the views. The information provided by the views is always useful, even if it is partial and not complete. The problem of a partial rewriting is touched upon briefly in [CGLV99]. However, there this problem is considered only as an extension of the complete rewriting, enriching the set of the views with new elementary one-symbol views, and materializing them before query evaluation. The choice of the new elementary views to be materialized is done in a brute force way, using some cost criteria depending on the application.

In this thesis we use a very different approach. For each word in the regular language of the query we do the best possible using views. If the word contains a sub-path that a view has traversed before, we use that view for evaluation. We present generalized query answering algorithms that access the database only when necessary. For the "been there" subpaths our algorithms use the views. Note that we do not materialize any new views, we only consult the database "on the fly," as needed.

The outline of the chapter is as follows. In Section 4.3 we formalize the problem of query rewriting using views in the realistic framework of cached views and available database. Then we discuss the utility of algebraic rewritings. We illustrate through an example that the complete rewritings can be empty for a particular query, while the partial information provided by the views is no less than 99% of the complete "missing" information. In Section 4.4 we introduce and formally define a new algebraic, formal-language operator, the *exhaustive replacement*. Simply described, given two languages L_1 and L_2, the result of the exhaustive replacement of L_2 in L_1 is the replacement, by a special symbol, of all the words of L_2 that occur as sub-words in the words of L_1. Then we give a theorem showing that the result of the exhaustive replacement can be represented as an intersection of a rational transduction and a regular language. The proof of the theorem is constructive and provides an algorithm for computing the exhaustive replacement operator. In Section 4.5 we present the partial possibility rewriting that is a generalization of the previously introduced

exhaustive replacement operator. In Section 4.6 we define a partial lower rewriting. It is the largest subset of words in the partial possibility rewriting such that their expansions to the the database alphabet are contained in the query language. In Section 4.7 we review a typical query answering algorithm for regular path queries and show how two modify it into two other "lazy" algorithms for utilizing the partial lower and possibility rewritings respectively. The computational complexity is studied in Section 4.8. We show that, although exponential, the algorithms proposed for computing the partial possibility and partial lower rewritings are essentially optimal.

4.3 Background

Rewriting of regular queries revisited. If we look from a word problem perspective to the l-rewriting Q' and p-rewriting Q'' of a query Q with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, we realize the following facts about the rewritings.

A *maximal lower rewriting* (l-rewriting) of a user query Q using $\mathbf{V}$ is a language Q' over Ω, that includes *all* the words $v_{i_1} \ldots v_{i_k} \in \Omega$, such that

$$def(v_{i_1} \ldots v_{i_k}) \subseteq Q.$$

A *maximal possibility rewriting* (p-rewriting) of a user query Q using $\mathbf{V}$ is a language Q'' over Ω, that includes *all* the words $v_{i_1} \ldots v_{i_k} \in \Omega$, such that

$$def(v_{i_1} \ldots v_{i_k}) \cap Q \neq \emptyset.$$

For instance, if $re(Q)$ is $(RS)^*$, and we have the views V_1, V_2, V_3 and V_4 available, with $re(V_1) = R + SS$, $re(V_2) = S$, $re(V_3) = SR$ and $re(V_4) = (RS)^2$ respectively, the l-rewriting is v_4^* and the p-rewriting is $(v_4 + v_1 v_3^* v_2)^*$.

There are cases when the l-rewriting and p-rewriting can be empty, even if the desired answer is not. Suppose for example that query Q is $re(Q) = R_1 \ldots R_{100}$ and we have available two views V_1 and V_2, where $re(V_1) = R_1 \ldots R_{49}$ and $re(V_2) =$

$R_{51} \ldots R_{100}$. It is easy to see that both the l-rewriting and p-rewriting are empty. However, depending on the application, a "partial rewriting" such as $v_1 R_{50} v_2$ could be useful. In the next section we develop a formal algebraic framework for the partial rewritings. This framework is flexible enough and can be easily tailored to the specific needs of the various applications. In Section 4.7 we demonstrate the usability of the partial rewritings in query optimization.

4.4 Replacement – A New Algebraic Operator

In this section we introduce and study a new algebraic operation, the exhaustive replacement in words and languages. It is similar in spirit to the deletion and insertion language operations studied in [Kari91].

Let W be a word, and M a ϵ-free language over some alphabet, and let $\dagger$ be a symbol outside that alphabet. Then we define

$$\rho_M(W) = \begin{cases} \{W_1 \dagger W_3 : \exists\ W_2 \in M \text{ such that } W = W_1 W_2 W_3\} & \text{if non-empty} \\ \{W\} & \text{otherwise.} \end{cases}$$

Furthermore, let L be a set of words over the same alphabet as M. Then define $\rho_M(L) = \bigcup_{W \in L} \rho_M(W)$. We can now define the *powers of* ρ_M as follows:

$$\rho_M^1(\{W\}) = \rho_M(W), \quad \rho_M^{i+1}(\{W\}) = \rho_M(\rho_M^i(\{W\})).$$

Let k be the smallest integer such that $\rho_M^{k+1}(\{W\}) = \rho_M^k(\{W\})$. We then set

$$\rho_M^*(W) = \rho_M^k(\{W\}).$$

(It is clear that k is at most the number of symbols in W.)

The *exhaustive replacement* of a ϵ-free language M in a language L, using a special symbol $\dagger$ not in the alphabet, can be simply defined as

$$L \rhd M = \bigcup_{W \in L} \rho_M^*(W).$$

Intuitively, the exhaustive replacement $L \rhd M$ replaces in every word $W \in L$ the non-overlapping occurrences of words from M with the special symbol $\dagger$. Moreover,

between two occurrences of words of M that have been replaced, no word from M remains as a subword.

Example 5 Let $L = \{RSRSRSR, RRSRSR, RSRRSRRSR\}$, $M = \{RSR\}$. Then

$$L \rhd M = \{\dagger S\dagger, RS\dagger SR, R\dagger SR, RRS\dagger, \dagger\dagger\dagger\},$$

being the union of the sets:

$$\begin{aligned} \rho^*_{\{RSR\}}(RSRSRSR) &= \{\dagger S\dagger, RS\dagger SR\}, \\ \rho^*_{\{RSR\}}(RRSRSR) &= \{R\dagger SR, RRS\dagger\}, \\ \rho^*_{\{RSR\}}(RSRRSRRSR) &= \{\dagger\dagger\dagger\}. \end{aligned}$$

Computing the Replacement Operation. To this end, we will give first a characterization of the $\rhd$ operator. The construction in the proof of our characterization provides the basic algorithm for computing the result of the $\rhd$ operator on given languages. The construction is based on finite transducers.

Theorem 10 *Let L and M be regular languages over an alphabet Δ. There exists a finite transducer T and a regular language M' such that:*

$$L \rhd M = T(L) \cap M'.$$

Proof. Let $A = (S, \Delta, \delta, s_0, F)$ be a nondeterministic finite automaton that accepts the language M. Let us consider the finite transducer:

$$T = (S \cup \{s'_0\}, \Delta, \Gamma, \delta', s'_0, \{s'_0\}),$$

where $\Gamma = \Delta \cup \{\dagger\}$, and, written as a relation,

$$\delta' = \{(s, R, s', \epsilon) : (s, R, s') \in \delta\} \cup$$

$$\{(s_0', R, s_0', R) : R \in \Delta\} \cup$$
$$\{(s_0', R, s, \epsilon) : (s_0, R, s) \in \delta\} \cup$$
$$\{(s_0', R, s_0', \dagger) : (s_0, R, s) \in \delta \text{ and } s \in F\} \cup$$
$$\{(s, R, s_0', \dagger) : (s, R, s') \in \delta \text{ and } s' \in F\}.$$

Intuitively, transitions in the first set of δ' are the transitions of the "old" automaton modified so as to produce ϵ as output. Transitions in the second set mean that "if we like, we can leave everything unchanged," i.e. each symbol gives itself as output. Transitions in the third set are for jumping non-deterministically from the new initial state s_0' to the states of the old automaton A, that are reachable in one step from the old initial state s_0. These transitions give ϵ as output. Transitions in the fourth set are for handling special cases, when from the old initial state q_0, an old final state can be reached in one step. In these cases we can replace the one symbol words accepted by A with the special symbol $\dagger$. Finally, the transitions of the fifth set are the most significant. Their meaning is: in a state, where the old automaton has a transition by a symbol, say R, to an old final state, in the transducer there will be an additional transition $R/\dagger$ to s_0', which is also the (only) final state of T. Observe that if the transducer T decides to leave the state s_0' while a suffix U of the input string is unscanned, and enter the old automaton A, then it can return back only if there is a prefix U' of U, such $U' \in L(A)$. In this case the transducer replaces U', which is a subword of the input string, by the special symbol $\dagger$.

Given a word of $W \in L$ as input, the finite transducer T replaces arbitrary many occurences of words of M in W with the special symbol symbol $\dagger$.

For an example, suppose M is $R(SR)^* + RST$. Then an automaton that accepts this language is given in Figure 17 drawn with solid arrows. The corresponding finite transducer is shown in the same figure on the right. It consists of the automaton A, whose transitions now produce as output ϵ, plus the state s_0' and the additional transitions drawn with dashed arrows.

It is easy to see that

$$T(L) = L \cup \{U_1 \dagger U_2 \dagger \ldots \dagger U_k : \text{ for some } U \text{ in } L \text{ and words } W_i \text{ in } M, \\ U = U_1 W_1 U_2 W_2 \ldots W_{k-1} U_k\}.$$

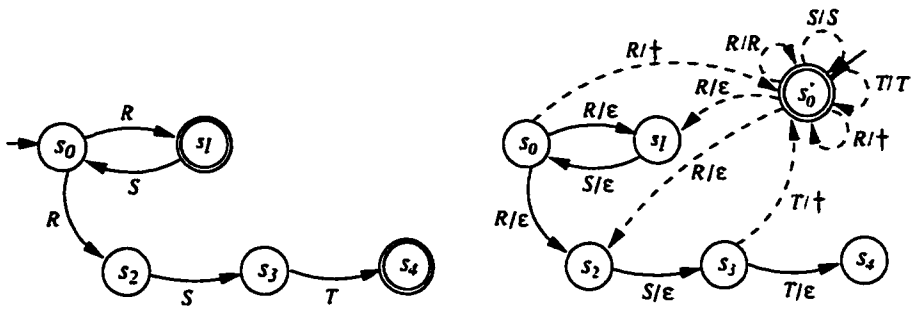


Figure 17: An example of the construction of a replacement transducer

From the transduction $T(L)$ we get all the words of L having replaced in them an arbitrary number of words from M. What we need is not an arbitrary but an exhaustive replacement of words from M. To achive this goal we will intersect the language $T(L)$ with a regular language M' which will serve as a "mask" for the words of $L \triangleright M$. We set

$$M' = (\Gamma^* M \Gamma^*)^c .$$

Now M' guarantees that no other candidate for replacing occurs inside the words of the final result. ■

4.5 Partial P-Rewritings

We can give a natural generalization of the definition of the replacement operator for the case when we like to exhaustively replace subwords not from one language only, but from a finite set of languages (such as a finite set of view definitions). For this purpose, let W be a word and $\mathbf{M} = \{M_1, \ldots, M_n\}$ be a set of languages over some aplhabet, and let $\{\dagger_1, \ldots, \dagger_n\}$ be a set of symbols outside that alphabet. Now we define

$$\rho_{\mathbf{M}}(W) = \begin{cases} \{W_1 \dagger_i W_3 : \exists\ W_2 \in M_i \text{ such that } W = W_1 W_2 W_3\} & \text{if non-empty} \\ \{W\} & \text{otherwise.} \end{cases}$$

Then, $\rho^*_{\mathbf{M}}$ is defined similarly to ρ^*_M.

The *generalized exhaustive replacement* of $\mathbf{M} = \{M_1, \ldots, M_n\}$ in a language L,

by the corresponding special symbols $\dagger_1, \ldots, \dagger_n$, is

$$L \rhd \mathbf{M} = \bigcup_{W \in L} \rho^*_{\mathbf{M}}(W).$$

In the following we will define the notion of the *partial p-rewriting* of a database query Q using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Definition 2 The *partial p-rewriting* of a query Q over Δ using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions over Δ is

$$Q \rhd \mathbf{V},$$

with $\Omega = \{v_1, \ldots, v_n\}$ as the corresponding set of special symbols.

As a generalization of Theorem 10 we can give the following result about the partial p-rewriting of a query Q over Δ using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions over Δ.

Theorem 11 *The partial p-rewriting $Q \rhd \mathbf{V}$ can be effectively computed.*

Proof. Let $A_i = (S_i, \Delta, \delta_i, s_{0i}, F_i)$, for $i \in [1, n]$ be n nondeterministic finite automata that accept the corresponding V_i languages. Let us consider the finite transducer:

$$T = (S_1 \cup \ldots \cup S_n \cup \{s'_0\}, \Delta, \Delta \cup \Omega, \delta', s'_0, \{s'_0\}),$$

where

$$\begin{aligned}
\delta' \;=\; & \{(s, R, s', \epsilon) : (s, R, s') \in \delta_i,\ i \in [1, n]\} \cup \\
& \{(s'_0, R, s'_0, R) : R \in \Delta\} \cup \\
& \{(s'_0, R, s, \epsilon) : (s_{0i}, R, s) \in \delta_i,\ i \in [1, n]\} \cup \\
& \{(s'_0, R, s'_0, \dagger_i) : (s_{0i}, R, s) \in \delta_i \text{ and } s \in F_i,\ i \in [1, n]\} \cup \\
& \{(s, R, s'_0, \dagger_i) : (s, R, s') \in \delta_i \text{ and } s' \in F_i,\ i \in [1, n]\}.
\end{aligned}$$

The transducer T performs the following task: given a word of Q as input, it replaces nondeterministically some words of $V_1 \cup \ldots \cup V_n$ from the input with the

corresponding special symbols. The proof of this claim is similar as in the previous theorem.

From the transduction $T(Q)$ we get all the words of Q having replaced in them an arbitrary number of words from $V_1 \cup \ldots \cup V_n$. But what we want is the exhaustive replacement $Q \triangleright \mathbf{V}$. For this we intersect the language $T(Q)$ with the regular language

$$((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*)^c,$$

which will serve as a mask for extracting the words in the exhaustive replacement. ■

We note here that the partial p-rewriting of a query is a generalization of the p-rewriting. Indeed, consider the the substitution from $\Omega \cup \Delta$ that maps each $v_i \in \Omega$ to the corresponding regular view language V_i and each database symbol $R \in \Delta$ to itself. This substitution is the extension of the *def* substitution to the Δ alphabet and we call it *def'*. Then the partial p-rewriting is the set of *all* the words W on $\Omega \cup \Delta$, with no subwords in any of $V_1, \ldots, V_n$, such that $def'(W)$ has a non empty intersection with Q. The conceptual similarity of the partial p-rewriting with p-rewriting can also be observed in another way; change the above mask to Ω^* and the result will be the p-rewriting, as opposed to the partial p-rewriting.

4.6 Partial L-Rewritings

We defined the l-rewriting of a query Q given a set of view definitions $\mathbf{V} = \{V_1, \ldots, V_n\}$ as the set of all the words W on the view alphabet Ω such that $def(W)$ is contained in the query language Q. In the same spirit we will define the partial l-rewriting. It will be the set of all "mixed" words W on the alphabet $\Omega \cup \Delta$, with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by the extended *def'* is contained in the query Q. The condition that there is no subword in $V_1 \cup \cdots \cup V_n$, says that in fact the partial l-rewriting is a subset of the partial p-rewriting.

Definition 3 The *partial l-rewriting* of a query Q on Δ is the language Q' on $\Omega \cup \Delta$

given by

$$Q' = \{W \in (Q \rhd \mathbf{V}) : def'(W) \subseteq Q\}.$$

We now give a method for computing the partial l-rewriting, given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions as input.

Algorithm 1 *1. Compute the complement Q^c of the query.*

2. *Construct the transducer T used for the partial p-rewriting. Then compute the transduction $T(Q^c)$.*

3. *Compute the complement $(T(Q^c))^c$ of the previous transduction.*

4. *Intersect the complement $(T(Q^c))^c$ with the mask*

$$M = ((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*)^c$$

Denote with Q' the result. ■

Theorem 12 *The mixed $\Omega \cup \Delta$ language Q' gives exactly the partial l-rewriting of Q.*

Proof. "$\subseteq$". $T(Q^c)$ is the set of all words W on $\Omega \cup \Delta$ such that $def'(W) \cap Q^c \neq \emptyset$. Hence, $(T(Q^c))^c$, being the complement of this set, will contain only $\Omega \cup \Delta$ words such that *all* the Δ-words in their substitution by *def'* will be contained in Q. This is the first condition for a word on $\Omega \cup \Delta$ to be in the partial l-rewriting of Q. Furthermore, intersecting with the mask M we keep in $(T(Q^c))^c$ only the $\Omega \cup \Delta$ words that do not contain Δ subwords in $V_1 \cup \cdots \cup V_n$. This is the second condition for a word on $\Omega \cup \Delta$ to be in the partial l-rewriting of Q.

"$\supseteq$". We will prove this direction by a contradiction. First observe that both the partial l-rewriting and the set Q' are subsets of the partial p-rewriting, that is, all their words "pass" the mask M. In other words their words do not have subwords in

$V_1 \cup \cdots \cup V_n$. Suppose now, that the mixed $\Omega \cup \Delta$-word W is in the partial l-rewriting but not in Q'. That is $def'(W) \subseteq Q$. On the other hand, since $W \notin Q'$ it follows that $W \in Q'^c$ which means that $W \in T(Q^c) \cup M^c$. But as we mentioned before, the word W, which belongs in the partial l-rewriting, "passes" the mask M and this implies that it cannot "pass" the complement of the mask. Therefore, $W \in T(Q^c)$. Thus $def'(W) \cap Q^c \neq \emptyset$ that is, $def'(W) \not\subseteq Q$, i.e. W cannot be in the partial l-rewriting, a contradiction. ■

4.7 Query Optimization Using Partial Rewritings and Views

In this section we show how to utilize partial rewritings in query optimization in a scenario where we have available a set of precomputed views, as well as the database itself. The views could be materialized views in a warehouse, or locally cached results from previous queries in a client/server environment. In this scenario the views are assumed to be excact, and we are interested in answering the query by consulting the views as far as possible, and by accessing the database only when necessary.

Formally, let $\Omega = \{v_1, \ldots v_n\}$ be the view alphabet and let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions as before. Given a database DB, which is a graph, where the edges are labelled with database symbols from Δ, we define the *view graph* $\mathcal{V}$ over $(\mathbf{V}, \Omega)$ to be a database over (D, Ω) induced by the set

$$\bigcup_{i \in \{1,\ldots,n\}} \{(a, v_i, b) : (a, b) \in ans(V_i, DB)\}.$$

of Ω-labelled edges.

It is now straightforward to show, that if the l-rewriting Q' is exact (meaning $def(Q') = Q$), then $ans(Q, DB) = ans(Q', \mathcal{V})$ (see Calvanese *et. al.* [CGLV2000]).

However, the cases when we are able to obtain an exact rewriting of the query using the views would be rare in practice, in general we have in the views only part of the information needed to answer the query. So, should we ignore this partial

information only beacuse it is not complete? In the previous sections we showed how this partial information can be captured algebraically by the partial rewritings. In the following, we use the partial rewritings not to avoid *completely* accessing the database, but to *minimize* such access as much as possible.

However, in order to be able to utilize the partial l-rewriting Q', it should be exact, i.e. we require that $def'(Q') = Q$. We can use for testing the exactness the optimal algorithm of [CGLV99].

Given an exact partial l-rewriting, we can use it to evaluate the query on the view-graph, and accessing the database in a "lazy" fashion, only when necessary. Before describing the lazy algorithm, let us review how query answering on semistructured databases typically works [ABS99].

Algorithm 2 We are given a regular expression for Q and a database graph DB. First construct an automaton A_Q for Q. Let N be the set of nodes in the database graph, and s_0 be the initial state in A_Q. For each node $a \in N$ compute a set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. Repeat 3 until $Reach_a$ no longer changes.
3. Choose a pair $(x, s) \in Reach_a$. If there is a database symbol R, such that a transition $s \xrightarrow{R} s'$ is in A_Q and an edge $x \xrightarrow{R} x'$ is in the database DB, then add the pair (x', s') to $Reach_a$.

Finally, set

$$ans(Q, DB) = \{(a, b) : a \in N,\ (b, s) \in Reach_a,\ \text{and } s \text{ is a final state in } A_Q\}.$$

■

In the following we modify this algorithm into a lazy algorithm for answering a query Q using its partial l-rewriting with respect to a set of cached exact views.

Algorithm 3 We are given an automaton $A_{Q'}$, corresponding to an exact partial l-rewriting Q' and the view graph $\mathcal{V}$. Let N be the set nodes in $\mathcal{V}$, and s_0 be the initial state in $A_{Q'}$. For each node $a \in N$ then compute a set $Reach_a$.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$, and $Expanded_a$ to **false**.
2. For each database symbol R, if there is in $A_{Q'}$ a transition $s_0 \xrightarrow{R} s$ from the initial state s_0, then access the database and add to $\mathcal{V}$ the subgraph of DB induced by the R-edges.
3. Repeat 4 until $Reach_a$ no longer changes.
4. Choose a pair $(x, s) \in Reach_a$. If there is a view or database symbol R, such that a transition $s \xrightarrow{R} s'$ is in $A_{Q'}$, go to 5.
5. If there is an edge $a \xrightarrow{R} a'$ in the viewgraph, add the pair (x', a') to $Reach_a$. Otherwise, if $Expanded_a =$ **false**, set $Expanded_a =$ **true**, access the database and add to $\mathcal{V}$ the subgraph of DB induced by all edges originating from a.

Set $eval(Q', \mathcal{V}, DB) =$

$$\{(a, b) : a \in N, (b, s) \in Reach_a, \text{ and } s \text{ is a final state in } A_{Q'}\}.$$

■

It is easy to verify the following.

Theorem 13 *Given a query Q and a set of exact views, if the partial l-rewriting Q' of Q is exact, then $eval(Q', \mathcal{V}, DB) = ans(Q, DB)$.*

Next, let us discuss how to utilize the partial p-rewriting Q'' of a query Q for computing the answer set $ans(Q, DB)$. If we use the same algorithm as in the case of the partial l-rewriting we might get a proper superset of the answer. Note however that, contrary to Algorithm 3, in any case the partial p-rewriting does not need to be exact.

Theorem 14 *Given a query Q and a set $\mathcal{V}$ of exact views, if Q'' is the partial p-rewriting of Q using $\mathcal{V}$, then $ans(Q, DB) \subseteq eval(Q'', \mathcal{V}, DB)$.*

In other words, we are not sure if all the pairs are valid. To be able to discard false hits, suppose that the views are materialized using Algorithm 2. We can then associate each pair (a, b) in the view graph with their derivation. That is, for each pair (a, b) connected with an edge, say v_i, in the view graph, we associate an automaton, say A_{ab}, with start state a and final states $\{b\}$. What is this automaton? For each pair (a, b), we can consider the database graph as a non-deterministic automaton DB_{ab} with initial state a and final states $\{b\}$. It is now easy to see that

$$A_{ab} = DB_{ab} \cap A_{V_i}$$

where A_{V_i} is an automaton for the view V_i. We are now ready to formulate the algorithm for using the partial p-rewriting in query answering.

Algorithm 4

1. Compute $eval(Q'', \mathcal{V}, DB)$ using Algorithm 3. During the execution of Algorithm 3 the view graph $\mathcal{V}$ is extended with new edges and nodes as described. Call the extended view graph $\mathcal{V}'$.
2. Replace in $\mathcal{V}$' each edge labeled with a view symbol, say v_i, between two objects a and b with the automaton A_{ab} of the derivation. Call the new graph $\mathcal{V}''$.
3. Set $verified(Q'', \mathcal{V}, DB) = eval(Q'', \mathcal{V}, DB) \cap \{(a, b) : ans(Q, \mathcal{V}''_{ab}) \neq \emptyset\}$, where $\mathcal{V}''_{ab}$ is a non-deterministic automata similar to D_{ab}.

It is easy verify the following.

Theorem 15 *Given a query Q and a set $\mathcal{V}$ of exact views, if Q'' is the partial p-rewriting of Q using $\mathcal{V}$, then $verified(Q'', \mathcal{V}, DB) = ans(Q, DB)$.*

4.8 Complexity Analysis

The following theorem establishes an upper bound for the problem of generating the exhaustive replacement $L \rhd M$, where L and M are regular languages.

Theorem 16 *Generating the exhaustive replacement of a regular language M from another language L can be done in exponential time.*

Proof. Let us refer to the cost of the steps in the constructive proof of the Theorem 10. To construct a non-deterministic automaton for the language M and using it to construct the transducer g is polynomial. To compute the transduction of the regular language L, $g(L)$, is again polynomial. But at the end, in order to compute the subset of the words in $g(L)$, to which no more replacement can be applied, is exponential. This is because we intersect with a mask that is a language described by an extended regular language containing complementation. ■

Theorem 17 *Let Γ be an alphabet and A, B be regular languages over Γ. Then the problem of deciding the emptiness of $A \cap (\Gamma^* B \Gamma^*)^c$ is PSPACE complete.*

Proof. First, observe that

$$[A \cap (\Gamma^* B \Gamma^*)^c = \emptyset] \Leftrightarrow [A \subseteq \Gamma^* B \Gamma^*]$$

But, this problem is a sub-case of the problem of testing regular expression containment, which is known to be PSPACE complete [HRS76]. So, there exists an algorithm running in polynomial space that decides the above problem.

Next, we show that the problem is PSPACE-hard. Let $\mathcal{L}$ be a language that is decided by a Turing machine M running in polynomial space n^k for some constant k. The reduction maps an input w into a pair of regular expressions explained in the following.

Denote with Γ the alphabet consisting of all symbols that may appear in a computation history. If Σ and Q are the M's tape alphabet and states, then $\Gamma = \Sigma \cup Q \cup \{\#\}$. We assume that all configurations have length n^k and are padded on the right by blank symbols if they otherwise would be too short. Let's suppose for a moment that we have organized some configurations in a tableau where each row of the tableau contains a configuration and we mark the begining and the end of each one by the marker #. Now, in this organization we consider all the 2×3 windows. A window is legal if that window does not violate the actions specified by the M's transition function. In other words, a window is legal if it might appear when each configuration correctly follows another. By a proved claim in the proof of the Cook-Levin Theorem we know that, if the top row of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one. We encode a set of configurations $C_1 \ldots C_l$ as a single string, with the configurations separated from each other by the # symbol as shown in the following figure.

$$\#\underbrace{\quad}_{C_1}\#\underbrace{\quad}_{C_2}\#\cdots\#\underbrace{\quad}_{C_l}\#$$

Now, we can describe the set of words in Γ^* with at least one illegal window with the following regular expression.

$$\Gamma^* B \Gamma^*$$

where

$$B = \bigcup_{bad(abc,def)} abc\Gamma^{(n^k-2)}def.$$

Clearly, the set of configuration sequences with no illegal windows is described by

$$(\Gamma^* B \Gamma^*)^c.$$

What we need now, is be able to extract from the set of sequences of this form, an accepting computation history for the input w. We already have assured that there is not any illegal window. After that, we need two more things: the start configuration C_1 must be

$$\#q_0 w_1 \dots w_n \underbrace{\sqcup \dots \sqcup}_{n^k-n} \#.$$

where $w = w_1 \dots w_n$, and there must appear a symbol q_{accept}. We encapture the condition about C_1 by the regular expression

$$A_1 = \#q_0 w_1 \dots w_n \sqcup^{n^k-n} \#\Gamma^*,$$

and the condition that there should be an accepting configuration by the regular expresion

$$A_2 = \Gamma^* q_{accept} \Gamma^*.$$

Putting A_1 and A_2 together we have the following regular expression

$$A = A_1 \cap A_2 = \#q_0 w_1 \dots w_n \sqcup^{n^k-n} \#\Gamma^* q_{accept} \Gamma^*.$$

Summarising, there is an accepting computation of M on input w if and only if

$$A \cap (\Gamma^* B \Gamma^*)^c \neq \emptyset.$$

We finish the proof by emphasizing that the size of the above expression is polynomial. ■

We are now in a position to prove the following result.

Theorem 18 *There exist regular languages L and M, such that the exhaustive replacement $L \triangleright M$ cannot be computed in polynomial time, unless PTIME = PSPACE.*

Proof. Suppose that given two regular expressions A and B on alphabet Γ we like to test the emptiness of $A \cap (\Gamma^* B \Gamma^*)^c$. Without loss of generality let us assume that there exists one symbol in A that does not not appear in B. To see why even with this restriction the above problem of emptiness is still PSPACE complete, imagine that we can simply have a tape symbol which does not appear at all in the definition of the transition function of the Turing machine. Then this symbol will appear in the above set A but not in B (see Appendix). Let us denote this special symbol with †. We substitute the † symbol in A with the regular expression B. The result will be another regular expression A' which has polynomial size. Clearly, $A \cap (\Gamma^* B \Gamma^*)^c = A \cap (A' \triangleright B)$.

As a conclusion, if we had a polynomial time algorithm producing a polynomial size representation for $A' \triangleright B$, we could polynomially construct an NFA for $A \cap (A' \triangleright B)$. Then we could check in NLOGSPACE the emptiness of this NFA. This means that, the emptiness of $A \cap (\Gamma^* B \Gamma^*)^c$ could be checked in PTIME, which is a contradiction, unless PTIME=PSPACE. ∎

Corollary 3 *The algorithm in the proof of Theorem 11 for computing the partial p-rewriting of a query Q using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, is essentially optimal.*

Theorem 19 *Given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the partial l-rewriting can be computed in 2EXPTIME.*

Proof. Let us refer to the constructive proof of the Theorem 12. To compute the complement Q^c of the query is exponential. To transduce it to $T(Q^c)$ is polynomial. To complement again is exponential. So, in total we have 2EXPTIME. To compute the mask is EXPTIME and to intersect is polynomial. Finally, 2EXPTIME + EXPTIME = 2EXPTIME. ∎

For the partial lower rewriting we have the following.

Theorem 20 *Algorithm 1 for computing the partial l-rewriting of a query Q using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, is essentially optimal.*

Proof. Polynomially intersect the partial l-rewriting with Ω^* and get the l-rewriting of [CGLV99]. But, the l-rewriting is optimally computed in doubly exponential time in [CGLV99], so our algorithm is essentially optimal. ■

Chapter 5

Conclusions and Future Directions

5.1 Contributions

In this thesis we study the problem of query rewriting using views for semistructured data. The corresponding problem for relational data has already received very much attention. However, in the context of semistructured data the first results on the topic are those of [CGLV99,CGLV2000]. We studied the problem in two realistic scenarios. The first one is related to information integration systems, in which the data sources are modelled as sound views over a global schema. Our contribution in this setting is the establishment of the connections between **algebraic approximations** and **answers**. Since, computing the answer of the query is **polynomial**, using rewritings, we can use them for optimized data integration. Namely,

- We prove that in the case of **sound views**, the answer to the query, computed using the rewriting of Calvanese *et. al.*, is a **subset** of the certain answer.
- We give a polynomial 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 the possible pairs needed to be fed to the intractable decision procedure of

The second scenario, is query optimization using cached views and existing database. In this setting we propose two kinds of algebraic rewritings that focus on extracting as much information as possible from the views, for the purpose of optimizing regular path queries. Sumarizing, our contributions in this setting are:

- We propose algorithms for query answering that make use of **all** the information provided by the cached views.
- Although extracting all the information from the views is very expensive, the complexity is limited in the length of the view and query expression only.
- Since the data complexity for using the rewritings is polynomial, we reduce data complexity into expression complexity.

5.2 Future Work

We are often willing to live with structural information that is approximate. In other words the semistructured data represented by a graph database can be an approximation of the real world rather than an exact representation. On the other hand the user by himself can have an approximate idea and/or knowledge about the world, and this has as a consequence a need for non exact information to be exctracted from the database. In both cases the conclusion is that we need to deal with approximate queries and databases, and give approximate answers to the user queries.

If we consider the database graph to be the Web-graph then the current search engines already deal with approximate matching of specific words or sentences against the HTML text of the nodes. The result is usually ranked with regard to the degree of proximity and then presented to the user. However, consider a scenario where the links in the HTML pages are labeled by some predicates, and we like to find not only specific HTML pages containg some given text, but also we want these pages to

be linked by a path on which the link label sequence conforms to a given language. Current search engines do not give the user the option to approximately query the Web-graph through regular expressions. The same is true also for the query languages for semistructured data; they do provide means to specify paths of edge labels through regular expressions, but they do not have capabilities to specify approximate paths.

One of our future research directions will be to establish a general framework for approximate reasoning in semi-structured databases. This general framework should be flexible enough to be tailored to the specific demands of the users or applications and should tackle both approximate queries and approximate databases.

Bibliography

[Abi97] S. Abiteboul. Querying Semistructured Data. *Proc. of ICDT* 1997 pp. 1-18.

[ABS99] S. Abiteboul, P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and Xml.* Morgan Kaufmann, 1999.

[AD98] S. Abiteboul, O. M. Duschka. Complexity of Answering Queries Using Materialized Views. *Proc. of PODS* 1998 pp. 254-263

[AHV95] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. L. Wiener. The Lorel Query Language for Semistructured Data. *Int. J. on Digital Libraries* 1997 1(1) pp. 68-88.

[Bun97] P. Buneman. Semistructured Data. *Proc. of PODS* 1997, pp. 117-121.

[BDFS97] P. Buneman, S. B. Davidson, M. F. Fernandez and D. Suciu. Adding Structure to Unstructured Data. *Proc. of ICDT* 1997, pp. 336-350.

[Brzo64] J. A. Brzozowski. Derivatives of Regular Expressions. *JACM 11(4)* 1964, pp. 481-494

[BL80] J. A. Brzozowski and E. L. Leiss. On Equations for Regular Languages, Finite Automata, and Sequential Networks. *TCS 10* 1980, pp. 19-35

[CGLV99] 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.

[CGLV2000] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of ICDE* 2000, pp. 389-398.

[CGLV2000] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing for Regular Path Queries with Inverse. *Proc. of PODS* 2000, pp. 58-66.

[CSS99] S. Cohen, W. Nutt, A. Serebrenik. Rewriting Aggregate Queries Using Views. Proc. of PODS 1999, pp. 155-166

[Con71] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall 1971.

[DFF+99] 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.

[DG97] O. Duschka and M. R. Genesereth. Answering Recursive Queries Using Views. *Proc. of PODS* 1997, pp. 109-116.

[FS98] M. F. Fernadez and D. Suciu. Optimizing Regular path Expressions Using Graph Schemas *Proc. of ICDE* 1998. pp. 14-23.

[FLS98] D. Florescu, A. Y. Levy, D. Suciu Query Containment for Conjunctive Queries with Regular Expressions *Proc. of PODS* 1998. pp. 139-148.

[GM99] G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. *Proc. of ICDT* 1999 pp. 332-347.

[GT2000] G. Grahne and A. Thomo. An Optimization Technique for Answering Regular Path Queries. *Proc. of WebDB* 2000.

[HU79] J. E. Hopcroft and J. D. Ullman *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley 1979.

[HRS76] H. B. Hunt and D. J. Rosenkrantz, and T. G. Szymanski, On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages. *Journal of Computing and System Sciences* 12(2) 1976, pp. 222-268

[Kari91] L. Kari. *On Insertion and Deletion in Formal Languages.* Ph.D. Thesis, 1991, Department of Mathematics, University of Turku, Finland.

[Lev99] A. Y. Levy. *Answering queries using views: a survey.* Submitted for publication 1999.

[LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *Proc. of PODS* 1995, pp. 95-104.

[MW95] A. O. Mendelzon and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp. 24:6,* (December 1995).

[MMM97] 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.

[MS99] T. Milo and D. Suciu. Index Structures for Path Expressions. *Proc. of ICDT,* 1999, pp. 277-295.

[PV99] Y. Papakonstantinou, V. Vassalos. Query Rewriting for Semistructured Data. proc. of SIGMOD 1999, pp. 455-466

[Ull97] J. D. Ullman. Information Integration Using Logical Views. *Proc. of ICDT* 1997, pp. 19-40.

[Var88] M. Y. Vardi. The universal-relation model for logical independence. *IEEE Software.*

[Yu97] S. Yu. Regular Languages. In: *Handbook of Formal Languages.* G. Rozenberg and A. Salomaa (Eds.). Springer Verlag 1997, pp. 41-110