QUERY OPTIMIZATION 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 Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

July 2003

Abstract

Query Optimization Using Views in Semistructured Databases

Alex-Imir Thomo, Ph.D. Concordia University

A wealth of query languages for semistructured graph data has emerged, and what almost all of these languages have in common is a graph navigation mechanism, which is not usually found in their relational predecessors. However, the navigation is very expensive since it typically involves many database or network accesses. As a consequence, the optimization of the navigational part of the queries is essential for having commercially attractive query processors similar to those for relational data.

One of the most well known ways for optimizing a query in general, is to use available information in precomputed or materialized views. At the heart of our approaches is the leverage of the concept of query rewritings using views. The query rewriting is a well known problem, which has been solved and deeply investigated for (non-recursive) queries over relational data. However, for navigational queries, which are a subset of the bigger family of the recursive datalog queries, the problem of rewriting is more complex and challenging.

In this thesis 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 modeled as sound views over a global schema. In such cases the "real" database is not available and what we try to compute for a query is its "certain answer according to the views." The second scenario, is query optimization using views when the real database is available, but the views are cheaper to access. In such a case we can use the database to answer parts of the query for which there are no relevant views. We propose algebraic rewritings that focus on extracting as much information as possible from the views for optimizing navigational queries.

To my wife Mirela

Acknowledgments

I want to express my deepest gratitude to my supervisor Dr. Gösta Grahne for so closely guiding me through all our research. This thesis would not have been possible without our long discussions about every detail and his exceptional way of encouraging and advising me. Time by time, I needed a talk with him in order to find the energy to continue.

I also want to express my gratitude to Professor Jaroslav Opatrny, that in a very smooth way introduced me to the "hard" Theory of Computing.

I want to especially thank Professor Moshe Vardi for accepting to be one of the committee members. It is a great honor for me to have this thesis being read and reviewed by Professor Moshe Vardi, whose groundbreaking work on the databases enlightened me many times during the PhD.

Last but not least, I would like to thank my wife Mirela that created my nicest time ever. Without her the PhD would have been much harder.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and aim

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 hypertext data in this way, and we are now starting to see the development of software that extracts and stores data from the Web.

The emergence of XML (eXtensible Markup Language) as a standard for data representation on the Web is expected to greatly facilitate the publication of electronic data by providing a simple syntax that is both human and machine-readable. Since its introduction, XML 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.

Both hypertext (HTML) and XML data are modeled as labeled graphs. In data published as HTML, the pages can naturally be considered as the graph nodes and

the HREF links between them as the graph edges. Since the HREF links are labeled, the representing graph is labeled as well.

Let's consider now the XML data. By just a look at any common XML fragment, a tree modeling comes also very naturally in mind. It is the nested structure of XML that directly suggests the model. Intuitively, each element is a node and its children are the attributes, which can be by themselves complex elements with their own further nesting. Moreover, because of the possible links between elements, in fact, the tree is generalized into a graph model.

In both the above database contexts, navigational queries have been proven to be very useful. Typically, navigational queries are expressed as regular expressions denoting paths in the graph representing the data. Such queries are called "regular path queries."

For an example in the context of hypertext data consider the multisite Web of Ericsson Inc. at *http://www.ericsson.com*. As we can easily verify, there are hundreds of Web pages full of information and we naturally would like to be helped by some navigational mechanism, instead of endlessly manually browsing. For the appropriate abstraction level, let there be a function mapping the HREF-links to symbols of an alphabet. For example, links leading to the Ericsson Canada site could be labeled with *"canada."* Suppose now that the user wants to learn "technical" news about *Bluetooth* developments at Ericsson Canada. Unfortunately, he has to do an extensive browsing for that. He has to read fetched pages, then follow relevant links in those pages, pressing the "back" button nervously because he probably followed the wrong path a couple of times, until he finally reaches the right page; that with the Canadian white papers about *Bluetooth*. Wouldn't it be better to just say *canada* . *products* . _* . *mobile internet* . _* . *white papers* . *pdf* and leave the system to do the browsing for him?

Queries of a navigational nature are also very essential for XML data. To illustrate this let's consider an organisation which publishes data about books, articles and software. For instance a fragment of the organisation XML database would be the following.

```
<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>
```

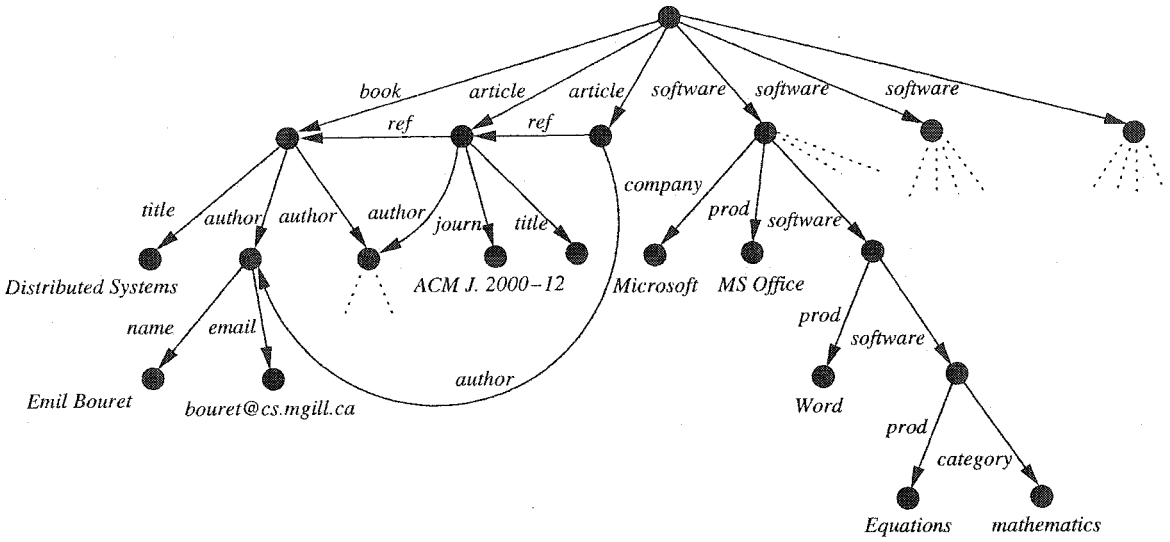

Figure 1: XML as a graph.

Intuitively, this XML database can be represented as the graph of Figure 1.

Suppose now that the user is a mathematician who wants to know what software is intended primarily for mathematicians. He would try "greping" huge XML files with the UNIX *grep* utility for the word "mathematics." However, this is a very unsatisfactory solution because his intention is not only to find out where the word "mathematics" appears. What he really wants is "mathematics" to appear inside a "software." Clearly, a keyword search is not a structural search.

Another problem with the "greping" solution is that although XML can exists in text files, we emphasize here that XML should be considered only as a logical representation of the data. The days of text XML will soon be over. Almost all the major database vendors are now using the XML as a front-end to their somehow "unpopular" object-relational features. Also, we are starting to see the emergence of native XML repositories that semantically "shred" XML into disk blocks, trying to provide the usual database characteristics. As a consequence, the "greping" solution would only work if the whole XML "file" is reassembled and brought to the user, and this is obviously not an acceptable solution.

Clearly in such cases, querying through regular expressions provides an elegant solution to the problem. Returning to our concrete example, a typical query that

satisfies our hypothetical mathematician user would be *software** . *category* . *mathematics*, where the node value "mathematics" is considered as a self-loop to the node itself.

We motivated in the above the usefulness of regular expressions for navigational querying of the data. However, note that we are querying huge persistent graph structures. This said, it is not difficult to see that the evaluation of regular expressions is a slow process and far more complex than the evaluation of first order queries on relational tables involving a few join steps aided by fast relational index structures. The navigation often comprises prohibitive database or network accesses. Adding here the inherent recursion present in regular expressions, we can easily realize that the traversed paths by a typical navigation can be arbitrarily long and very costly.

Obviously, the optimization of navigational query evaluation on data graphs is an imperative matter. This thesis follows the paradigm of query optimization using precomputed or materialized views. We investigate the theoretical ground for using views to rewrite arbitrary regular path queries. Notably, the rewritings that we obtain can be used to optimize more general queries like those belonging to the wider family of conjunctive regular path queries.

We present a detailed overview of our work in the next section, after we introduce some necessary definitions.

1.2 Databases, queries and views

As mentioned before, we abstract the HTML and XML data by a labeled data-graph. In general, the data being modeled by graphs are called *semistructured.*

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 labeled 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 asking for subgraphs of the database that match the given pattern [MW95, FLS98, CGLV99, CGLV2000a, CGLV2000b]. 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 labeled *article*, followed by any number of other edges, then by an edge labeled *ref* and finally by an edge labeled with *ullman* or *widom*.

Formally, we consider a *query* Q to be a finite or infinite regular language over Δ.

If there is a path labeled $R_1, R_2, \ldots, R_k$ 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

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

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

A view is the result of evaluating a query. The query that generates the view is called view definition. 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 there is a view name v_i. We call the set $\Omega = \{v_1, \ldots, v_n\}$ the *outer* or *view alphabet.* For each $v_i \in \Omega$, we set $\mathit{def}(v_i) = V_i$. The substitution *def* associates with each view name v_i, in the Ω alphabet, the language V_i. Also, we extend the substitution *def* to the Δ alphabet associating each symbol with itself. The substitution *def* is applied

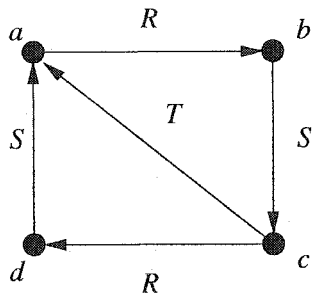

Figure 2: An example of a graph database

to words, languages, and regular expressions, over $\Omega \cup \Delta$ in the usual way (see e. g. [HU79]). Sometimes we need to refer to regular expressions representing the languages Q and V_i. In order to simplify the notation, we will blur the distinction between the regular expressions and the languages that they represent.

Given a database DB, we define the *view graph* $\mathcal{V}_{DB}$ to be the graph induced by the set

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

of Ω-labeled edges.

Example 2 *Let DB be the graph in Figure 2. Suppose that we have two views available $V_1 = \{SR\}$ and $V_2 = \{STR\}$. Then, $ans(V_1, DB) = \{(b, d), (d, b)\}$ and $ans(V_2, DB) = \{(b, b)\}$. It is easy to see that the view graph is as shown in Figure 3.*

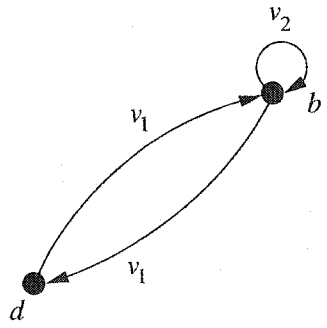

Figure 3: The view graph

1.3 View based rewritings

In data integration, data warehousing and query optimization, the problem of query rewriting using views is well known [LMSS95, Ull97, CGLV99, Lev2000]. Simply stated, the problem is to reformulate the query in terms of the view definitions.

Query rewriting in relational databases is by now rather well investigated. Several papers investigate this problem for the case of conjunctive queries [LMSS95, Ull97, PV99]. However, in the framework of semistructured data the problem of rewriting has received much less attention.

Let Q be a query and $\{V_1, \ldots, V_n\}$ be view definitions on an alphabet Δ. Consider the alphabet $\Omega = \{v_1, \ldots, v_n\}$ of corresponding view symbols. Formally, we define a rewriting to be a language on $\Omega \cup \Delta$. There are infinitely many rewritings, and the problem is to compute the most appropriate rewriting for a specific context. One of the criteria for reasoning about a rewriting Q' is the "relevance" to the given query. For this we naturally compute $def(Q')$, which is a language on Δ, and then "compare" it with the query language.

Definition 1 A rewriting Q' is *relevant* to a query Q if for each word $w \in Q'$ we have that $def(w) \cap Q \neq \emptyset$.

Definition 2 A rewriting Q' is *contained* in a query Q if for each word $w \in Q'$ we have that $def(w) \subseteq Q$.

Definition 3 A rewriting Q' is *exact* with respect to a query Q if $def(Q') = Q$.

Clearly, the exactness implies containment and containment implies relevance. Hence, any exact or contained rewriting is relevant.

On the other hand, taking into consideration only the relevance is not enough. To see this, recall that by definition, we also have that the query Q is an exact rewriting and completely relevant to itself. This indicates that a second criterion should be the

"optimality" of a rewriting. Obviously, we want to minimize the Δ symbols appearing in the words of a rewriting, since we assume that the view edges are cheaper to access [1].

Example 3 *Let $Q = (RR)T + (RR)S(RR) + T^5$ and suppose we have three views available: $V_1 = RR$, $V_2 = S + SS$, and $V_3 = T^5$. What could be a rewriting? We present six relevant rewritings that come to mind.*

$$\begin{aligned} Q^{(1)} &= v_1 v_2 v_1 + v_3 \\ Q^{(2)} &= v_1 T + v_1 v_2 v_1 + v_3 \\ Q^{(3)} &= v_3 \\ Q^{(4)} &= v_1 T + v_1 S v_1 + v_3 + T^5 \\ Q^{(5)} &= v_1 T + v_3 \\ Q^{(6)} &= v_1 T + v_1 S v_1 + v_3 \end{aligned}$$

The first rewriting $Q^{(1)}$ is (logically) obtained by the following procedure: Whenever there is a query *word* w such that $w \in V_i \dots V_j$, replace it with $v_i \dots v_j$ in the rewriting. Intutively, this rewriting is the (maximal) set of all Ω words that are relevant to the query. Formally, we have that $w \in Q^{(1)}$ iff $w \in \Omega^*$ and $def(w) \cap Q \neq \emptyset$. This rewriting turns out to be important for answering queries using views in information integration systems, where the database is not available. Since, this rewriting is an Ω language we can "evaluate" it on the view graph only. In **Chapter 2** [2], we present a construction for this rewriting and show that, if we consider it as a query on the view graph, then the answer that we compute will always contain the "certain answer" of the original query. Moreover, this will happen even when $Q \not\subseteq def(Q^{(1)})$.

The second rewriting $Q^{(2)}$ is the maximal and optimal relevant rewriting in $\Omega \cup \Delta$. To obtain it, we exhaustively do the following: Choose an arbitrary query word w. Whenever there is a *subword* x such that $x \in V_i$, for some $i \in [1, n]$, replace it with v_i in the rewriting. "Exhaustively" means that in any word of the rewriting there

[1] We will formally define the optimality in Chapter 3.

[2] The chapters 2 and 3 are a re-worked version of [Tho2000]. Also, different view points, discussions and proofs have been added since then.

cannot be a subword on Δ that is also a member of some view language. We present a construction for computing such rewritings in **Chapter 3**. In general, this rewriting is a mixed language on $\Omega \cup \Delta$. As such, we cannot evaluate it on the view graph only. We certainly need also the real database in order to evaluate the subwords on Δ. Based on the relationship $Q \subseteq def(Q^{(2)})$, (which is easy to verify) in Chapter 3, we will present an algorithm for using the rewritings of this kind in query optimization when we have both the view graph and the database graph and accessing the view graph is cheaper.

The next rewriting $Q^{(3)}$ is the maximal contained rewriting on Ω. This rewriting is presented in the seminal paper of Calvanese, De Giacomo, Lenzerini, and Vardi [CGLV99], and formally it is the set of *all* words w on Ω, such that $def(w) \subseteq Q$. Since this rewriting is a pure Ω language, it can be evaluated on the view graph only. This fact makes it appropriate to be used in the context of integration systems where the database is not available. As we have mentioned before, this is also true for $Q^{(1)}$, while the other rewritings, being "mixed" languages on $\Omega \cup \Delta$, are mainly aimed for the optimization in a context where both the view graph and the database graph are available.

The problem with the rewriting $Q^{(3)}$ is that it can happen to not be exact. To overcome this problem, a partial rewriting is introduced in [CGLV99]. This is the rewriting $Q^{(4)}$ in our example. The idea behind it, is to incrementally add new elementary one-symbol view definitions to the set of initial view definitions and then recompute the previous rewriting with the new set. This is repeated each time a new elementary view is added until an exact rewriting is obtained. The drawback of this approach is that sometimes "non-optimal" words can be computed as well in the rewriting. To see this, observe that in order to model the query word $(RR)T$, the algorithm is forced at some point to add T as a new elementary view definition. However, by doing so and then recomputing the maximal language on the (new) view alphabet, will result in computing also the "non-optimal" Δ word T^5, which could be avoided since it is exactly modeled by v_3.

The rewriting $Q^{(5)}$ is contained and "optimal." Intuitively, it will never contain some subword describing paths in the database that could have been "seen" before

by any of the views. This rewriting is studied in **Chapter 3**, and is formally defined as the $\subseteq$-largest subset of $Q^{(2)}$, that is a contained rewriting. However, as we can see from the example, this rewriting can happen to not be exact. Consequently, if $Q^{(5)}$ is not exact, then we are not guaranteed to obtain all the answers to a given query by using this rewriting even if we consult the database for evaluating the Δ subwords of the rewriting.

We have that $Q^{(3)} \subseteq Q^{(5)}$. This means that if we cannot avoid accessing the database, i.e. when $Q^{(3)}$ is not exact, then the rewriting $Q^{(5)}$ is at least as good as $Q^{(3)}$, and it can be in many cases better. To see this, observe that the only way $Q^{(3)}$ and $Q^{(5)}$ could be used for lossless query optimization is to compute $Q - def(Q^{(3)})$ and $Q - def(Q^{(5)})$ and then answer this part of the query un-helped directly on the database. Clearly, $Q - def(Q^{(5)})$ is smaller than $Q - def(Q^{(3)})$.

If we take a closer look in the example, we can see that the non-exactness of $Q^{(5)}$ stems from the fact that $v_1 S v_1$ is missing from the rewriting. This word is not there because it was filtered out from the "parent" rewriting $Q^{(2)}$ and computing the word $v_1 v_2 v_1$ instead, because the symbol S was considered replaceable by v_2 (recall $V_2 = S + SS$). However, this decision turns out to not allow $Q^{(5)}$ to be exact, since it is defined as a subset of $Q^{(2)}$, and being a contained rewriting cannot include $v_1 v_2 v_1$, whose substitution $def(v_1 v_2 v_1) \not\subseteq Q$.

It is clear from the above discussion that in order to have an exact rewriting we should not replace S by v_2 in $v_1 S v_1$. The rewriting $Q^{(6)}$ does exactly that. Generalizing the idea, we introduce in **Chapter 4** a taxonomy for comparing different contained rewritings. Then, we compute the "maximal and contained partial rewriting," which is the set of *all* words w on $\Omega \cup \Delta$ such that $def(w) \subseteq Q$ and there is no subword decomposition $w = xyz$, such that $y \in V_i$ and $xV_iz \subseteq Q$, for some $i \in [1, n]$. We prove that this rewriting is "optimal" and exact. If we cannot completely avoid accessing the database because $Q^{(3)}$ happens to not be exact, then the $Q^{(6)}$ rewriting is the most preferable rewriting for lossless (exact) query optimization. A question can arise here. Does $Q^{(6)}$ deprecate the use of $Q^{(5)}$? The answer is yes, but we still prefer to present the construction of $Q^{(5)}$ because its computational complexity is lower than that of $Q^{(6)}$, and hence could be used in cases when a completely lossless

optimization is not necessary.

The rewritings that we have presented so far may look different if we take into account eventual constraints that we might know or learn about the database on which the query is to be evaluated. For instance, if we knew in Example 3 that the query and the views are to be considered with respect to databases in which any two objects connected by a path spelling the word $(RR)S(RR)$ are also connected by a path spelling the word $(RR)T$, then we could say that $Q^{(6)} = Q^{(5)}$. The study of the rewritings under constraints is the focus of **Chapter 5**.

1.4 The maximally contained Ω-rewriting

In order to make the thesis self-contained we briefly present in this section the query rewriting ($Q^{(3)}$) of Calvanese, De Giacomo, Lenzerini, and Vardi [CGLV99]. It is the $\subseteq$-largest contained rewriting on Ω, and reflecting this property, we call it: the *Maximally Contained Rewriting* (MCR) on Ω. Formally, for a given query Q, the maximally contained rewriting $\text{MCR}_{\text{V}}(Q)$, is the set of all words on Ω such that their substitution through *def* is contained in the query language Q. In symbols,

$$\text{MCR}_{\text{V}}(Q) = \{w : w \in \Omega^* \text{ and } def(w) \subseteq Q\}.$$

An elegant method for computing the MCR of a query is given in [CGLV99]. Their algorithm is:

1. Construct a DFA $A_Q(\Delta, S, s_0, \delta, F)$ such that $Q = L(A_Q)$.
2. Construct automaton $B = (\Omega, S, s_0, \delta', S - F)$, where $(s_i, v, s_j) \in \delta'$ iff there exists $w \in V$ such that $(s_i, w, s_j) \in \delta^*$.
3. The MCR is the Ω language accepted by complementing the B automaton.

Step 2 can also be expressed slightly different: Consider each pair of states. If they are connected in A_Q by a walk labeled with a word in V_i, put a transition v_i between them in B.

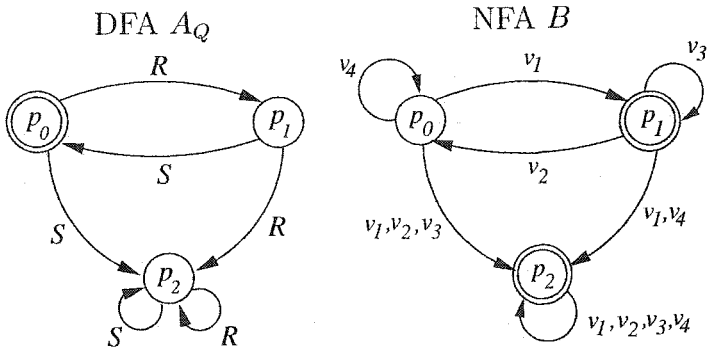

Figure 4: The DFA for the query and the corresponding "view" automaton.

Example 4 *Let $Q = (RS)^*$ and $V_1 = R + S^2$, $V_2 = S$, $V_3 = SR$, $V_4 = (RS)^2$. Then the minimal[3] DFA A_Q for the query Q is shown in Figure 4, left and the corresponding automaton B is shown in in Figure 5, right. The resulting complement automaton B^c is shown in Figure 5. Note that the "trap" and unreachable states have been removed for clearness.*

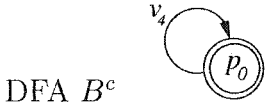

Figure 5: The resulting MCR given by DFA B^c.

Observe that, if B accepts an Ω-word $v_1 \cdots v_m$, then there exist m Δ-words w_1, $\ldots$, w_m such that $w_i \in V_i$ for $i = 1, \ldots, m$ and such that the Δ-word $w_1 \ldots w_m$ is rejected by A_Q. On the other hand if there exists a Δ-word $w_1 \ldots w_m$ that is rejected by A_Q such that $w_i \in V_i$ for $i = 1, \ldots, m$, then the Ω-word $v_1 \cdots v_m$ is accepted by B. That is, B accepts an Ω-word $v_1 \cdots v_m$ if and only if there is a Δ-word in $def(v_1 \cdots v_m)$ that is rejected by A_Q. Hence, B^c being the complement of B accepts an Ω-word if and only if all Δ-words $w = w_1 \ldots w_m$ such that $w_i \in V_i$ for $i = 1, \ldots, m$, are accepted by A_Q.

The complexity of computing the MCR of a regular path query Q is in 2EXPTIME and this bound is shown to be tight ([CGLV99]).

In order to use the MCR of a query Q for losless query optimization, we should

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

test its exactness. An algorithm for this is given in [CGLV99] and is as follows.

1. Construct an automaton $B = (\Delta, S_B, s_{B0}, \delta_B, F_B)$ 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 a contained rewriting of Q, $L(B) \subseteq Q = L(A_Q)$.
2. Check whether $L(A_Q) \subseteq L(B)$, that is, check whether $L(A_Q \cap B^c) = \emptyset$.

Theorem 1 *[CGLV99] The* MCR Q' *is an exact rewriting of the query Q with respect to a set* $\mathbf{V}$ *of regular view definitions, if and only if $L(A_Q \cap B^c) = \emptyset$.*

Corollary 1 *[CGLV99] An exact rewriting of Q with respect to* $\mathbf{V}$ *exists if and only if $L(A_Q \cap B^c) = \emptyset$.*

Theorem 2 *[CGLV99] 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.*

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

1.5 Finite transducers and rational relations

In most of our constructions, the concept of the (rational) transducer is instrumental. It provides a succint approach for transforming words and for mappings between languages.

A *finite transducer* $\mathcal{T} = (P, I, O, \delta, p_0, F)$ consists of a finite set of states P, an input alphabet I, and an output alphabet O, also a starting state p_0, a set of final states F, and a transition-output relation $\delta \subseteq P \times I^* \times P \times O^*$. Intuitively, for instance, $(p_0, u, p_1, w) \in \delta$ means that if the transducer is in state p_0 and reads word u it can go to state p_1 and emit the word w. For a given word $u \in I^*$, we say that a word $w \in O^*$ is an *output of* $\mathcal{T}$ *for* u if there exists a sequence $(p_0, u_1, p_1, w_1) \in \delta$, $(p_1, u_2, p_2, w_2) \in \delta$, $\ldots$, $(p_{n-1}, u_n, p_n, w_n) \in \delta$ of state transitions in $\mathcal{T}$, such that $p_n \in F$, $v = u_1 \ldots u_n$, and $w = w_1 \ldots w_n$. A finite automaton is simply a transducer without output, i.e. the tuples in the transition relation are triplets of the form (p, a, q) instead of quadruplets of the for (p, a, q, b).

An example of a finite transducer $(\{p_0, p_1, p_2\}, \Omega, \Delta, \delta, \{p_2\})$ is shown in Figure 6.

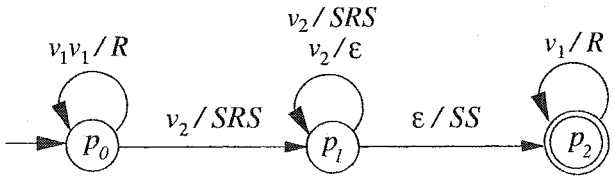

Figure 6: A finite transducer $\mathcal{T}$

We shall also use the symbol $\mathcal{T}$ to denote the set of all pairs $(v, w) \in I^* \times O^*$, where w an output of $\mathcal{T}$ for v. Finally, $\mathcal{T}$ can also be seen as a mapping from languages to languages, and we write

$$\mathcal{T}(L) = \{w : (v, w) \in \mathcal{T}, \text{ for some } v \in L\}.$$

It is well known that $\mathcal{T}(L)$ is regular whenever L is.

A finite transducer $\mathcal{T} = (P, I, O, \delta, p_0, F)$ is said to be in the *standard form*, if $\delta \subseteq S \times (I \cup \{\epsilon\}) \times S \times (O \cup \{\epsilon\})$. 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]).

Chapter 2

Query processing in information integration systems

2.1 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 heterogeneous data sources. 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 global schema.* A mediated global schema is set of virtual relations, in the sense that they are not stored anywhere. The mediated global 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, known as *local-as-view* (LAV), is to describe the contents of the local sources as *views* over the

mediated global 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. In a LAV architecture, a local change to a data source can be handled locally, by adding, removing or updating only the view definition concerning this source; therefore LAV scales very well.

For illustrating the problem, let the mediated schema be that of our bookstore database graph of Figure 1. In other words the virtual (binary) relations are those specified by the edge labels of the graph. However, recall that the real database does not exist; it is virtual.

Suppose now, that we have the following three data sources. The first provides a listing of (x, y) object pairs, 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 contains (x, y) pairs 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.

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

Finally, the third source contains (x, y) pairs 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 software is somehow related to some book or article,"* i.e. $Q = ref^*.software^*$, 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 on the views.

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

It is important to note here that this formulation of the query is to be answered on the view graph only, which could be as shown in Figure 7.

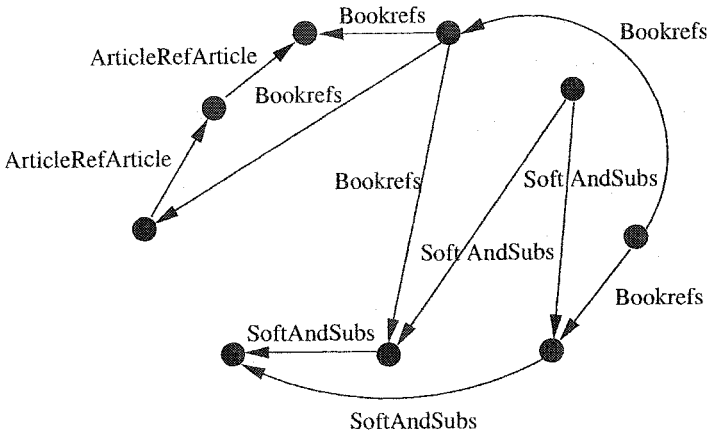

Figure 7: An example of a view graph

2.2 Formal background

Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be a set of view definitions and let $\Omega = \{v_1, \ldots v_n\}$ be the corresponding view alphabet. Consider $\mathcal{V}$ to be the view graph for the set of objects found in the views. As we have mentioned before $\mathcal{V}$ is a database over (D, Ω). In the terminology of data integration, a view graph is called a *source collection*, and we prefer to use this name in this chapter for compatibility with previous work on the subject. However, we retain the symbol $\mathcal{V}$ in order to clearly emphasize that a source collection and a view graph are essentially the same.

A source collection $\mathcal{V}$ defines a set $poss(\mathcal{V})$ of databases over (D, Δ) as follows (see [GM99]):

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

Suppose now the user gives a query Q on the database alphabet Δ, but we only have a source collection $\mathcal{V}$ available. This situation is the basic scenario in information integration (see e.g. [Ull97, LMSS95, GM99]). The best we can do is to approximate

by the *certain answer for* Q *using* $\mathcal{V}$.

$$cert_{\mathcal{V}}(Q) = \bigcap_{DB \in poss(\mathcal{V})} ans(Q, DB).$$

In [CGLV2000a], Calvanese, De Giacomo, Lenzerini and Vardi describe an exponential algorithm $\mathcal{A}_{Q,\mathcal{V}}(a, b)$ that returns "yes" or "no" depending on whether a given pair (a, b) is in the certain answer for Q or not. Since this problem is coNP-complete in the number of objects in $\mathcal{V}$ (data complexity), it is highly unlikely to find an essentially better algorithm for computing the certain answer.

The possibility of using rewritings for answering queries is (briefly) discussed in [CGLV2000a]. Since rewritings have proved to be highly successful in attacking the corresponding problem for relational databases [Lev2000], 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 $\mathcal{V}$. However, under the more realistic "sound view assumption" [1] adopted in this chapter we are only guaranteed to get a subset of the certain answer. The following propositions hold:

Theorem 3 *Let Q' be the maximally contained Ω-rewriting (MCR) of Q using* **V**. *Then, for each source collection $\mathcal{V}$ over* **V**,

$$ans(Q', \mathcal{V}) \subseteq cert_{\mathcal{V}}(Q).$$

PROOF. Let $(a, b) \in Q'(\mathcal{V})$ and let DB be an arbitrary database in $poss(\mathcal{V})$. Since $(a, b) \in Q'(\mathcal{V})$ 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 $\mathcal{V}$ such that $v_{i_1} \ldots v_{i_{k+1}} \in Q'$. Since $DB \in poss(\mathcal{V})$, 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 4 *There is a query Q and a set of view definitions* **V**, *such that there is an exact rewriting Q' of Q using* **V**, *but for some source collections $\mathcal{V}$, the set $ans(Q', \mathcal{V})$ is a proper subset of $cert_{\mathcal{V}}(Q)$.*

[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 [Var88].

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 by evaluating the MCR on the source collection $\mathcal{V}$ we get a subset of the certain answer. In the next section we describe a "possibility" rewriting Q'' of Q using $\mathbf{V}$, such that for all source collections $\mathcal{V}$:

$$cert_{\mathcal{V}}(Q) \subseteq ans(Q'', \mathcal{V}).$$

2.3 The possibility rewriting

We will compute the biggest relevant rewriting on Ω. We call this rewriting the *possibility rewriting*, and denote it PR. Intuitively, the PR is the set of *all* words on the view alphabet Ω, such that their substitution by *def* contains at least a word in Q. Formally, a language Q'' on Ω is the PR of Q if

$$Q'' = \{w : w \in \Omega^* \text{ and } def(w) \cap Q \neq \emptyset\}.$$

The possibility rewriting has the following desirable property:

Theorem 5 *$cert_{\mathcal{V}}(Q) \subseteq ans(Q'', \mathcal{V})$, for each source collection $\mathcal{V}$.*

PROOF. Assume that there exists a source collection $\mathcal{V}$ and a pair $(a, b) \in cert_{\mathcal{V}}(Q)$, such that $(a, b) \notin ans(Q'', \mathcal{V})$. Since the pair (a, b) is in the certain answer of the query Q, it follows that for each database $DB \in poss(\mathcal{V})$ there is a path $a \xrightarrow{w} b$, where $w \in Q$. Now, we will construct from $\mathcal{V}$ a database $DB_{\mathcal{V}}$ such that $ans(Q, DB_{\mathcal{V}}) \not\ni (a, b)$. For each edge labeled v_i from one object x to another object y in $\mathcal{V}$ we choose an arbitrary word $w_i \in def(v_i)$ and put in $DB_{\mathcal{V}}$ the "new" objects $c_1, \ldots, c_{k-1}$, where k is the length of w_i, and a path $x, c_1, \ldots, c_{k-1}, y$ labeled with the word w_i. Each

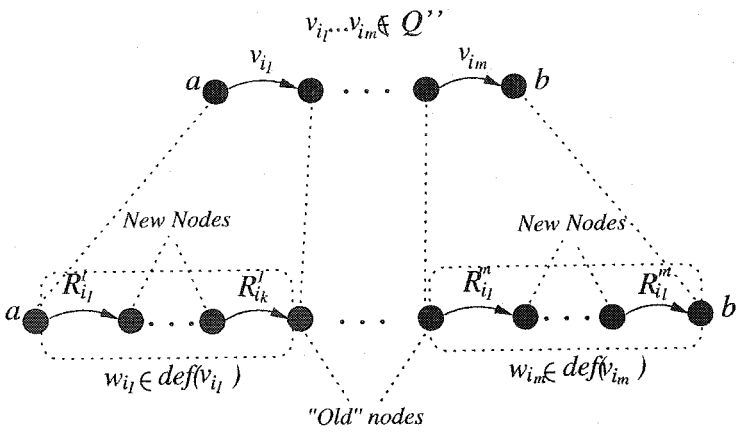

Figure 8: Visualisation of the proof for Theorem 5.

time we introduce "new" objects, so all the constructed paths are disjoint. Obviously, $DB_{\mathcal{V}} \in poss(\mathcal{V})$. It is easy to see that $ans(Q, DB_{\mathcal{V}}) \not\ni (a, b)$ because otherwise there would be a path $v_{i_1} \dots v_{i_m}$ in $\mathcal{V}$ 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 $(a, b) \in ans(Q'', \mathcal{V})$, From the fact that $ans(Q, DB_{\mathcal{V}}) \not\ni (a, b)$ it then follows that $cert_{\mathcal{V}}(Q) \not\ni (a, b)$; a contradiction. For a visualisation of the proof see Figure 8. ■

It is worth noting here that Theorem 5 shows that $ans(Q'', \mathcal{V})$ contains the certain answer to the query Q even when algebraically $def(Q'') \not\supseteq Q$.

Now, we want to reason about the other tuples (pairs), which are not in the $cert_{\mathcal{V}}(Q)$, and which we can obtain by the evaluation of the possibility rewriting on the source collection. We show that although those tuples are not in the $cert_{\mathcal{V}}(Q)$, they are at least "possible" in the sense that there exist databases, which are consistent with the views, and such that if we evaluate the query on them, we obtain the afore mentioned tuples.

Theorem 6 *Let $(a, b) \in ans(Q'', \mathcal{V})$. Then, there exist a database $DB_{\mathcal{V}}$, such that $(a, b) \in ans(Q, DB_{\mathcal{V}})$.*

PROOF. From the source collection $\mathcal{V}$, we build the database $DB_{\mathcal{V}}$ by replacing each edge v_i by an automaton for V_i. We introduce "new" objects for representing the

internal states of the automaton, while we represent the initial state, and final state [2] by the the source and destination node of the edge labeled by v_i. Clearly, $DB_{\mathcal{V}}$ is consistent with the views. Now, since $(a, b) \in ans(Q'', \mathcal{V})$, there exists a path spelling $v_{i_1} \ldots v_{i_m} \in Q''$, from a to b in $\mathcal{V}$. From the fact $v_{i_1} \ldots v_{i_m} \in Q''$, we get that $V_{i_1} \ldots V_{i_m} \cap Q \neq \emptyset$, i.e. there exists a word $w \in V_{i_1} \ldots V_{i_m} \cap Q$. Now, from the construction of $DB_{\mathcal{V}}$, we have that between a and b, for each word in $V_{i_1} \ldots V_{i_m}$, there exists a path spelling it. So, we can also find a path spelling w, and this means that $(a, b) \in ans(Q, DB_{\mathcal{V}})$. ■

2.4 Computing the possibility rewriting

Our aim in this section is to construct a transducer $\mathcal{T}$, such that $\mathcal{T}(Q) = Q''$ (PR of Q). We start with one node representing both the starting state and the final state. Then we 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, ..., 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 9, 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 9, right). Third, we get rid of "macro-transitions" of the form v/E^*. Suppose we have

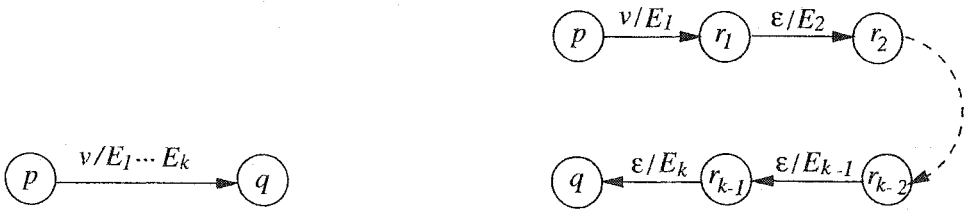

Figure 9: Decomposing a "macro" transducer I.

an edge labeled v/E^* from p to q in the "macro-transducer." (See Figure 10, left).

[2] Without loss of generality, we assume that the automaton for V_i is one with a single final state.

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 10, right.

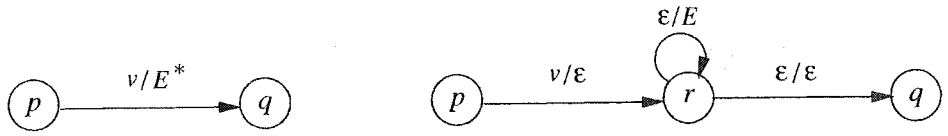

Figure 10: Decomposing a "macro" transducer II.

At the end, we interchange the input and output of the transitions. Let's call the resulting transducer $\mathcal{T}$. The following theorem is true.

Theorem 7 *For the* PR Q'', *we have that* $Q'' = \mathcal{T}(Q)$.

PROOF. Recall that by definition $Q'' = \{w : w \in \Omega \text{ and } def(w) \cap Q \neq \emptyset\}$ and $\mathcal{T}(Q) = \{w : (u, w) \in \mathcal{T}, \text{ for some } u \in Q\}$.

"$\subseteq$" Let $w = v_{i_1} \dots v_{i_k}$ be in Q''. This means that there exists a word u in $V_{i_1} \dots V_{i_k} \cap Q$. From the construction of $\mathcal{T}$, the automaton that we get if we ignore the output of the transitions accepts the language $(V_1 + \dots + V_n)^*$. Hence, we have that $(u, w) \in \mathcal{T}$. Now, since u is also in Q, we finally have that $w \in \mathcal{T}(Q)$.

"$\supseteq$" Let $w = v_{i_1} \dots v_{i_l}$ be in $\mathcal{T}(Q)$. Since $w \in \mathcal{T}(Q)$, we have that there exists a word $u \in Q$, such that $(u, w) \in \mathcal{T}$. Now, from the construction of $\mathcal{T}$ we have that $u \in V_{i_1} \dots V_{i_l}$. This means that $V_{i_1} \dots V_{i_l} \cap Q \neq \emptyset$. From the last, $w \in Q''$ follows.

■

We now describe an algorithm that given a regular language L and finite transducer $\mathcal{T}$ constructs a finite state automaton that accepts the language $\mathcal{T}(L)$. For this, let $A = (P_A, I, \delta_A, p_0, F_A)$ be an ϵ-free NFA that accepts L, and let $\mathcal{T} = (P_\mathcal{T}, I, O, \delta_\mathcal{T}, q_0, F_\mathcal{T})$ be a transducer in standard form. Then, we construct an NFA:

$$A_{\mathcal{T}(L)} = (P_A \times P_\mathcal{T}, O, \delta_{\mathcal{T}(L)}, (p_0, q_0), F_A \times F_\mathcal{T}),$$

where $\delta_{\mathcal{T}(L)}$ is defined by,

$$\begin{aligned}\delta_{\mathcal{T}(L)} &= \{((p,q),v,(p',q')) : (p,R,p') \in \delta_A \text{ and } (q,R,q',v) \in \delta_T\} \\ &\cup \{((p,q),v,(p,q')) : (q,\epsilon,q',v) \in \delta_T\}\end{aligned}$$

Theorem 8 *([Yu97]) The automaton $A_{\mathcal{T}(L)}$ accepts exactly the language $\mathcal{T}(L)$.*

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

Corollary 2 *Given a query Q and a set $\mathbf{V}$ of view definitions, the possibility rewriting of Q using $\mathbf{V}$ can be effectively constructed.* ■

Example 5 Let the query be $Q = (RS)^*$. and the views be $V_1 = R + S^2$, $V_2 = S$, $V_3 = SR$, and $V_4 = (RS)^2$. The corresponding view alphabet is $\Omega = \{v_1, v_2, v_3, v_4\}$.

The DFA [3] A accepting the query Q is given in Figure 11 (left), and the transducer characterizing the views is given in Figure 11 (right). We transform the transducer into standard form (Figure 12, left), and then interchange the input with output (Figure 12, right). The constructed automaton is shown in Figure 13, 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.

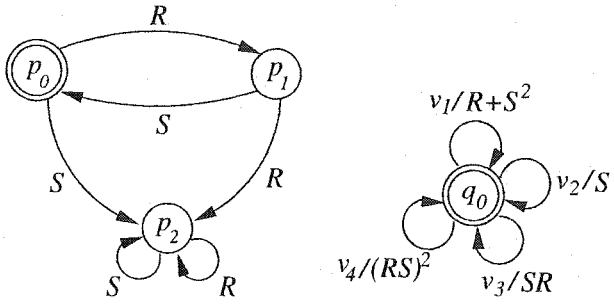

Figure 11: Query automaton and macro transducer.

[3] An ϵ-free NFA would do as well.

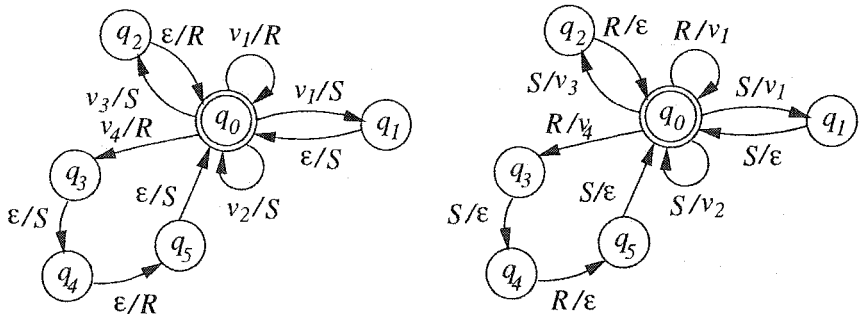

Figure 12: Transducers for the view definitions.

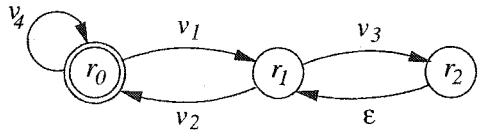

Figure 13: Automaton of the PR.

Our algorithm computes the PR Q'' represented by $(v_4 + v_1 v_3^* v_2)^*$, and the algorithm of [CGLV99] computes the MCR Q' represented by v_4^*. Now, suppose that the source collection $\mathcal{V}_n$ is induced by the following set of labeled edges (see Figure 14):

$$\begin{aligned} \{(i, v_1, a_i) &: 1 \leq i \leq n-1\} \cup \\ \{(a_i, v_2, i+1) &: 1 \leq i \leq n-1\} \cup \\ \{(a_i, v_3, a_{i+1}) &: 1 \leq i \leq n-1\} \cup \\ \{(i, v_4, i+2) &: 1 \leq i \leq n-2\} \end{aligned}$$

We can now compute

$$ans(Q'', \mathcal{V}_n) = \{(i, j) : 1 \leq i \leq n-1, i \leq j \leq n\},$$

and

$$ans(Q', \mathcal{V}_n) = \{(i, 2k) : 1 \leq i \leq n-1, 0 \leq k \leq n/2\}.$$

■

Finally, regarding the computational complexity for computing the possibility rewriting, it is easy to verify the following theorem.

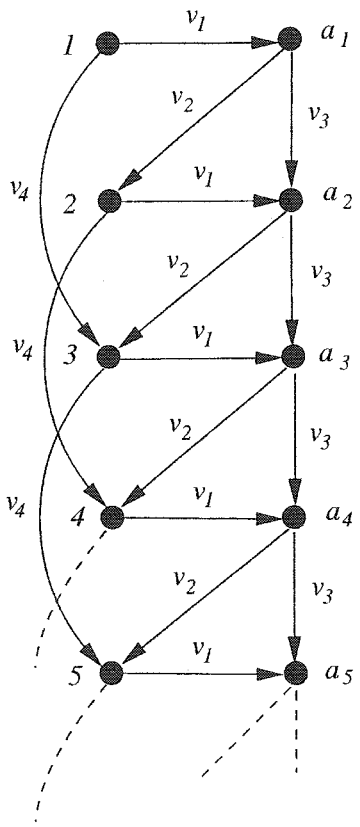

Figure 14: Source collection for Example 5

Theorem 9 *The automaton characterizing the* PR *can be built in time polynomial in the size of the regular expression representing the query and the size of the regular expressions representing the views.* ■

Chapter 3

Query optimization using cached views: First attack

3.1 Introduction

Based on practical observations, the most expensive part of answering queries on semistructured data is finding those graph patterns described by regular expressions. This is because a regular expression can describe arbitrarily 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. Such an optimizer can be used for the broader class of full fledged query languages for semistructured data.

In semistructured data, as well as in other data models such as relational and "object oriented", the importance of utilizing views for query optimization is well recognized [LMSS95], [CGLV99], [Lev2000]. Clearly, by having available information on relevant views we can greatly enhance the query evaluation with respect to efficiency.

As we have already seen, the notion of query rewriting in terms of the view definitions is instrumental for taking into account the information available in the

views for query answering. Hence, the first question that comes to mind is whether the rewritings that we have presented so far, namely the MCR of [CGLV99] and the PR, can help in the process of query optimization. In fact, they can surely be used for this purpose, but as we show in this chapter, we can definitely do better.

The problem with the above mentioned rewritings is that, since the context was data integration where the "real" database is usually not available, the rewritings were pure Ω languages in order to allow their evaluation on the view graph only. In other words, those rewritings model –using views– only complete words of the query. But in practice, the cases in which we can infer from the views full words for the query, are very rare. The views can cover *partial* words that can be satisfactorily long for using them in optimization, but if they are not complete words, they are ignored by the above mentioned rewritings. It would therefore be desirable to have partial rewritings in order to capture and utilize all the information provided by the views.

The problem of computing a partial rewriting is initially treated by Calvanese, De Giacomo, Lenzerini, and Vardi in [CGLV99]. There this problem is considered as an extension of the complete rewriting, enriching the set of the views with new elementary one-symbol views, chosen among the database relations (or symbols). The choice of the new elementary views is done in a brute force way, using a cost criterion depending on the application. However, there are cases when the algorithm of [CGLV99] for computing partial rewritings gives "too much" in the sense illustrated by $Q^{(4)}$ in the Example 3. It essentially contains redundant un-rewritten words and subwords from the query.

In this chapter we use a different approach for computing partial rewritings. For each word in the query language we exhaustively replace the *subwords* that appear as words in any of the view languages. In this spirit we compute two partial rewritings. The first is the biggest "exhaustive" relevance rewriting. The other is a contained one. Namely, it is the biggest contained subset of the first rewriting. Both of them have been illustrated in Example 3 by $Q^{(2)}$ and $Q^{(5)}$ respectively, and their properties have also been briefly discussed after.

The partial rewritings can be used to optimize the query evaluation by using the

following simple idea. If a query subword is to see a database sub-path that a view has traversed before, we use that view for evaluation. Generalizing this idea we present two query answering algorithms that are "lazy" in the sense that they access the database only when necessary. For the "been there" subpaths our algorithms use the views. Note that we do not materialize any new view; we only consult the database "on the fly," as needed.

The outline of the chapter is as follows. In Section 3.2 we recall the definitions of the MCR and PR for a query, and present a warm-up example for which the MCR and PR are empty, while the partial information provided by the views is no less than 99% of the complete "missing" information. In Section 3.3 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 3.4 we present the exhaustive partial possibility rewriting that is a generalization of the previously introduced exhaustive replacement operator. In Section 3.5 we define a partial contained rewriting, which as mentioned before is related to the partial possibility rewriting. In Section 3.6 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 rewritings. The computational complexity is studied in Section 4.5. We show that, although exponential, the algorithms proposed for computing the exhaustive possibility and contained partial rewritings are essentially (or almost) optimal.

3.2 Warm-up

Let us recall the definition of the maximally contained rewriting MCR and the possibility rewriting PR for a given query Q using a set of view definitions $\mathbf{V} = \{V_1, \ldots, V_n\}$

with $\Omega = \{v_1, \ldots, v_n\}$ as the corresponding view alphabet.

The MCR is the set $\{w : w \in \Omega^* \text{ and } def(w) \subseteq Q\}$. The containment condition could be very strong in practice, and can result in very small or empty rewritings. By relaxing this condition we got the possibility rewriting PR. The PR is the set $\{w : w \in \Omega^* \text{ and } def(w) \cap Q \neq \emptyset\}$.

However, both the MCR and PR of a query could be empty. Suppose for example that query Q is $Q = R_1 \ldots R_{100}$ and we have available two views V_1 and V_2, where $V_1 = R_1 \ldots R_{49}$ and $V_2 = R_{51} \ldots R_{100}$. It is easy to see that the MCR and PR are empty, while 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.

3.3 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 [1] language, both over some alphabet Δ, and let $\dagger$ be a symbol outside Δ. 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 Δ 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:

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

[1] The restriction for M to be ϵ-free is not really important because it is difficult to envision an application where we want to obtain a node of a graph related (or paired) with itself through an empty relation.

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 an ϵ-free language M in a language L, using a special symbol † not in the alphabet Δ, 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 †. Moreover, between two occurrences of words of M that have been replaced, no word from M remains as a subword.

Example 6 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 $\mathcal{T}$ and a regular language M' such that:*

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

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:

$$\mathcal{T} = (S \cup \{s_0'\}, \Delta, \Gamma, \delta', s_0', \{s_0'\}),$$

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

$$\begin{aligned}
\delta' \;=\; & \{(s, R, s', \epsilon) : (s, R, s') \in \delta\} \cup && (1)\\
& \{(s_0', R, s_0', R) : R \in \Delta\} \cup && (2)\\
& \{(s_0', R, s, \epsilon) : (s_0, R, s) \in \delta\} \cup && (3)\\
& \{(s_0', R, s_0', \dagger) : (s_0, R, s) \in \delta \text{ and } s \in F\} \cup && (4)\\
& \{(s, R, s_0', \dagger) : (s, R, s') \in \delta \text{ and } s' \in F\}. && (5)
\end{aligned}$$

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, there will in the transducer be an additional transition $R/\dagger$ to s_0', which is also the (only) final state of $\mathcal{T}$. Observe, that if the transducer $\mathcal{T}$ 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 that $U' \in L(A)$. 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 $\mathcal{T}$ replaces arbitrarily many occurrences of words of M in w with the special symbol $\dagger$.

For an example, suppose M is $R(SR)^* + RST$. Then an automaton that accepts this language is given in Figure 15 drawn with solid arrows. The corresponding finite

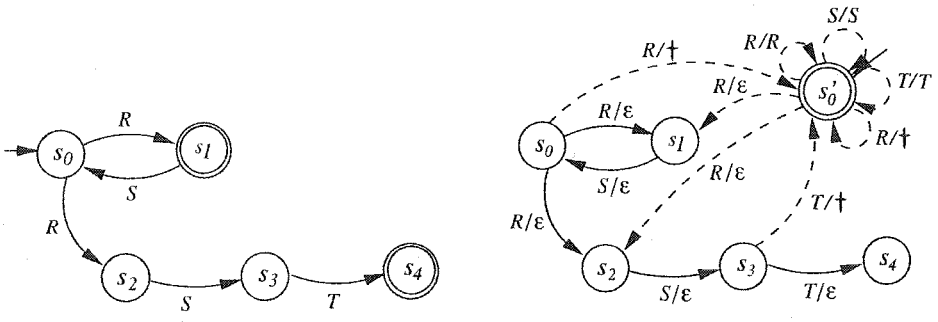

Figure 15: Construction of a replacement transducer

transducer is shown in the same figure in 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 straightforward to verify that

$$\mathcal{T}(L) = L \cup \{U_1 \dagger U_2 \dagger \ldots \dagger U_k : \text{ for some } U \text{ in } L \text{ and words } w_i \text{ in } M, \\ U = U_1 w_1 U_2 w_2 \ldots w_{k-1} U_k\}.$$

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

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

where $(.)^c$ denotes the set complement. Now M' guarantees that no other candidate for replacing occurs inside the words of the final result. ■

3.4 Exhaustive partial possibility rewriting

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 alphabet, 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 *exhaustive partial possibility rewriting* of a database query Q using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Definition 4 The *exhaustive partial possibility rewriting* (EPPR) 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 EPPR of a query Q over Δ using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions over Δ.

Theorem 11 *The* EPPR *for a query Q 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:

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

where

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

The transducer $\mathcal{T}$ 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 to the previous theorem.

From the transduction $\mathcal{T}(Q)$ 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 $Q \rhd \mathbf{V}$. 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. ■

The EPPR is a generalization of the PR. The conceptual similarity of these two rewritings can also be observed in the following way. Change the above mask to Ω^* and the result will be the PR, as opposed to the EPPR.

3.5 Exhaustive and contained partial rewriting

In this section we will be concerned with extracting the biggest contained rewriting from the EPPR.

Definition 5 The *exhaustive and contained partial rewriting* (ECPR) of a query Q on Δ is the language on $\Omega \cup \Delta$ given by

$$\{w \in (Q \rhd \mathbf{V}) : def(w) \subseteq Q\}.$$

We now present a method for computing the ECPR 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 $\mathcal{T}$ used for the EPPR. Then compute the transduction $\mathcal{T}(Q^c)$.
3. Compute the complement $(\mathcal{T}(Q^c))^c$ of the previous transduction.
4. Intersect the complement $(\mathcal{T}(Q^c))^c$ 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 above language Q' is exactly the* ECPR *of Q.*

PROOF. "$\subseteq$". $\mathcal{T}(Q^c)$ is the set of all words w on $\Omega \cup \Delta$ such that $def(w) \cap Q^c \neq \emptyset$. Hence, $(\mathcal{T}(Q^c))^c$, 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 ECPR. Furthermore, intersecting with the mask M we keep in $(\mathcal{T}(Q^c))^c$ 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 ECPR.

"$\supseteq$". We will prove this direction by a contradiction. First observe that both the ECPR and the set Q' are subsets of the EPPR. It follows that, 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 ECPR but not in Q'. Then $def(w) \subseteq Q$, and on the other hand, since $w \notin Q'$ it follows that $w \in Q'^c$, which means that $w \in \mathcal{T}(Q^c) \cup M^c$. But as we mentioned before, the word w, which belongs in the ECPR, "passes" the mask M and this implies that it cannot "pass" the complement of the mask, i.e. $w \in \mathcal{T}(Q^c)$. Thus $def(w) \cap Q^c \neq \emptyset$ that is, $def(w) \not\subseteq Q$, i.e. w cannot be in the ECPR, and this is a contradiction. ■

3.6 Query optimization using partial rewritings

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 exact, and we are interested in answering the query by consulting the views as far as possible, and by accessing the database only when necessary.

Let $\mathcal{V}$ be the view graph and DB the database. It is easy to show, that if a language Q' is the MCR for a query Q, and it is exact, then $ans(Q, DB) = ans(Q', \mathcal{V})$ [CGLV2000a]. Suppose now that Q' is instead the ECPR of a query Q. If the ECPR is exact, then we can use it for computing the full answer to the query. For this, we evaluate the ECPR trying to do the best on the view-graph, and accessing the database in a "lazy" fashion, only when necessary. Our lazy algorithm can be used even in the cases when the ECPR is not exact. In general, the algorithm can be used with any contained partial rewriting Q'. The output will be a set of object pairs that is subset of the full answer to the query. In particular, if the partial rewriting is exact, then we obtain the answer to the query in its entirety. If the rewriting is not exact, then we need to compute the answer to the difference $Q - def(Q')$ directly on the database. Notably, in order to test whether a rewriting is exact or not, we can use the optimal algorithm presented in [CGLV99].

Now, returning to the ECPR versus MCR, we note that even when the ECPR is not exact, it is more useful for optimization than the MCR. This is because the ECPR always contains the MCR. Hence, the ECPR is at least as good as the MCR, and can be better in many cases. The following example illustrates this fact. Suppose $Q = RST + U$ and $V = RS$. Then the MCR is empty, while the ECPR is $Q' = VT$, which at least always gives a subset of the desired answer.

Before describing our lazy algorithm, let us review how query answering on semistructured databases typically works [ABS99].

Algorithm 2

Input: A regular path query Q and a database DB.

Output: The answer to query Q on database DB.

Method: 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 $(b, s) \in Reach_a$. If there is a database symbol R, such that there is an edge $b \xrightarrow{R} b'$ in the database DB and there is a transition $s \xrightarrow{R} s'$ in A_Q, then add the pair (b', s') to $Reach_a$.

Finally, set

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

■

It is easy to see that the above algorithm can be interpreted as an "intersection" of the query automaton and the database graph considered as an NFA automaton, where the nodes are both initial and final states. Now, if $A_Q = (S, \Delta, \delta, s_0, F)$ is the query automaton, then a pair (a, b) is in $ans(Q, DB)$ if there is a state $s \in F$ such that (b, s) is reachable from (a, s_0) in the intersection automaton (through Cartesian product) $DB \times A_Q$.

In the following we will modify the Algorithm 2 into a lazy algorithm for evaluating any contained partial rewriting of a query Q (and in particular the ECPR), with respect to a set of exact views.

Algorithm 3

Input: A contained partial rewriting Q', a view graph $\mathcal{V}$, and a database DB.

Output: The evaluation of Q' on the view graph $\mathcal{V}$ and database DB.

Method: First construct an automaton $A_{Q'}$ for Q'. Let s_0 be the initial state in $A_{Q'}$. Then, for each node $a \in N$, we compute a set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(a, s_0)\}$.
2. For each transition $s_0 \xrightarrow{R} s$ in $A_{Q'}$, access DB and add to $\mathcal{V}$ the subgraph of DB induced by *all* the edges originating from the nodes b, which have some outgoing edges R. For each such node b, create a flag $Expanded_b$, and set it to **true**.
3. Repeat 4 until $Reach_a$ no longer changes.
4. Choose a pair $(b, s) \in Reach_a$.
   (a) If there is a transition $s \xrightarrow{v_i} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{v_i} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (b) Similarly, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, and there is an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (c) Otherwise, if there is a transition $s \xrightarrow{R} s'$ in $A_{Q'}$, but there is not an edge $b \xrightarrow{R} b'$ in $\mathcal{V}$, and there is not a flag $Expanded_b$ = **true**, then access the database and add to $\mathcal{V}$ the subgraph of DB induced by *all* the edges originating from b. Create a flag $Expanded_b$, and set it to **true**.

Set $eval(Q', \mathcal{V}, DB) = \{(a, b) : (b, s) \in Reach_a$, and s is a final state in $A_{Q'}\}$. ■

Let's recall that the set of nodes comprising the viewgraph $\mathcal{V}$ is a subset of the nodes comprising the database graph DB. As shown in [CGLV2000a], if the views are exact and Q' is the MCR that happens to be exact, then $eval(Q', \mathcal{V}, DB)$, which becomes $ans(Q', \mathcal{V})$ in the MCR case, is equal to $ans(Q, DB)$. This is because, in such a case, every word $R_1 \ldots R_n \in Q$ will have some representative word $v_1 \ldots v_m \in Q'$ such that $def(v_1 \ldots v_m) \ni R_1 \ldots R_n$. This means in turn that, if there is a path $a \xrightarrow{R_1 \ldots R_n} b$ in the database, then there will also be a path $a \xrightarrow{v_1 \ldots v_m} b$ in the viewgraph, i.e. $ans(Q, DB) \subseteq ans(Q', \mathcal{V})$. On the other hand since Q' is exact, we have that $def(v_1 \ldots v_m) \subseteq Q$ for any $v_1 \ldots v_m \in Q'$, and this means that $ans(Q', \mathcal{V}) \subseteq ans(Q, DB)$.

In the case when Q' is an exact partial rewriting, all the above reasoning remains the same except that now we need to answer Q' not simply on $\mathcal{V}$ but on the union $\mathcal{V} \cup DB$. Clearly, $ans(Q', \mathcal{V} \cup DB) = ans(Q, DB)$. As we have already mentioned, computing the answer to a query, is the same as computing the intersection through Cartesian product of the query with the database graph. So, for computing $ans(Q', \mathcal{V} \cup DB)$ in essence we have to compute $Q' \times (\mathcal{V} \cup DB) = (Q' \times \mathcal{V}) \cup (Q' \times DB)$. We use on purpose the sign "$\times$" instead "$\cap$". The intersection as languages could be empty, while the needed part of Cartesian product not. For example if $Q' = v_1 R v_2$, the viewgraph is $\mathcal{V} = \{a \xrightarrow{v_1} b, c \xrightarrow{v_2} d\}$, and there is an edge labeled by R between b and c in the database, then $Q' \times (\mathcal{V} \cup DB) = \{(a, d)\}$.

The hope in using the ECPR in query answering is that the smaller the needed portion of the second part $Q' \times DB$ is, the greater the optimization is. We compute this part in a lazy fashion in Algorithm 3. In the steps 2 and 5 we add parts of the database in the viewgraph "on demand" by the query answering algorithm. In fact in step 5 if we need to access a node in the database we add in the viewgraph *all* (not only what we need) the outgoing edges and the neighbor nodes. This was motivated by considering the database as set of HTML pages related to each other through links. Accessing a page in the web could be time consuming, but then to parse the HTML text in main memory for finding the links and the adresses where these links point is efficient. During the evaluation of a query, a page (node) could be visited many times, and if it is already fetched before from the database with all its links, then there is no need to consult the database again. For each page x that was fetched from the database during the execution of the algorithm, we set the corresponding flag $Expanded_x$ to true, meaning that now we have full local information for this page.

From all the above, we have that

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

Next, let us discuss how to utilize the EPPR 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 ECPR we might get a proper superset of the answer. Note however that, contrary to

Theorem 13, in any case the EPPR does not need to be exact.

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

PROOF. Straightforward from the exactness of the views and the fact that $Q \subseteq def(Q'')$. ■

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 viewgraph with their derivation. That is, for each pair (a, b) connected with an edge, say v_i, in the viewgraph, 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 EPPR in query answering.

Algorithm 4

Input: The EPPR Q'', a view graph $\mathcal{V}$, and a database DB.

Output: The answer to query Q.

Method:

1. Compute $eval(Q'', \mathcal{V}, DB)$ using Algorithm 3. During the execution of Algorithm 3 the viewgraph $\mathcal{V}$ is extended with new edges and nodes as described. Call the extended viewgraph $\mathcal{V}'$.

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 $verified(Q'', \mathcal{V}, DB) = eval(Q'', \mathcal{V}, DB) \cap \{(a, b) : Q \cap L(\mathcal{V}''_{ab}) \neq \emptyset\}$, where $\mathcal{V}''_{ab}$ is a non-deterministic automaton similarly defined as DB_{ab}.

■

The usefulness of the above algorithm can be expressed as the following result.

Theorem 15 *Given a query Q and a set $\mathcal{V}$ of exact views, if Q'' is the* EPPR *of Q, then by using $\mathcal{V}$, we have that $verified(Q'', \mathcal{V}, DB) = ans(Q, DB)$.*

PROOF. Easy to see from the fact that since the views are exact, the automaton $\mathcal{V}''_{ab}$ will serve as an accurate snapshot or dataguide [NUWC97] for the database sub-graph between the nodes a and b. ■

3.7 Complexity analysis

The following theorem establishes an upper bound for the problem of generating the exhaustive replacement $L \triangleright M$, 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 $\mathcal{T}$ takes polynomial time. To compute the transduction of the regular language L, $\mathcal{T}(L)$, takes again polynomial time. But at the end, in order to compute the subset of the words in $\mathcal{T}(L)$, to which no more replacement can be applied takes exponential time. This is because we intersect with a mask that is a language described by an extended regular language containing complementation.

■

In order to show that the above upper complexity bound is essentially (or almost) optimal we need the following theorem.

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.

Let's 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 the proof of the Cook-Levin Theorem (see [Sip96]) 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.

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

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

$$\#q_0 w_1 \ldots w_n \underbrace{\sqcup \ldots \sqcup}_{n^k-n} \#,$$

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

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

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

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

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 A and that of B in 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. 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* EPPR *of a query Q using a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, is essentially (or almost) optimal.*

Theorem 19 *Given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the* ECPR *of Q 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 $\mathcal{T}(Q^c)$ 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 lower bound of the ECPR we have the following.

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

PROOF. Polynomially intersect the ECPR with Ω^* and get the MCR of [CGLV99]. But, the MCR is optimally computed in doubly exponential time in [CGLV99], so our algorithm is optimal. ■

Chapter 4

Better rewritings and optimizations: Second attack

4.1 Introduction

In this chapter we will present a better rewriting that has all the desirable properties that we have mentioned before. First, let us briefly discuss again the contained rewritings in a query optimization context. As explained in Chapter 3, in such a context the partial rewritings usually provide more help in query answering. Hence, let's focus on the partial rewriting of [CGLV99] and in the ECPR presented in Chapter 3.

In [CGLV99], the problem of computing a partial rewriting is considered as an extension of the complete rewriting MCR, enriching the set of the views with new elementary one-symbol views, chosen among the database relations (or symbols). The choice of the new elementary views is done in a brute force way, using a cost criterion depending on the application. It is worth saying here that although the space of choices for the new views is exponential on the alphabet size, this exponential component is absorbed by the double-exponential complexity of the query rewriting and

hence the algorithm is essentially optimal. However, there are cases when the algorithm of [CGLV99] for computing partial rewritings gives "too much" as illustrated in our initial example of Chapter 1. It essentially contains redundant un-rewritten (sub)words from the query.

On the other hand, we presented in the previous chapter an algorithm for computing the ECPR for a query. Using that algorithm we avoided getting "too much" in the rewriting, but unfortunately, the rewriting is not guaranteed to be exact, and this fact diminishes its usability.

In this chapter we will introduce the "maximal and contained partial rewritings" (MCPR's) for regular path queries using views. We will show that they don't give "too much," that they are always exact and more useful for the optimization of the regular path queries. Then, by using the Algorithm 3 we can obtain an evaluation that equals the answer to the original query.

We also explore the use of the partial rewritings for the optimization of the wider class of conjunctive regular path queries (CRPQ's). We introduce the "conjunctive exact partial rewritings" (CEPR's) and present an algorithm for utilizing them in CRPQ evaluation.

Finally, through a complexity theoretic analysis, we prove that our algorithm for computing the "maximal and contained partial rewritings" is essentially optimal.

4.2 A taxonomy for rewritings

Let L be a language and M an ϵ-free language, both of them over some alphabet Δ. Let $\dagger$ be a symbol outside Δ and set $def(\dagger) = M$. Similarly as in Chapter 1 we use the following definitions.

1. A *partial M-rewriting* (PR) of L is a language L' over $\Delta \cup \{\dagger\}$.
2. A PR is a *contained partial M-rewriting* (CPR) of L if $def(L') \subseteq L$.

3. A CPR is said to be *exact*, if $def(L') = L$.

Also, we note here that the (complete) rewritings, i.e. the ones that use only the † symbol, are special cases of the contained partial rewritings. Hence, we will blur the distinction between them and call the partial rewritings just *rewritings*.

It is clear that there can be several rewritings of the same language L. In order to compare different rewritings, we introduce a partial order between the languages over $\Delta \cup \{\dagger\}$. With this partial order we want to capture the intuition that the more subwords on the Δ-alphabet that have been replaced by † in a rewriting, the "bigger" (and "better") the rewriting.

Let L_1 and L_2 be languages over $\Delta \cup \{\dagger\}$. We define L_1 to be M-smaller than L_2, denoted $L_1 \leq_M L_2$, if it is possible to substitute by † some (not necessarily all) occurrences of words over M occurring as subwords in L_1, and obtain L_2 as a result. Also, the same word can participate more than once in the creation of new words.

Obviously $\leq_M$ is transitive and reflexive. It is not antisymmetric, as for instance $\{\dagger RR, \dagger\dagger, RRRR\} \leq_M \{\dagger\dagger, RRRR\}$, and $\{\dagger RR, \dagger\dagger, RRRR\} \geq_M \{\dagger\dagger, RRRR\}$, when M is for example $\{RR\}$. However, if we define $L_1 \equiv_M L_2$ iff $L_1 \leq_M L_2$ and $L_2 \leq_M L_1$, we will get a partial order on the equivalence classes.

Notably, we have that if a set L' is $\leq_M$-maximal, then its equivalence class is a singleton. To show this, we first present the following theorem.

Theorem 21 *Let L' be a language on $\Delta \cup \{\dagger\}$. Then L' is $\leq_M$-maximal, if and only if, there does not exist a word $w \in L'$, such that $w = w_1 w_2 w_3$, and $w_2 \in M$.*

PROOF. "if." This direction is easy to see because, if there is no word of M that appears as a subword in any of the words of L', then it is impossible to obtain any new $\leq_M$-larger language by substituting some subword belonging to M with †.

"only if." Let's suppose that there are subwords of L', which belong to M. Then, for each word $w \in L'$ compute a word w_M by exhaustively replacing occurrences of M

words in w until nothing can be replaced anymore. If there is no subword in w that can be replaced by †, then compute w_M equal with w. Clearly, for each word w there is at least one such word, and the number of steps for computing it is bounded by the length of w. Now, consider the language $L'' = \bigcup_{w \in L'} \{w_M\}$. For the language L'' we have that (a) $L'' \neq L'$, (b) $L' \leq_M L''$, and (c) we cannot obtain any new language from L'' by substituting † in place of some subword belonging to M. We can see that from (a) and (c), L' and L'' cannot belong to the same equivalence class, and from (b), L'' is $\leq_M$-larger than L'. All the above show that L' cannot be maximal, and this is a contradiction. ■

Corollary 4 *Let L' be a language on $\Delta \cup \{\dagger\}$. If L' is $\leq_M$-maximal, then its $\equiv_M$-equivalence class is a singleton.*

PROOF. Since L' is $\leq_M$-maximal, from the above theorem we have that from L' we cannot obtain any new language by substituting † in place of some subword belonging to M. This means in turn, that the equivalence class of L' is a singleton. ■

Now, consider this partial order restricted to the set of all the contained M-rewritings of a language L. We will denote the restricted partial order with $\leq_M^L$. Obviously, we are interested in the $\leq_M^L$-maximal contained M-rewritings of L. With a similar reasoning as before, we can prove the next theorem and corollary. The main difference is that we cannot now replace just any subword which is a word in M. Naturally, a word $w = w_1 w_2 w_3$, where $w_2 \in M$ and $def(w_1 \dagger w_3) \subseteq L$, is not yet an "optimal" word, and we call w_2 a subword *eligible for replacement.*

Theorem 22 *Let L' be a contained rewriting of L on $\Delta \cup \{\dagger\}$. Then L' is $\leq_M^L$-maximal, if and only if, there does not exist a word $w \in L'$, such that $w = w_1 w_2 w_3$, where $w_2 \in M$, and $def(w_1 \dagger w_3) \subseteq L$.*

PROOF. "if." This direction is easy to see because, if there is no word of M that appears as a subword eligible for replacement in any of the words of L', then it is impossible to obtain any new ($\leq_M^L$-larger) rewriting from L'.

"only if." Let's suppose that there are subwords eligible for replacement in the words of L'. Then, for each word $w \in L'$ compute a word w_M, by exhaustively replacing the eligible for replacement subwords in w, until nothing can be replaced anymore. If there is no eligible for replacement subword in w, then compute w_M equal with w. Clearly, for each word w there is at least one such word w_M, and the number of steps for computing it, is bounded on the length of w. Now, consider the rewriting $L'' = \bigcup_{w \in L'} \{w_M\}$. For the rewriting L'' we have that (a) $L'' \neq L'$, (b) $L' \leq_M^L L''$, and (c) we cannot obtain any new $\leq_M^L$-larger rewriting from L''. We can see that from (a) and (c), L' and L'' cannot belong to the same equivalence class, and from (b), L'' is $\leq_M^L$-larger than L'. All the above show that L' cannot be a $\leq_M^L$-maximal rewriting, and this is a contradiction. ■

Corollary 5 *Let L' be a contained rewriting on $\Delta \cup \{\dagger\}$. If L' is $\leq_M^L$-maximal, then its $\equiv_M^L$-equivalence class is a singleton.*

PROOF. Since L' is $\leq_M^L$-maximal, from the above theorem we have that from L' we cannot obtain any new $\leq_M^L$-larger rewriting. This means in turn that the equivalence class of L' is a singleton. ■

Also, we require exactness for a rewriting. If such, the rewriting is more useful for query optimization [CGLV2000a, GT2001a].

A contained rewriting that is both $\leq_M^L$-maximal and exact is the union of all $\leq_M^L$-maximal M-rewritings of L. We call this rewriting the *maximal and contained partial M-rewriting* of L, and denote it with $\text{MCPR}_M(L)$.

From all the above, we can see that the $\text{MCPR}_M(L)$ is the set of *all* the words on $\Delta \cup \{\dagger\}$ with no subword for potential "contained replacement."

Example 7 Let $M = \{RSR, S\}$, and

$$L = \{RSRS, SRSR, RSRRSR, SS, SRSRS, SSS\}.$$

Then

$$\text{MCPR}_M(L) = \{\dagger\dagger, S\dagger S\}.$$

Here, †† is obtained from $RSRS$, $SRSR$, $RSRRSR$, or SS. The word $S\dagger S$ is obtained from $SRSRS$ or SSS. There are no more eligible subwords for replacement. For instance, replacing subwords in $\underline{S}\dagger\underline{S}$ that belong to M, would violate the containment condition necessary for being a rewriting.

Formally, we have that:

Theorem 23 *The* $\text{MCPR}_M(L)$ *is* $\leq_M^L$*-maximal and exact.*

PROOF. The $\leq_M^L$-maximality follows from the fact that the $\text{MCPR}_M(L)$ is the union of the sets of words w with no subwords eligible for replacement. For the exactness, observe that by the definition we have $def(\text{MCPR}_M(L)) \subseteq L$. On the other hand, consider a word $w \in L$. Testing a finite number of times (function of the length of w) we can find a word $w' \in (\Delta \cup \{\dagger\})^*$ with the smallest possible number of Δ symbols, and such that $w \in def(w') \subseteq L$. Clearly, $w' \in \text{MCPR}_M(L)$. ■

We can give a natural generalization of the definition of the MCPR for the case when we like to 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, suppose we are given the ϵ-free view languages $\mathbf{V} = \{V_1, \ldots, V_n\}$ and a target query language Q, all of them over the alphabet Δ. Also, consider the view alphabet $\Omega = \{v_1, \ldots, v_n\}$, for which as defined in Chapter 1, we have $def(v_i) = V_i$, for $i \in [1, n]$. As defined there, a *contained partial* $\mathbf{V}$*-rewriting* of Q is a language Q' over $\Delta \cup \Omega$, such that $def(Q') \subseteq Q$. The partial rewriting is said to be *exact* if $def(Q') = Q$. Then, in order to compare the rewriting we define analogously the partial order $\leq_{\mathbf{V}}$ on $(\Delta \cup \Omega)^*$ as follows.

Let Q_1 and Q_2 be languages over $\Delta \cup \Omega$. We define Q_1 to be $\mathbf{V}$-smaller than Q_2, denoted $Q_1 \leq_{\mathbf{V}} Q_2$, if it is possible to substitute by $v_{i_1}, \ldots, v_{i_k}$ some (not necessarily all) occurrences of words over $V_{i_1}, \ldots, V_{i_k}$ respectively, occurring as subwords in Q_1, and obtain Q_2 as a result.

As before, we restrict this partial order to the set of all the contained $\mathbf{V}$-rewritings of a language Q. Similarly, we denote the restricted partial order with $\leq_{\mathbf{V}}^Q$. Finally,

we define the *maximal and contained partial* $\mathbf{V}$*-rewriting* of Q, denoted $\text{MCPR}_{\mathbf{V}}(Q)$, to be the *union* of all $\leq_{\mathbf{V}}^{Q}$-maximal $\mathbf{V}$-rewritings Q' of Q. Clearly, as in Theorem 23, we can show that the $\text{MCPR}_{\mathbf{V}}(Q)$ is $\leq_{\mathbf{V}}^{Q}$-maximal and exact.

In the following, we will compare the MCPR with the partial rewritings presented so far. We begin with the partial rewriting of [CGLV99]. In that paper, candidate sub-alphabets $\Delta' \subseteq \Delta$ are selected using some cost criteria, and then the symbols of Δ' are considered as new elementary one-symbol views. Each time, using the algorithm presented in [CGLV99], a rewriting is computed and after that its exactness is tested. However, the algorithm used, although very suitable for computing all the rewritings in the view alphabet Ω, can compute "non-optimal" $\Delta \cup \Omega$ words when it is used for computing partial rewritings. In general, the partial rewriting of [CGLV99], although an exact one, is not always $\leq_{\mathbf{V}}^{Q}$-maximal. We call the partial rewriting of [CGLV99] the *greatest contained partial rewriting* or GCPR because it contains *all* the words w over $\Delta' \cup \Omega$, such that $def(w) \subseteq Q$. Observe here that the maximality in this rewriting is with respect to the containment of $\Delta' \cup \Omega$ words and not really to their optimality regarding the non-view symbols.

Now, let's consider the ECPR presented in Chapter 3. This rewriting can also be seen as the union of all $\leq_{\mathbf{V}}$-maximal and contained $\mathbf{V}$-rewritings of Q. By Theorem 21 we have that this is 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 *def* is contained in the query Q. Clearly, for computing ECPR, we replace subwords of Q with view symbols thinking only about the optimality of words with regard to the non-view symbols, but not caring about the exactness of the rewriting. Since any $\leq_{\mathbf{V}}$-maximal and contained rewriting is also $\leq_{\mathbf{V}}^{Q}$-maximal, the result is that we get always a $\leq_{\mathbf{V}}^{Q}$-maximal rewriting, but its exactness is not guaranteed.

Finally, the MCPR is a rewriting that satisfies both the optimality with regard to the non-view symbols and the exactness. Summarizing, and also including the maximally contained (complete) rewriting MCR in the comparison, we have the following table.

	$\leq_{\mathbf{V}}^{Q}$-maximality	Exactness
MCR [CGLV99]	YES	NO
GCPR [CGLV99]	NO	YES
ECPR [GT2001a]	YES	NO
MCPR [GT2003a]	YES	YES

Table 1: Maximality and exactness of the contained rewritings

4.3 Computing the maximal and contained partial rewriting

To this end, we will first give a characterization of the maximal and contained partial M-rewriting of a language L. The construction in the proof of our characterization provides the basic algorithm for computing the $\text{MCPR}_M(L)$ on a given regular language L.

Theorem 24 *Let L and M be regular languages over an alphabet Δ. There exists a finite transducer $\mathcal{T}$ and a regular language M', such that*

$$\text{MCPR}_M(L) = (\mathcal{T}(L^c))^c \cap M',$$

where $(.)^c$ denotes set complement.

PROOF. Let $A = (S, \Delta, \delta, s_0, F)$ be a nondeterministic finite automaton that accepts the language M. Similarly as in Thoerem 10, we construct the finite transducer:

$$\mathcal{T} = (S \cup \{s_0'\}, \Delta, \Delta \cup \{\dagger\}, \delta', s_0', \{s_0'\}),$$

where δ', written as a relation, is

$$\begin{aligned}
\delta' \;=\; & \{(s, R, s', \epsilon) : (s, R, s') \in \delta\} \cup \\
& \{(s_0', R, s_0', R) : R \in \Delta\} \cup \\
& \{(s_0', R, s, \epsilon) : (s_0, R, s) \in \delta\} \cup \\
& \{(s_0', R, s_0', \dagger) : (s_0, R, s) \in \delta \text{ and } s \in F\} \cup \\
& \{(s, R, s_0', \dagger) : (s, R, s') \in \delta \text{ and } s' \in F\}.
\end{aligned}$$

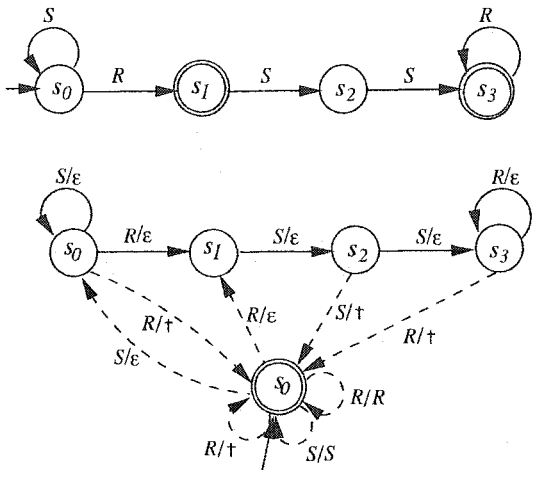

Figure 16: Construction of a replacement transducer

Given a word of $w \in \Delta^*$ as input, the finite transducer $\mathcal{T}$ replaces arbitrarily many occurrences of words of M in w with the special symbol †. For another example [1], suppose M is given by the automaton in Figure 16, top. The corresponding finite transducer is shown at the bottom of the same figure. It consists of the automaton for M, whose transitions now produce as output ϵ, plus the state s_0', and the additional transitions, which are drawn with dashed arrows.

As in Theorem 10, if L' is a language on Δ, it is straightforward to verify that

$$\begin{aligned}\mathcal{T}(L') \;= L' \cup\; \{&u_1 \dagger u_2 \dagger \ldots \dagger u_k : \\ &\text{for some } u \in L' \text{ and words } w_i \in M, \\ &u = u_1 w_1 u_2 w_2 \ldots w_{k-1} u_k\}.\end{aligned}$$

Recall that, we have $def(\dagger) = M$, and $def(R) = R$ for each $R \in \Delta$. From the above, we can also easily characterize the set $\mathcal{T}(L')$ from another point of view. Namely, $\mathcal{T}(L')$ is the set of all words w on $\Delta \cup \{\dagger\}$, such that $def(w) \cap L' \neq \emptyset$.

Now, let's consider the transduction $\mathcal{T}(L^c)$. As characterized above, $\mathcal{T}(L^c)$ will be the set of all words w on $\Delta \cup \{\dagger\}$ such that $def(w) \cap L^c \neq \emptyset$. Hence, $\mathcal{T}(L^c))^c$, being the complement of this set, will contain *all* the $\Delta \cup \{\dagger\}$ words, such that *all* the

[1] One example is given in Theorem 10

Δ-words in their substitution by *def* will be contained in L. This is the containment condition for a word on $\Delta \cup \{\dagger\}$ to be in a CPR. Clearly, $\mathcal{T}(L^c))^c$ is a CPR, and namely, it is the union of *all* the (M-) CPR's of L. Hence, $\text{MCPR}_M(L) \subseteq \mathcal{T}(L^c))^c$.

However, in order to compute $MPR_M(L)$, what we like is not a contained arbitrary, but a contained exhaustive replacement of words from M. To achieve this goal, we should filter out from the set $T(L^c)^c$, the words having eligible subwords for replacements that do not violate the containment constraint. Formally speaking, we are interested in filtering out the non $\leq_M^L$-maximal words from $T(L^c)^c$.

For this, consider another special symbol $\dagger'$, such that $\dagger' \notin \Delta \cup \{\dagger\}$, and the substitution $def' : \Delta \cup \{\dagger\} \cup \{\dagger'\} \longrightarrow \Delta \cup \{\dagger\}$, such that

$$def'(\dagger') = M, \;\; def'(\dagger) = \dagger, \text{ and } \; def'(R) = R \text{ for } R \in \Delta.$$

Now, we build a transducer $\mathcal{T}'$ in the same way as we did for the transducer $\mathcal{T}$, but for the extended alphabet $\Delta \cup \{\dagger\}$ and considering $\dagger'$ as the special symbol. Reasoning similarly as before, if we transduce the complement of $(\mathcal{T}(L^c))^c$ i.e. $\mathcal{T}(L^c)$, we have that $(\mathcal{T}'(\mathcal{T}(L^c)))^c$ is the set of *all* words $w \in \Delta \cup \{\dagger\} \cup \{\dagger'\}$, such that $def'(w) \subseteq (\mathcal{T}(L^c))^c$. Let $L_{\dagger'}$ be the subset of $(T'(T(L^c)))^c$ such that the words in $L_{\dagger'}$ have at least one $\dagger'$ symbol. Formally,

$$L_{\dagger'} = (T'(T(L^c)))^c \cap (\Delta \cup \{\dagger, \dagger'\})^* \dagger' (\Delta \cup \{\dagger, \dagger'\})^*.$$

We show the following lemma.

Lemma 1 *$def'(L_{\dagger'})$ is the set of all (contained) non $\leq_M^L$-maximal words on $\Delta \cup \{\dagger\}$.*

Proof. Let $w \in L_{\dagger'}$. Then, $w = w_1 \dagger' w_2$ and since $w \in (T'(T(L^c)))^c$ we have that $def'(w) \subseteq ((T(L^c)))^c$ that is $def'(w_1) M\, def'(w_2) \subseteq ((T(L^c)))^c$, which means that $def(def'(w_1) M\, def'(w_2)) \subseteq L$. Hence, the set $def'(w) = def'(w_1) M\, def'(w_2)$ contains only (contained) non $\leq_M^L$-maximal words.

On the other hand, let w be an arbitrary (contained) non $\leq_M^L$-maximal word on $\Delta \cup \{\dagger\}$. We can write $w = w_1 w_2 w_3$, where $w_2 \in M$ and $def(w_1) M\, def(w_3) \subseteq L$. Also, we have that $def(w_1) M\, def(w_3) = def(w_1 M w_3)$. Since $(T(L^c))^c$ is the set of all

the words u on $\Delta \cup \{\dagger\}$ such that $def(u) \subseteq L$, we have that $w_1 M w_3 \subseteq (T(L^c))^c$. This tells us that $w_1 \dagger' w_3 \in (T'(T(L^c)))^c$. Also, $w_1 \dagger' w_3 \in (\Delta \cup \{\dagger, \dagger'\})^* \dagger' (\Delta \cup \{\dagger, \dagger'\})^*)$. Hence, for the arbitrary non $\leq_M^L$-maximal word w, we found a word $w' = w_1 \dagger' w_3$ in $L_{\dagger'}$, such that $w \in def'(w')$. ■ Lemma 1

From the above lemma it becomes clear that

$$MPR_M(L) = (T(L^c))^c \cap (def'(L_{\dagger'}))^c.$$

■

As a generalization of Theorem 24, we can give the following result about the maximal partial $\mathbf{V}$-rewriting, of a query Q with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Theorem 25 *Given a regular path query Q, the* $\text{MCPR}_{\mathbf{V}}(Q)$ *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:

$$\mathcal{T} = (S_1 \cup \ldots \cup S_n \cup \{s_0'\}, \Delta, \Delta \cup \Omega, \delta', s_0', \{s_0'\}),$$

where

$$\begin{aligned}
\delta' \;=\; & \{(s, R, s', \epsilon) : (s, R, s') \in \delta_i,\ i \in [1, n]\} \cup \\
& \{(s_0', R, s_0', R) : R \in \Delta\} \cup \\
& \{(s_0', R, s, \epsilon) : (s_{0i}, R, s) \in \delta_i,\ i \in [1, n]\} \cup \\
& \{(s_0', R, s_0', v_i) : (s_{0i}, R, s) \in \delta_i \text{ and } s \in F_i,\ i \in [1, n]\} \cup \\
& \{(s, R, s_0', v_i) : (s, R, s') \in \delta_i \text{ and } s' \in F_i,\ i \in [1, n]\}.
\end{aligned}$$

The transducer $\mathcal{T}$ performs the following task: given a language L' as input, it produces as output all the words of L', having replaced in them an arbitrary number of words from $V_1 \cup \ldots \cup V_n$ with the corresponding special symbols.

Let's now consider the transduction $\mathcal{T}(Q^c)$. As in the proof of Theorem 24, we can also see the transduction $\mathcal{T}(Q^c)$ as the set of all words w on $\Delta \cup \Omega$ such that

$def(w) \cap Q^c \neq \emptyset$. Hence, $(\mathcal{T}(Q^c))^c$, being the complement of this set, will contain only $\Delta \cup \Omega$ words, such that *all* the Δ-words in their substitution by *def* will be contained in Q. This is the containment condition for a word on $\Delta \cup \Omega$ to be in the $\text{MCPR}_{\mathbf{V}}(Q)$.

However, what we like is not a contained arbitrary, but a contained exhaustive replacement of words from $V_1 \cup \ldots \cup V_n$. To achieve this goal, we filter out from $\mathcal{T}(Q^c)^c$ the words that still have subwords for potential "contained replacements." Formally speaking, we are interested in filtering out the non $\leq^Q_{\mathbf{V}}$-maximal words from $T(Q^c)^c$.

For this, we consider the alphabet $\Omega' = \{v'_1, \ldots, v'_n\}$, such that $\Omega' \cap (\Delta \cup \Omega) = \emptyset$. We define the substitution $def' : \Delta \cup \Omega \cup \Omega' \longrightarrow \Delta \cup \Omega$, such that

$$def'(v'_i) = V_i, \;\; def'(v_i) = v_i, \text{ for } i \in [1, n], \text{ and } def'(R) = R \text{ for } R \in \Delta.$$

Now, we build a transducer $\mathcal{T}'$ in the same way as we did for the transducer $\mathcal{T}$, but for the extended alphabet $\Delta \cup \Omega$ and considering Ω' as the set of special symbols. Reasoning similarly as before, if we transduce the complement of $(\mathcal{T}(Q^c))^c$ i.e. $\mathcal{T}(Q^c)$, we have that $(\mathcal{T}'(\mathcal{T}(Q^c)))^c$ is the set of *all* words $w \in \Delta \cup \Omega \cup \Omega'$, such that $def'(w) \subseteq (\mathcal{T}(Q^c))^c$. Let $Q_{\Omega'}$ be the subset of $(T'(T(Q^c)))^c$ such that the words in $Q_{\Omega'}$ have at least one Ω' symbol. Formally,

$$Q_{\Omega'} = (T'(T(Q^c)))^c \cap (\Delta \cup \Omega \cup \Omega')^* \Omega' (\Delta \cup \Omega \cup \Omega')^*$$

As for the Theorem 24 we can show the following lemma.

Lemma 2 *$def'(Q_{\Omega'})$ is the set of all (contained) non $\leq^Q_{\mathbf{V}}$-maximal words on $\Delta \cup \Omega$.*

From the above lemma it becomes clear that

$$MPR_V(Q) = (\mathcal{T}(Q^c))^c \cap (def'(Q_{\Omega'}))^c.$$

■

4.4 Optimizing conjunctive regular path queries

The MCPR can be used as the input rewriting for the Algorithm 3, and in a lazy fashion we can compute the answer to the original query. Moreover, since the MCPR is an exact rewriting, it is all we need for computing the whole answer. In this section we will try for something more advanced. Our aim is to optimize the evaluation of the conjunctive regular path queries by using exact partial rewritings and in particular the MCPR.

A *conjunctive regular path query* (CRPQ) Q is an expression of the form

$$Q(x_1, \ldots, x_l) : -y_1 E_1 z_1, \ldots, y_k E_k z_k,$$

where $x_1, \ldots, x_l$, $y_1, \ldots, y_k$, $z_1, \ldots, z_k$ are (not necessarily all distinct) variables over the universe of objects D, such that all the distinguished (head) variables $x_1, \ldots, x_l$ occur in the body, i. e. each x_i is some y or z, and $E_1, \ldots, E_k$ are regular languages (or regular path queries, RPQ) over the database alphabet Δ. We call the conjunctions yEz of a CRPQ, *regular path atoms*, or simply *atoms*.

The answer set $ans(Q, DB)$ to a CRPQ Q over a database $DB = (N, E)$ is the set of tuples $(a_1, \ldots, a_l)$ of nodes of DB, such that there is a total mapping τ from $y_1, \ldots, y_k, z_1, \ldots, z_k$ to N with $\tau(x_i) = a_i$ for every distinguished variable x_i of Q, and $(\tau(y), \tau(z)) \in ans(E, DB)$ for every atom yEz in Q.

We say for an atom yEz, when $\tau(y) = a$ and $\tau(z) = b$, that y and z have been *bound* to the objects a and b respectively.

Suppose now, that we have available a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of conjunctive regular path views, which are defined as

$$V_i(x_{i1}, \ldots, x_{il_i}) : -y_{i1} E_{i1} z_{i1}, \ldots, y_{ik_i} E_{ik_i} z_{ik_i},$$

for $i \in [1, n]$. Let $\mathbf{E}$ be the set of all E_{ij} above, for $i \in [1, n]$, and Ω be a set of $e_{ij} \notin \Delta$ corresponding symbols. Then, we define the *conjunctive exact partial* $\mathbf{V}$*-rewriting*, or $\text{CEPR}_{\mathbf{V}}(Q)$, of a CRPQ Q with respect to a set $\mathbf{V}$ of conjunctive regular path view definitions, as

$$\text{CEPR}_{\mathbf{V}}(x_1, \ldots, x_l) : -y_1 E'_1 z_1, \ldots, y_k E'_k z_k,$$

where E'_i, for $i \in [1,k]$, is an *exact* partial rewriting of the regular language E_i with respect to $\mathbf{E}$, with Ω as the corresponding set of special symbols.

Suppose that the sub-mechanism for answering the regular path atoms of the CRPQ's remembers the regular expressions and caches the corresponding answer sets of the regular path atoms in the recently processed CRPQ's. Observe that the cached answer sets of the regular path atoms E_{ij} do not necessarily contain the full set $ans(E_{ij}, DB)$, but only those pairs that were called for, by the sideways information passing mechanism used. To formalize the "fullness" of a cached answer set, we introduce the notions of the "global completeness" and "local completeness," for the subsets of $ans(E, DB)$, for some E.

Let P be a subset of $ans(E, DB)$, for some E and DB. Then P is said to be *globally complete* if $P = ans(E, DB)$. On the other hand, P is *locally complete with respect to a node* a, if $P \supseteq \sigma_{\$1=a}(ans(E, DB))$. If P is locally complete wrt all nodes in $\pi_{\$1}(P)$, we say that P is *locally complete* [2].

We observe that, if the query processor can traverse the database graph only in the forward direction (as is the case in the web, see [AV99]), then the cached answer set of any atom $yE_{ij}z$, for which z is not already bound to some objects, is locally complete. For simplicity, we call the atoms *locally complete*, when we have computed for them locally complete answer sets. As we will see, the locally complete atoms can be very useful in the query optimization.

Now, let's consider the atoms, which have the z variable already bound to some objects. We observe that, for such atoms, we could have computed locally incomplete answer sets. In order to see this, suppose for example, that the variable z has been bound to the objects b and c. Then, if the variable y bounds to an object, say a, the RPQ answering sub-mechanism using "cut"-like constructs, can stop the evaluation after computing in the answer, starting to navigate from a, the objects b and c. But, the object a could be connected with paths spelling words in the same regular language, to other database objects besides b and c.

However, if the variable z has been also bound to another object, say d, which

[2] We consider for the simplicity of notations a set of object pairs as a binary relational table.

cannot be reached from a, by following a path spelling a word in E_{ij}, then we inevitably have to compute a locally complete answer set for the (sub)atom $aE_{ij}z$, in order to reject the pair (a, d). I such a case, the "cut"-like constructs do not have have a chance to execute. So, $aE_{ij}z$ is locally complete, and we cache its answer set for future optimizations.

Now, let's see how we can optimize the answering of the conjunctive regular path queries using conjunctive exact partial rewritings.

Consider the regular path atom $yE_{ij}z$ in some recently processed view V_i. As explained before, if the variable z is not bound to some database object, then $yE_{ij}z$ is locally complete. Otherwise, we consider all the (sub)atoms $aE_{ij}z$, which are locally complete. Let's denote with $cache_{ij}$, the cached answer set of the locally complete atom $yE_{ij}z$, or if otherwise, the union of the cached answer sets of its locally complete (sub)atoms. Then, we define the *atom-view-graph* $\mathcal{V}$ to be a database over (D, Ω) induced by the set

$$\bigcup_{i \in \{1,\ldots,n\}} \{(a, e_{ij}, b) : (a, b) \in cache_{ij}\}.$$

of Ω-labeled edges. [3]

The evaluation will alternate between the database-graph and atom-view-graph. Suppose that we want to answer the atom yEz and y has already been bound to a node, say a. We now need to compute the nodes reachable from a, by a path spelling a word in E. Recall that for E we have computed a corresponding exact rewriting E'. We start by answering $aE'z$ on DB. Intuitively, when during the navigation we are in a node b, and the regular expression for E' requires an Ω-symbol, say e_{ij}, to be matched, then we look at the atom-view-graph $\mathcal{V}$. If there is a node b in $\mathcal{V}$, and in that node we find outgoing e_{ij} edges, then we advance the navigation in the atom-view-graph. Since the cached answer for E_{ij} is locally complete, we are sure that we get all the answers had we answered E_{ij} directly in the database starting from b.

On the other hand, if we do not find an object b in $\mathcal{V}$, or even if we find b in $\mathcal{V}$, but

[3] From another point of view, we can also see that a regular path atom (or (sub)atom), along with its locally complete answer set, is nothing else but a local node constraint as defined in [AV99].

there are no outgoing edges labeled with e_{ij}, then we evaluate E_{ij}, starting from b in the database graph, and enrich the atom-view-graph accordingly. Clearly, we do not add by this operation any overhead for answering parts of some new Δ^* word, that is not in E, because the rewriting is exact. Formally, we have the following algorithm for answering aEz on the basis of E'.

Algorithm 5

Input: An exact rewriting E' for the (sub)atom aEz, and a database DB.

Output: The answer to the (sub)atom aEz on database DB.

Method: First construct an automaton $A_{E'}$ for E'. Let s_0 be the initial state in $A_{E'}$. We compute the 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 $(b, s) \in Reach_a$.
   (a) If there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, then add the pair (b', s') to $Reach_a$.
   (b) Otherwise, if there is a transition $s \xrightarrow{e_{ij}} s'$ in $A_{E'}$, but there is not an edge $b \xrightarrow{e_{ij}} b'$ in $\mathcal{V}$, answer $bE_{ij}z$ on DB and enrich $\mathcal{V}$ accordingly.
   (c) If there is a transition $s \xrightarrow{R} s'$ in $A_{E'}$, and there is an edge $b \xrightarrow{R} b'$ in the database DB, then add the pair (b', s') to $Reach_a$.

Finally, set $eval(E', a, DB) = \{(a, b) : (b, s) \in Reach_a,$ and s is a final state in $A\}$. ■

It is easy to see that the following theorem is true.

Theorem 26 *Given a CRPQ Q and a set $\mathbf{V}$ of views as above, using the rewriting $\text{CEPR}_{\mathbf{V}}(Q)$, we have that for some regular atom E in Q, the answer to aEz equals $eval(E', a, DB)$.*

Now, for the case of regular path atoms aEz, where the variable z has already been bound to some objects, we slightly modify the above algorithm to evaluate the answer to such atoms as well. For this, let B be the set of objects where z has been bound. Then, in the above algorithm we *incrementally* compute in each step the set $eval(E', a, DB)$ and change the loop (2) to

Repeat 3 until $Reach_a$ no longer changes or $\pi_{\$2}(eval(E', a, DB))$ equals B.

We note that, the second condition of the loop termination is there for restricting the search space, in case we find (or verify) that we can reach from a *all* the objects in B by following paths, which spell words in E. Clearly, in this case, the answer set of the atom aEz equals $a \times B$. On the other hand, we also stop if $Reach_a$ reaches a fixed point, and in this case the answer to the atom aEz equals $a \times (\pi_{\$2}(eval(E', a, DB)) \cap B)$. In such a case, although the variable z had already been bound, the set $eval(E', a, DB))$ is fully computed and so, it is locally complete.

Finally, if we set $B = \emptyset$, when the variable z has not been already bound to some objects, then we can have a single *RPQ-answer-and-cache* algorithm. This algorithm would compute the answer set to an atom yEz, by iterating over all the objects a, where the variable y could be bound, and compute aEz by using an exact partial rewriting $aE'z$, as described above. At the end, we check whether the set $Reach_a$ has reached a fixed point. If yes, then we cache the locally complete $eval(E', a, DB))$, for future RPQ optimizations.

4.5 Complexity analysis

Theorem 27 *Given a language L and an ϵ-free language M both of them over an alphabet Δ, the problem of generating the* $\text{MCPR}_M(L)$ *is in 3EXPTIME.*

PROOF. Let us refer to the constructive proof of the Theorem 24. From there we have that

$$MPR_V(L) = (\mathcal{T}(L^c))^c \cap (def'(L_{\dagger'}))^c.$$

Consider the first term of the intersection. To compute the complement Q^c of the query is exponential. To transduce it to $\mathcal{T}(Q^c)$ is polynomial. To complement again is exponential. So, we can compute the first term in 2EXPTIME.

Now, let's consider the second term. We have that

$$L_{\dagger'} = (T'(T(L^c)))^c \cap (\Delta \cup \{\dagger, \dagger'\})^* \dagger' (\Delta \cup \{\dagger, \dagger'\})^*.$$

Clearly, to compute $L_{\dagger'}$ is in 2EXPTIME and to compute $def'(L_{\dagger'})$ is polynomial. To complement again is exponential. So, we can compute $(def'(L_{\dagger'}))^c$ in 3EXPTIME.

Totally, we have 2EXPTIME + 3EXPTIME = 3EXPTIME. ■

From the above theorem, we can easily derive the following corollary, regarding the upper complexity bound for the generation of the $\text{MCPR}_{\mathbf{V}}(Q)$ of a query Q, with respect to a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions.

Corollary 6 *Given a language Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of generating the $\text{MCPR}_{\mathbf{V}}(Q)$ is in 3EXPTIME.*

We emphasize here that the above complexity analysis is a worst case analysis. In fact, the above complexity comes from possible DFA's state "blow up." However, despite these worst case possibilities, experimental results in [GW97], for converting graphs into DFA's, are encouraging, indicating that for the majority of the cases, the running time is very reasonable and the resulting DFA's are significantly smaller than their sources. Also, observe that if the probability of getting an exponential size DFA for an NFA is $p << 1$, then the probability of getting a triple exponential "blow up" in our case would roughly be p^3, which is very small.

Now, in order to prove that the above established upper bound is essentially optimal we will need the following constructions and theorems. Consider the set

$$(\mathcal{T}(Q^c))^c \cap ((\Delta \cup \Omega)^* (V_1 \cup \ldots \cup V_n)(\Delta \cup \Omega)^*)^c,$$

where the transducer $\mathcal{T}$ is defined as in Theorem 25. This is the *exhaustive contained partial rewriting* (ECPR), presented in Chapter 3. As discussed before, it is the set of

all "mixed" words w on the alphabet $\Delta \cup \Omega$, with no subword in $V_1 \cup \cdots \cup V_n$, such that their substitution by *def* is contained in the query Q. Obviously, $def(\text{ECPR}_{\mathbf{V}}(Q)) \subseteq Q$ and we are interested in the complexity of testing its exactness with respect to the query Q. We prove the following theorem regarding the upper bound of the above exactness problem.

Theorem 28 *Given a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions, the problem of testing $def(\text{ECPR}_{\mathbf{V}}(Q)) = Q$ is in 2EXPSPACE.*

PROOF. By Theorem 27 the automaton $(\mathcal{T}(Q^c))^c$ can exponentially "blow up" and namely be of doubly exponential size. The automaton for

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

can also exponentially "blow up" but its size can be up to single exponential. So, in total the automaton for $\text{ECPR}_{\mathbf{V}}(Q)$ can be of up to doubly exponential size. Now, let's consider $def(\text{ECPR}_{\mathbf{V}}(Q))$. For the substitution *def* there exists a polynomial size transducer $\mathcal{T}_{def}$ such that $def(\text{ECPR}_{\mathbf{V}}(Q)) = \mathcal{T}_{def}(\text{ECPR}_{\mathbf{V}}(Q))$ [Yu97, GT2000]. Thus, the size of the automaton $\mathcal{T}_{def}(\text{ECPR}_{\mathbf{V}}(Q))$ will be polynomial on the size of the automaton for $\text{ECPR}_{\mathbf{V}}(Q)$ i.e. it can be doubly exponential on the size of the automaton for Q. Now, to test the exactness $def(\text{ECPR}_{\mathbf{V}}(Q)) = Q$ is in PSPACE with regard to the size of the automaton for $\text{ECPR}_{\mathbf{V}}(Q)$. So, in total we have that testing the exactness of $\text{ECPR}_{\mathbf{V}}(Q)$ is in 2EXPSPACE. ■

For the lower bound of the exactness $def(\text{ECPR}_{\mathbf{V}}(Q)) = Q$ we will use the maximally contained rewriting, $\text{MCR}_{\mathbf{V}}(Q)$ of [CGLV99]. As we know, the $\text{MCR}_{\mathbf{V}}(Q)$ is the set of *all* words w in Ω^*, such that $def(w) \in Q$. In [CGLV99], the exactness problem for the $\text{MCR}_{\mathbf{V}}(Q)$ is proven to be 2EXPSPACE complete. We will give in the following a reduction of the exactness problem for $\text{MCR}_{\mathbf{V}}(Q)$ to the exactness problem for $\text{ECPR}_{\mathbf{V}}(Q)$.

Consider a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions. Now, consider the set $\mathbf{V}' = \{V_1', \ldots, V_n'\}$ of new view definitions, where $V_i' = \$V_i\$$ for $i = [1, n]$ and $\$ \notin \Delta \cup \Omega$. Regarding the query Q, we will transform it into a new query Q' through a transduction. For this we construct the following transducer.

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: $\mathcal{T}_{\$\$} = (S_1 \cup \ldots \cup S_n \cup \{s'_0\}, \Delta, \Delta \cup \{\$\}, \delta', s'_0, \{s'_0\})$, where

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

The transducer $\mathcal{T}_{\$\$}$ performs the following task: given a language L as input, it produces as output all the words of L having inserted in them the symbol \$ to mark the beginning and the end of a subword that belongs to V_i for some $i \in [1,n]$. Moreover, only the words of L, which we can divide into a sequence of subwords, such that each belongs to some view language, are transduced. All the other L words are filtered out. Formally,

$$\begin{aligned}\mathcal{T}_{\$\$}(L) &= \{\$u_1\$\$u_2\$\ldots\$u_k\$: \\ &\quad \text{for some } u \text{ in } L, u = u_1u_2\ldots u_k \\ &\quad \text{and } \forall i \in [1,k]\ \exists j \in [1,n] \text{ such that } u_i \in V_j\}.\end{aligned}$$

Theorem 29 *Consider a query Q and a set $\mathbf{V} = \{V_1, \ldots, V_n\}$ of view definitions. Construct Q and $\mathbf{V}' = \{V'_1, \ldots, V'_n\}$ as above. Let $\Omega' = \{v'_1, \ldots, v'_n\}$ and def be the usual view substitution on Ω', i.e. $def(v'_i) = V'_i$, for $i \in [1,n]$. Also, let $def_{\$\to\epsilon}$ be the substitution that erases the symbol \$ from some language on $\Delta \cup \{\$\}$. Then, the following is true: the $\text{MCR}_{\mathbf{V}}(Q)$ is exact if and only if*

1. *$def_{\$\to\epsilon}(Q') = Q$, and*
2. *$\text{ECPR}_{\mathbf{V}'}(Q')$, with respect to $\mathbf{V}' = \{V'_1, \ldots, V'_n\}$, is exact.*

PROOF. "if." The fact that $def_{\$\to\epsilon}(Q') = Q$ means that for *any* word $w \in Q$ there exists a representation $w = u_1u_2\ldots u_k$ such that $\forall i \in [1,k]\ \exists j \in [1,n]$ for which we have $u_i \in V_j$. In other words, for each $w \in Q$ there exists a corresponding $w' \in Q'$ with inserted markers \$. Now, observe that, because of the markers \$, the $\text{ECPR}_{\mathbf{V}'}(Q')$ is forced to be equal with the $\text{MCR}_{\mathbf{V}'}(Q')$. So, the exactness of $\text{ECPR}_{\mathbf{V}'}(Q')$ means

exactness of $\text{MCR}_{\mathbf{V}'}(Q')$, which in turn means that for each word $w' \in Q'$ there is a word $v'_{i_1} \dots v'_{i_m}$ such that $w' \in def(v'_{i_1} \dots v'_{i_m})$, and $def(v'_{i_1} \dots v'_{i_m}) \subseteq Q'$. It is easy to see that for the corresponding word $w = def_{\$\to\epsilon}(w')$ we have that $w \in def(v_{i_1} \dots v_{i_m})$ and $def(v_{i_1} \dots v_{i_m}) \subseteq Q$. Since we showed that for each $w \in Q$ there exists a $w' \in Q'$, we imply that each word of Q can be represented by some word in Ω^* which belongs to the $\text{MCR}_{\mathbf{V}}(Q)$. Together with the fact that $def(\text{MCR}_{\mathbf{V}}(Q)) \subseteq Q$, from the above we conclude the exactness of $\text{MCR}_{\mathbf{V}}(Q)$.

"only if." From the exactness $def(\text{MCR}_{\mathbf{V}}(Q)) = Q$, we have that for each word $w \in Q$ there exists a word $v_{i_1} \dots v_{i_m} \in \text{MCR}_{\mathbf{V}}(Q)$ such that $w \in def(v_{i_1} \dots v_{i_m})$. But, this means that w can be written as $w = u_1 \dots u_m$, where $u_i \in V_i$, for $i \in [1, m]$. From this, we have that $w' = \$u_1\$\$u_2\$ \dots \$u_m\$ \in Q'$. Since all the above hold for each word of Q, the condition (1) follows.

Regarding the condition (2) recall that because of the markers \$ we have that $\text{ECPR}_{\mathbf{V}'}(Q') = \text{MCR}_{\mathbf{V}'}(Q')$. From the construction of Q', observe that for each word $v_{i_1} \dots v_{i_m} \in \text{MCR}_{\mathbf{V}}(Q)$ the corresponding word $v'_{i_1} \dots v'_{i_m}$ is in the $\text{MCR}_{\mathbf{V}'}(Q')$. From this, it is not difficult to conclude that $\text{MCR}_{\mathbf{V}'}(Q')$, and in turn $\text{ECPR}_{\mathbf{V}'}(Q')$ are exact with respect to Q'. ■

Based on the above theorem we give the following theorem regarding the optimality of the construction given in Theorem 28 for testing the exactness of the ECPR of a query Q.

Theorem 30 *Given a query Q and a set $\mathbf{V} = \{V_1, \dots, V_n\}$ of view definitions, the problem of testing $def(\text{ECPR}_{\mathbf{V}}(Q)) = Q$ is 2EXPSPACE complete.*

PROOF. For the upper bound we refer to Theorem 28. For the lower bound consider the reduction from the exactness problem for the MCR of an arbitrary query and an arbitrary set of view definition to the testing of the conditions (1) and (2) in Theorem 29. Observe that the automaton for the modified query of the first condition, since it is being obtained by a polynomial size transducer, is polynomial as well. This means in turn that to test the language equality of the first condition of Theorem 28 is in

PSPACE. So, since the exactness problem for the MCR is 2EXPSPACE complete, the lower bound for the exactness problem of the ECPR is also 2EXPSPACE. ■

Theorem 31 *If for all the queries Q and sets of views $\mathbf{V}$, we were able to compute the $\text{MCPR}_{\mathbf{V}}(Q)$ in 2EXPTIME, then we would be able to decide the exactness of $\text{ECPR}_{\mathbf{V}}(Q)$ in 2EXPTIME, which is impossible by Theorem 30, unless 2EXPTIME = 2EXPSPACE.*

PROOF. Consider the $\text{ECPR}_{\mathbf{V}}(Q)$. Recall that this is 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 *def* is contained in the query Q. So, such words qualify for inclusion in $\text{MCPR}_{\mathbf{V}}(Q)$. Hence, we have that $\text{ECPR}_{\mathbf{V}}(Q) \subseteq \text{MCPR}_{\mathbf{V}}(Q)$. On the other hand, observe that if $\text{ECPR}_{\mathbf{V}}(Q)$ is *exact* then we must have $\text{ECPR}_{\mathbf{V}}(Q) = \text{MCPR}_{\mathbf{V}}(Q)$. Now, we can test the exactness of $\text{ECPR}_{\mathbf{V}}(Q)$ by testing the condition $\text{MCPR}_{\mathbf{V}}(Q) \subseteq \text{ECPR}_{\mathbf{V}}(Q)$, which is equivalent with the non emptiness of $\text{MCPR}_{\mathbf{V}}(Q) \cap (\text{ECPR}_{\mathbf{V}}(Q))^c$. Since by construction, we get a DFA for the $\text{ECPR}_{\mathbf{V}}(Q)$, to complement does not add anything to the complexity of testing the emptiness of the above intersection. So, this test can be done in polynomial time in the size of $\text{MCPR}_{\mathbf{V}}(Q)$ and $\text{ECPR}_{\mathbf{V}}(Q)$. Now, if we were able to generate the $\text{MCPR}_{\mathbf{V}}(Q)$ in, say, 2EXPTIME, and since we are able to generate the $\text{ECPR}_{\mathbf{V}}(Q)$ in 2EXPTIME, then we could test the exactness of $\text{ECPR}_{\mathbf{V}}(Q)$ in 2EXPTIME, which is impossible by Theorem 30, unless 2EXPTIME = 2EXPSPACE. ■

From the above theorem, we conclude that we cannot compute the $\text{MCPR}_{\mathbf{V}}(Q)$ in 2EXPTIME, unless 2EXPTIME = 2EXPSPACE. Now, the question whether our 3EXPTIME algorithm for computing the $\text{MCPR}_{\mathbf{V}}(Q)$ is optimal or not, is equivalent to the (clasical) question of the existence of some complexity gap between 2EXPSPACE and 3EXPTIME [Sip96].

Chapter 5

Query rewriting using views under constraints.

5.1 Introduction

In this chapter, our aim is to reason about query containment and query rewriting using views, all under constraints. The constraints are facts that we know or learn about the structure of the databases, on which a query is to be evaluated. Intuition says that if we have some knowledge about the territory we are going to navigate, then we can navigate more wisely. Notably, constraints for semistructured data are investigated in [AV99, BFW98, BFW99, DT2001]. In [AV99], *local path constraints* are introduced. As defined there, a path constraint with respect to a node a of the database, is a pair of regular path queries (Q_1, Q_2), such that in the intended databases, the set of nodes reachable from a along paths spelling words in Q_1, is a subset of the nodes reachable from a along paths spelling words in Q_2.

An important extension of path constraints for databases having a special root node, say r, has been considered in [BFW98, BFW99]. There, a path constraint holds only from nodes a that are reachable from r, by following paths labeled with words in a prefix language which is regular as well.

In all of [AV99, BFW98, BFW99], the *constraint implication* problem is studied. Namely, the implication problem is to test whether a new constraint follows from the ones already known. As a constraint is a query containment in [AV99], by solving the implication problem, the containment problem is being solved as well. However, there are limitations. The implication is based only on the constraints holding in (from) a particular node, and so, it doesn't take into consideration other constraints holding in nearby nodes. Also, the constraints holding in a node only imply constraints holding in that same node. As a result, the methods presented in [AV99] can only be used to decide containment of regular path queries starting from the same node as the local constraints, and under the assumption that there are no other constraints holding in the nearby nodes, which the queries can eventually reach.

In [BFW98, BFW99], for the databases having the special root node, it is shown that in general, implication of extended path constraints is undecidable, and the papers leave open the question of query containment. Finally, we can also view [DT2001] as solving containment of regular path queries under constraints, for restricted classes of regular path queries, namely those expressible by first order logic.

In our study we consider general semistructured databases (as defined in Chapter 1), which do not have any special root nodes. We capture (partial) knowledge about such databases, with path constraints where each constraint is a pair of regular path queries for which the containment or equality of their answer sets on the target databases is known to hold. The queries in such constraints ask for pairs of nodes connected by paths spelling words in the corresponding queries, rather than the nodes reachable by such paths from some root. Our constraints are a semantic generalization of the path constraints in [AV99], while they can be considered as a special case of the constraints in [BFW98, BFW99], when the prefix language is "everything" i.e. the star of the database alphabet. With our semantics for the regular path constraints, we eliminate the afore mentioned limitations associated with the constraints in [AV99], and we are able to capture most of the practical cases for which the constraints in [BFW98, BFW99] are motivated.

We study query containment in this general setting. Query containment is considered starting from *all* the nodes of the database, not just from a special node.

Then, based on the query containment, we reason about the query rewriting using views. We define *constrained rewritings* and give a characterization that enables their computation. We demonstrate that when we take constraints into account, we can always compute more useful rewritings, which use the views optimally. As mentioned before, query rewriting using views offers substantial optimization when the views are relevant to the query, and by using constraints we make more room for such relevance.

Even if the queries and the views are to be evaluated starting from all the nodes in the database, as is shown in Chapter 3, a rewriting can be efficiently used to optimize the query evaluation in the case when the query is to be computed, or the views have been computed, starting from some nodes only. Such partial query/view evaluations usually occur when dealing with regular path atoms of conjunctive regular path queries. A formal algorithm for this case is given in Section 4.4. The algorithm is based on the intuition of "un-rewriting" when reaching nodes from which a view has not been evaluated.

Query rewriting using views has also other applications apart from traditional query optimization. In the following example we illustrate how beneficial our rewritings are in a Web based scenario. The example also shows the need for the generalized path constraints used in this chapter.

Let's consider again the HTML pages of the multi-site Web of Ericsson Inc. As mentioned in Chapter 1, for the appropriate abstraction level, let there be a function mapping the HREF links to symbols of an alphabet. For example, links for the Ericsson Canada site could be labeled with *"canada."* By browsing, the reader can easily verify that the following constraints hold from *all* the nodes of the Ericsson Web.

- $canada \,.\, products \,.\, mobile\ systems \,.\, mobile\ internet = canada \,.\, mobile\ internet$
- $italy \,.\, tecnologie \,.\, bluetooth \,.\, white\ papers =$
  $canada \,.\, technology \,.\, bluetooth \,.\, white\ papers$

We can continue a long way learning constraints like these. Observe that the above constraints also hold from the nodes not having at all a link labeled *"canada"*

or *"italy."* In such cases, the constraints hold because the left-hand side and the right-hand side queries have both empty answer sets.

Let's take a closer look at the first constraint. It is true because at the moment there exists a shortcut link labeled *"mobile systems"* in the main page of Ericsson Canada. As a matter of fact, such shortcuts are temporary and will be replaced with something else in a couple of days. However, we would like to answer the users still asking queries like $Q = \_^* \,.\, canada \,.\, mobile\ internet \,.\, white\ papers$. By considering some important long browsing paths as regular path views, e.g. $V = \_^* \,.\, canada \,.\, products \,.\, mobile\ systems \,.\, mobile\ internet \,.\, white\ papers$, we could rewrite Q as $V \,.\, white\ papers$ and answer the user.

Now consider the second constraint. It is clear that, if the user gives a query having a subquery asking for Canadian technology Bluetooth white papers, and the Ericsson Canada site is down for maintenance, then we could answer the user by using a query rewriting with a corresponding path view in the Italian site. Obviously, in neither case would we be able to have a rewriting if we lacked the knowledge captured in the constraints.

5.2 Background

Path queries and constraints. As we know, a *(regular) path query*[1] Q is a finite or infinite (regular) language over the alphabet Δ.

A query Q_1 is *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff $ans(Q_1, DB) \subseteq ans(Q_2, DB)$, for all DB's. We say that Q_1 is *equivalent* to Q_2 and write $Q_1 \equiv Q_2$, when $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$. It is easy to see that the above query containment coincides with the (algebraic) language containment of Q_1 and Q_2 and the query equivalence coincides with the language equality, i.e. $Q_1 \sqsubseteq Q_2$ iff $Q_1 \subseteq Q_2$ and $Q_1 \equiv Q_2$ iff $Q_1 = Q_2$. This is the reason why we silently, in all the previous chapters, considered only language

[1] A query Q could be non-regular as well. However, perhaps since very few problems would be decidable, larger classes of queries have not been considered in previous literature. We also restrict ourselves to regular path-queries, except for a technicality in Section 5.5.

containment, and used for simplicity only the symbol "$\subseteq$."

If for two queries Q_1 and Q_2, we have in general $Q_1 \not\sqsubseteq Q_2$, we could be interested in the set $\{DB\}$ of databases, such that $ans(Q_1, DB) \subseteq ans(Q_2, DB)$. Clearly, an expression $Q_1 \sqsubseteq Q_2$ could be seen as a constraint restricting the set of databases to only those satisfying the containment. We now proceed with the following definition:

Definition 6

1. A *path constraint* is an expression of the form $Q_1 \sqsubseteq Q_2$, where Q_1 and Q_2 are path queries.
2. A database DB *satisfies* a path constraint, denoted $DB \models Q_1 \sqsubseteq Q_2$, if $ans(Q_1, DB) \subseteq ans(Q_2, DB)$.
3. DB satisfies a set $\mathcal{C}$ of path constraints, denoted by $DB \models \mathcal{C}$, if it satisfies each constraint in $\mathcal{C}$.
4. A query Q_1 is contained in a query Q_2 under a finite set of constraints $\mathcal{C}$, denoted $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$, if for each database DB such that $DB \models \mathcal{C}$, we also have $ans(DB, Q_1) \subseteq ans(DB, Q_2)$.
5. A query Q_1 is equivalent to a query Q_2 under a finite set of constraints $\mathcal{C}$, denoted $Q_1 \equiv_{\mathcal{C}} Q_2$, if $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$ and $Q_2 \sqsubseteq_{\mathcal{C}} Q_1$.
6. If two queries Q_1 and Q_2 are *words*, i.e., simply sequences of labels, the constraint $Q_1 \sqsubseteq Q_2$ is called a *word constraint.* Word constraints will also be written as $w_1 \sqsubseteq w_2$, when Q_1 equals w_1 and Q_2 equals w_2.

We can easily see now, that the query containment (equivalence) under constraints, no longer coincides with the containment (equality) of regular languages.

Rewrite systems. A (semi-Thue) *rewrite system* $\mathcal{R}$ is a finite subset of $\Delta^* \times \Delta^*$. The elements of $\mathcal{R}$ are called *rewrite rules.* A rewrite system $\mathcal{R}$ induces a *single-step*

reduction relation $\rightarrow_{\mathcal{R}}$ over Δ^* defined as

$$\begin{aligned} \rightarrow_{\mathcal{R}} \;=\; & \{(v,w) : v = xty \text{ and } w = xuy \\ & \text{for some } (t,u) \in \mathcal{R}, \text{ and } x,y \in \Delta^*\}. \end{aligned}$$

We denote with $\stackrel{*}{\rightarrow}_{\mathcal{R}}$ the reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$. Testing whether a given pair (v,w) is an element of $\stackrel{*}{\rightarrow}_{\mathcal{R}}$ is called the *rewrite problem* for $\stackrel{*}{\rightarrow}_{\mathcal{R}}$. We shall sometimes use infix notation for the reduction relation and write $(u,w) \in \rightarrow_{\mathcal{R}}$ as $u \rightarrow_{\mathcal{R}} v$. For $\stackrel{*}{\rightarrow}_{\mathcal{R}}$ the convention is similar.

The following fundamental result is well known (see e.g. [BO93])

Theorem 32 *The rewrite problem is undecidable in general.*

We define the set of *rewrite ancestors* and *rewrite descendants* of a word w, with respect to $\mathcal{R}$, to be $anc_{\mathcal{R}}(w) = \{x \in \Delta^* : x \stackrel{*}{\rightarrow}_{\mathcal{R}} w\}$ and $desc_{\mathcal{R}}(w) = \{x \in \Delta^* : w \stackrel{*}{\rightarrow}_{\mathcal{R}} x\}$, respectively. For a language $L \subseteq \Delta^*$ we set $anc_{\mathcal{R}}(L) = \cup_{w \in L} anc_{\mathcal{R}}(w)$, and $desc_{\mathcal{R}}(L) = \cup_{w \in L} desc_{\mathcal{R}}(w)$,

We say that two Δ-words u and w are equivalent with respect to $\mathcal{R}$ iff $v \stackrel{*}{\rightarrow}_{\mathcal{R}} w$ and $w \stackrel{*}{\rightarrow}_{\mathcal{R}} v$. We denote the equivalence class of a word w with respect to $\mathcal{R}$ by $[w]_{\mathcal{R}}$, or simply by $[w]$, if $\mathcal{R}$ is evident from the context.

Let us now consider a reduction relation we would obtain by applying the rewrite rules only in the prefix of words. Formally, we define the *prefix-reduction relation*

$$\begin{aligned} \mapsto_{\mathcal{R}} \;=\; & \{(v,w) : v = ty, \text{ and } w = uy, \\ & \text{for some } (t,u) \in \mathcal{R} \text{ and } y \in \Delta^*\}. \end{aligned}$$

We will denote the reflexive an transitive closure of $\mapsto_{\mathcal{R}}$ with $\stackrel{*}{\mapsto}_{\mathcal{R}}$.

We define the set of *prefix rewrite ancestors* and *prefix rewrite descendants* of a word w, with respect to $\mathcal{R}$, to be $panc_{\mathcal{R}}(w) = \{x \in \Delta^* : x \stackrel{*}{\mapsto}_{\mathcal{R}} w\}$ and $pdesc_{\mathcal{R}}(w) = \{x \in \Delta^* : w \stackrel{*}{\mapsto}_{\mathcal{R}} x\}$, respectively. For a language $L \subseteq \Delta^*$ we set $panc_{\mathcal{R}}(L) = \cup_{w \in L} panc_{\mathcal{R}}(w)$ and $pdesc_{\mathcal{R}}(L) = \cup_{w \in L} pdesc_{\mathcal{R}}(w)$.

Finite transducers and rational relations. Let $\mathcal{T} = (P,\ I,\ O,\ \delta,\ s,\ F)$ be a finite transducer. We repeat here some concepts introduced in Chapter 1 for ease of reading, and also introduce some more definitions and finally a lemma. As mentioned in Chapter 1, we also use the symbol $\mathcal{T}$ to denote the set of all pairs $(v, w) \in I^* \times O^*$, where w is an output of $\mathcal{T}$ when providing v as input. Finally, $\mathcal{T}$ can also be seen as a mapping from languages to languages, and we write

$$\mathcal{T}(L) = \{w : (v, w) \in \mathcal{T},\ \text{for some } v \in L\}.$$

It is well known that $\mathcal{T}(L)$ is regular whenever L is.

A possibly infinite subset of $\Delta^* \times \Delta^*$ will be called a *word relation*. A word relation $\mathscr{R}$ is *rational* if there exists a transducer $\mathcal{T} = (P,\ \Delta,\ \Delta,\ \delta,\ s,\ F)$, such that $\mathscr{R} = \mathcal{T}$. We say that the transducer $\mathcal{T}$ *recognizes* the relation $\mathscr{R}$. It is easy to see that, if we reverse the input with the output in a transducer $\mathcal{T}$, we get a transducer recognizing the inverse $\mathscr{R}^{-1}$ of the relation $\mathscr{R}$ that $\mathcal{T}$ recognizes. We call the transducer obtained in this way from T, the *inverse transducer* and denote it with $\mathcal{T}^{-1}$.

Let $\mathscr{R}$ be a word relation. We define the powers of $\mathscr{R}$ as follows: $\mathscr{R}^0 = \emptyset, \mathscr{R}^1 = \mathscr{R}$, and

$$\mathscr{R}^{i+1} = \{(vv', ww') : (v, w) \in \mathscr{R}^i,\ \text{and } (v', w') \in \mathscr{R}\}.$$

The *power closure* of $\mathscr{R}$ is defined as $\mathscr{R}^{\otimes} = \bigcup_{i \in \mathbf{N}} \mathscr{R}^i$.

It is also easy to see that class of rational relations is closed under power closure. Formally, we have

Lemma 3 *Let $\mathscr{R}$ be a rational relation. Then $\mathscr{R}^{\otimes}$ is rational.*

PROOF. Construct a transducer $\mathcal{T} = (P,\ \Delta,\ \Delta,\ \delta,\ s,\ F)$ for $\mathscr{R}$. Then, the transducer $\mathcal{T}' = (P,\ \Delta,\ \Delta,\ \delta',\ s,\ F)$, where $\delta' = \delta \cup \{(f, \epsilon, s, \epsilon) : f \in F\}$, recognizes the relation $\mathscr{R}^{\otimes}$. ∎

5.3 Query containment under constraints

In this section we show that for word queries and constraints, the query containment problem is equivalent to the rewrite problem for general semi-Thue systems. Although this tells us that, in general, the query containment is undecidable even for word constraints, the equivalence is important because from that we will derive a characterization for reasoning about the containment of arbitrary regular path queries. In fact, there are useful subclasses of word constraints for which the (arbitrary) query containment is decidable, and we show one such subclass in the next section.

Let $\mathcal{C} = \{t_i \sqsubseteq u_i : i \in [1,n]\}$ be a set of word constraints. Consider the corresponding semi-Thue word rewriting system $\mathcal{R}_c = \{(t_i, u_i) : i \in [1,n]\}$. With slight abuse of notation, we shall denote the induced relations by $\rightarrow_c$, $\stackrel{*}{\rightarrow}_c$, $anc_c(\cdot)$, and $desc_c(\cdot)$. Now, suppose we are given two word queries w_1 and w_2, and we want to test the containment $w_1 \sqsubseteq_{\mathcal{C}} w_2$. The following theorem shows that deciding the above word query containment coincides with deciding the $w_1 \stackrel{*}{\rightarrow}_c w_2$ word rewrite problem.

Theorem 33 *$w_1 \sqsubseteq_{\mathcal{C}} w_2$ iff $w_1 \stackrel{*}{\rightarrow}_c w_2$.*

PROOF. It is easy to see that if $w_1 \stackrel{*}{\rightarrow}_c w_2$ then $w_1 \sqsubseteq_{\mathcal{C}} w_2$. To prove the converse, we will build a database DB_c such that $DB_c \models \mathcal{C}$ and

(§) If $DB_c \models w_1 \sqsubseteq w_2$ then $w_1 \stackrel{*}{\rightarrow}_c w_2$.

To see why this is important, suppose that we have this database DB_c and also have that $w_1 \sqsubseteq_{\mathcal{C}} w_2$. Since $w_1 \sqsubseteq_{\mathcal{C}} w_2$ and $DB_c \models \mathcal{C}$, we get that $DB_c \models w_1 \sqsubseteq w_2$. By (§), this means in turn that $w_1 \stackrel{*}{\rightarrow}_c w_2$.

In order to construct the above mentioned database we generalize the construction of Lemma 4.4 in [AV99]. Since the development is neither trivial nor derivable from [AV99], we go in the details of the construction and its properties.

Let k be the number of symbols in the longer of the two words w_1 and w_2. We

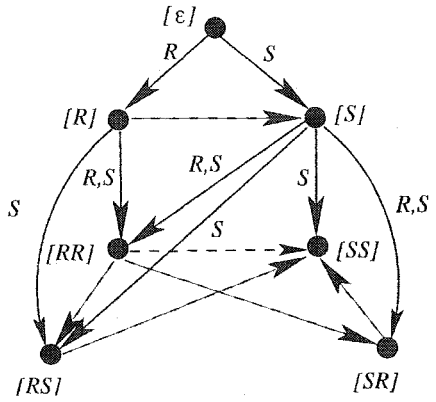

Figure 17: The construction of DB_c

then set $DB_c = (N_c, E_c)$, where

$$N_c = \{[x] : x \in \Delta^* \text{ and } |x| \leq k\},$$

where $[x]$ is the equivalence class of x with respect to the relation $\overset{*}{\rightarrow}_c$, and

$$E_c = \{([x], R, [y]) : ([x], [y]) \in N_c \times N_c \text{ and } y \in anc_c(xR)\}.$$

It should be emphasized that the definition of E_c does not require that the $[xR]$ is a node in N_c.

We note here that to "really" build this database we need to be able to decide whether $y \in anc_c(x)$ for any $[x]$ and $[y]$ in N_c. Clearly, this is possible only if we are able to decide the rewrite problem for the corresponding semi-Thue system. However, what we actually show is the existence of DB_c, although its construction might not be executable by a halting Turing machine.

For an example of the construction of the database DB_c, let's take $\mathcal{C} = \{R \sqsubseteq S\}$, that gives a corresponding rewrite system $\{(R, S)\}$, for which the rewrite problem is decidable. The construction for $k = 2$ is presented in Figure 17. The solid arrows represent database edges, while a dashed arrow from $[x]$ to $[y]$ indicates that $y \in anc_c(x)$.

We prove that any database DB_c, constructed as described above, always satisfies the set $\mathcal{C}$ of word constraints, from which it was derived. For this we need the following

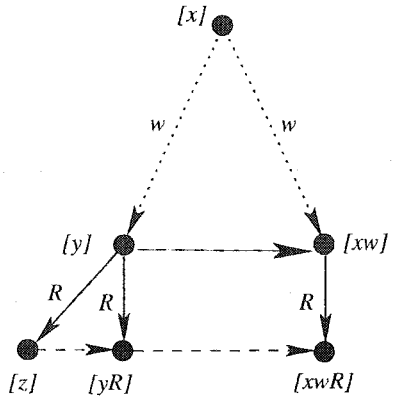

Figure 18: Proof of Lemma 4

lemma, which says that, starting from a node, say $[x]$, and following paths labeled with some (non-empty) word w, we reach all the nodes corresponding to the ancestors of $[xw]$. In order to simplify the notation, we will denote with $Reach([x], w)$ the set of nodes that can be reached from $[x]$ by following paths labeled with w in DB, i.e.

$$Reach([x], w) = \{[y] : ([x], [y]) \in ans(w, DB_c)\},$$

and with $Anc([x])$ we will denote the set of nodes $[y]$, such that $y \xrightarrow{*}_c x$, i.e.

$$Anc([x]) = \{[y] \in N_c : y \in anc_c(x)\}.$$

Again, we emphasize that $[x]$ is not required to be a node in DB_c for the definition of $Anc([x])$.

Lemma 4 *For each $w \in \Delta^+$ and $[x] \in N_c$, we have that*

$$Reach([x], w) = Anc([xw]).$$

PROOF. We proceed by induction. First, observe that by the construction, for $R \in \Delta$ we have that $[y] \in Reach([x], R)$ iff $y \in anc_c(xR)$, i.e. $[y] \in Anc([xR])$.

Suppose (by induction on the length of w) that $Reach([x], w) = Anc([xw])$, and let $R \in \Delta$. Then, by the induction hypothesis and the construction of DB_c, $Reach([x], wR)$ contains $Anc([xwR])$.

Let's show the converse. If $Reach([x], wR)$ is empty, then the claimed equality follows. Let $[z]$ be in $Reach([x], wR)$. Then, there exists y, such that $[y]$ is reachable from $[x]$ with a path spelling w, and there is an R-edge from $[y]$ to $[z]$ (see Figure 18). From the induction hypothesis we get $[y] \in Anc([xw])$. Also, by the construction of DB_c, $[z] \in Anc([yR])$. Now, since $[y] \in Anc([xw])$ we have that $[yR] \in Anc([xwR])$, so $Anc([yR]) \subseteq Anc([xwR])$. Thus, $[z]$ is in $Anc([xwR])$, and finally we have that $Reach([x], wR) = Anc([xwR])$. ■ Lemma 4

Let's return to the proof of Theorem 33. For any constraint $t_i \sqsubseteq u_i$ in $\mathcal{C}$, we have $Anc([t_i]) \subseteq Anc([u_i])$. This implies that, for any node $[x] \in DB_c$, $Anc([xt_i]) \subseteq Anc([xu_i])$, which by Lemma 4 is equivalent with $Reach([x], t_i) \subseteq Reach(\ [x], u_i)$. Since $[x]$ was an arbitrary DB_c node, we have that $DB_c \models t_i \sqsubseteq u_i$. Hence, $DB_c \models C$.

Consider now $x = \epsilon$. Lemma 4 transforms in this case into $Reach([\epsilon], w) = Anc([w])$. Then, if $DB \models w_1 \sqsubseteq w_2$, we conclude that $Reach([\epsilon], w_1) \subseteq Reach([\epsilon], w_2)$, which in turn implies that $Anc([w_1]) \subseteq Anc([w_2])$. Recall that by definition we have that $[w] \in Anc([w])$, where w is a word. So, $Anc([w_1]) \subseteq Anc([w_2])$ implies $[w_1] \in Anc([w_2])$, which finally implies $w_1 \xrightarrow{*}_c w_2$.

■ Theorem 33

From the above theorem and Theorem 32 we get the following corollaries.

Corollary 7 *Query containment under constraints is undecidable.*

Corollary 8 *The query containment problem for a class of word constraints is decidable iff the the word rewrite problem for the corresponding class of semi-Thue rewrite systems is decidable.*

The second corollary is a positive result, so we would be interested in knowing whether the general query containment is decidable for a subclass of word constraints with

decidable word query containment. Unfortunately, the answer to this question is negative as we show by the following lemmas and theorems. Nevertheless, they provide the basis for showing (in the next section) that a useful subclass of word constraints has decidable general query containment problem.

The following lemma is a generalization of Lemma 4.6 in [AV99].

Lemma 5 *Let $\mathcal{C}$ be a finite set of word constraints and Q_1, Q_2 regular path queries. If $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$ then for each $w_1 \in Q_1$ there exists $w_2 \in Q_2$ such that $w_1 \sqsubseteq_{\mathcal{C}} w_2$.*

PROOF. Let $w_1 \in Q_1$. Then, if $\mathcal{C} \models Q_1 \sqsubseteq Q_2$ we have that $\mathcal{C} \models w_1 \sqsubseteq Q_2$. Now, let's consider the database DB_c described in the proof of Theorem 33, for $k > |w_1|$. Since DB_c satisfies $\mathcal{C}$, it must also satisfy $w_1 \sqsubseteq Q_2$. So, the nodes that we can reach from let's say $[\epsilon]$ by following w_1 are a subset of the nodes that we can reach by following the words in Q_2. Formally, we have

$$Reach([\epsilon], w_1) \subseteq \bigcup_{w_2 \in Q_2} Reach([\epsilon], w_2).$$

Clearly, the node $[w_1] \in Reach([\epsilon], w_1)$, and so we have $[w_1] \in \bigcup_{w_2 \in Q_2} Reach([\epsilon], w_2)$. It follows that that the node $[w_1] \in Reach([\epsilon], w_2)$ for some $w_2 \in Q_2$. It easy to see that by the construction of DB_c, following paths spelling some word w, we reach only nodes $[w']$ such that $w' \in anc_c(w)$. So, for the word $w_2 \in Q_2$, since we reach by spelling it the node $[w_1]$, we get that $w_1 \in anc_c(w_2)$. This means that $w_1 \xrightarrow{*}_c w_2$, which as we showed in Theorem 33, coincides with $w_1 \sqsubseteq_{\mathcal{C}} w_2$. ■

From the above lemma and Theorem 33, we can easily see that the following is true.

Theorem 34 *Given a finite set $\mathcal{C}$ of word constraints, and (general) regular path queries Q_1 and Q_2, we have that $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$ iff $Q_1 \subseteq anc_c(Q_2)$.*

Let $\mathcal{R}$ be a word rewriting system and $\mathcal{R}^{-1}$ its inverse, obtained by reversing the direction of the pairs in $\mathcal{R}$. Clearly, the inverse of a word rewriting system is also a word rewriting system. It is easy to verify the following lemma.

Lemma 6 $\xrightarrow{*}_{\mathcal{R}^{-1}} = (\xrightarrow{*}_{\mathcal{R}})^{-1}$ *and* $anc_{\mathcal{R}}(L) = desc_{\mathcal{R}^{-1}}(L)$, *for each language* $L \subseteq \Delta^*$.

Now, we are ready to show that containment for (general) regular path queries, under finite sets of word constraints with decidable word query containment, is undecidable.

Theorem 35 *There exists a class of word constraints with decidable word query containment, but with undecidable query containment in general.*

PROOF. The proof is based on the notion of monadic word rewriting systems. A word rewriting system $\mathcal{R}$ is *length-reducing* if $|t| > |u|$ for each pair $(t, u) \in \mathcal{R}$. A word rewriting system $\mathcal{R}$ is *monadic* if it is length-reducing, and $u \in \Delta$ for each pair $(t, u) \in \mathcal{R}$. It is well known that the rewrite problem for length-reducing systems is decidable, and so it is for monadic systems [BO93].

Let us now consider the universality problem for the context free class CFG of grammars. Formally, this problem says that the language

$$\{<G> : G \in CFG \text{ and } L(G) = \Delta^*\},$$

where $<G>$ is a grammar encoding, is undecidable [Sip96]. Let $CFG_{\geq 2}$ be the subclass of context free grammars whose productions have right side of length greater or equal to 2. Without loss of generality, we have that the slightly modified language

$$\{<G> : G \in CFG_{\geq 2} \text{ and } L(G) = \Delta^+ - \Delta\}$$

is also undecidable. We will now present a reduction from this undecidable problem.

Let G be a grammar in $CFG_{\geq 2}$ with nonterminal symbols Γ, start symbol S, and productions of the form $(head, body)$. Clearly,

$$\mathcal{R}_G = \{(body, head) : (head, body) \text{ is a production in } G\}$$

is a monadic rewrite system over the extended alphabet $\Delta \cup \Gamma$. In this proof we will consider $\Delta \cup \Gamma$ as the database alphabet.

We take $Q_1 = \Delta^*$, $Q_2 = S$, and $\mathcal{C} = \{body \sqsubseteq head : (body, head) \in \mathcal{R}_G\}$. Since $\mathcal{R}_G$ belongs to the class of monadic rewrite systems, which has a decidable word rewrite

problem, from Corollary 8 it follows that $\mathcal{C}$ belongs in a subclass of word constraints with a decidable word query containment problem. Clearly, $L(G) \subseteq (\Delta^+ - \Delta)$. On the other hand, based on Lemma 6, we have that $(\Delta^+ - \Delta) \subseteq L(G)$ iff $(\Delta^+ - \Delta) \subseteq anc_c(S) \cap (\Delta^+ - \Delta)$, which is equivalent with $(\Delta^+ - \Delta) \subseteq anc_c(S)$, and this is finally equivalent by Theorem 34 with $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$.

■

5.4 A subclass with decidable query containment

Often, in a Web based scenario we can encounter sets of constraints, such that any left-hand side overlaps (if it does) with some right-hand side only by prefix. For example, the constraints presented in Section 5.1 were such ones. The high frequency of this type of constraints is because they, after a short prefix, start expressing local information about a node, say a, and the prefix is nothing else but the link (or sequence of few links) that we need to navigate from another node in order to reach a. In our example of Section 5.1, from any HTML page of the Ericsson Web, we can jump to the Ericsson Canada main page, only if there is a link labeled *"canada"* –the prefix– and then, various local facts can hold starting from there.

The local constraints –connected with particular nodes– have been very well motivated in [AV99]. However, as explained in Section 5.1, there are limitations associated with the way the constraints are treated in [AV99], most important of which are: (a) the non-extensibility of the methods to decide query containment when the queries start from other nodes than the constraints, and (b) not taking into consideration the interaction with other local constraints in nearby nodes. By prefixing the local constraints with the link label(s) needed to reach the relevant node, the local constraints are transformed into global ones, and we will give decision procedures that do not have the mentioned limitations [2].

[2] If from some node there is no link(s) leading to the relevant node, then the constraints still hold since the sets of the reachable nodes with the left- and right-hand sides (words) are empty.

We also relax the notion of "overlapping" to be more "generous," in the sense that it allows a left-hand side to not overlap at all with any right-hand side, and it also allows strictly internal sub-words to overlap. This relaxation of the overlapping allows for expressing not only local constraints, but also "pure" global constraints as it is the second constraint in Section 5.1 or the constraint *canada . events . bluetooth . latest* $\sqsubseteq$ *bluetooth . events . latest*, in which the left-hand side overlaps with the right-hand side by a strictly internal subword.

Now let's formally define our subclass of word constraints. We will start by considering rewrite systems $\mathcal{R}$. We first set $left(\mathcal{R}) = \{t : (t, u) \in \mathcal{R}, \text{ for some } u \in \Delta^*\}$, and $right(\mathcal{R}) = \{u : (t, u) \in \mathcal{R}, \text{ for some } t \in \Delta^*\}$. Now we have

Definition 7 ([Sen90]). Let $\mathcal{R}$ be a rewrite system. Then $\mathcal{R}$ is said to be

1. *Internal overlapping*, if some $t \in left(\mathcal{R})$ is a substring of a word $u \in right(\mathcal{R})$ or vice versa.

2. *Right overlapping*, if there are $x, y, w \in \Delta^*$, $t \in left(\mathcal{R})$, and $u \in right(\mathcal{R})$, such that $t = xw$ and $u = wy$, for $w \neq \epsilon$.

3. *Left overlapping*, if there are $x, y, w \in \Delta^*$, $t \in left(\mathcal{R})$, and $u \in right(\mathcal{R})$, such that $u = yw$ and $t = wx$, for $w \neq \epsilon$.

If a rewrite system $\mathcal{R}$ is neither internal, nor left or right overlapping, we say that $\mathcal{R}$ is *prefix overlapping*. Clearly, a prefix overlapping system allows in addition for non-overlap-ping at all, or overlappings of left- with right-hand sides by strictly internal subwords.

The following result has recently been obtained by Caucal.

Theorem 36 ([Ca2001]). *Let $\mathcal{R}$ be a prefix overlapping rewrite system. Then, we have that*

$$\xrightarrow{*}_{\mathcal{R}} = (\mapsto^*_{\mathcal{R}})^{\otimes}.$$

Abiteboul and Vianu have an important related recent result.

Theorem 37 ([AV99]). *Let $\mathcal{R}$ be an arbitrary rewrite system and $L \subseteq \Delta^*$ a regular language. Then, the set of prefix rewrite ancestors $panc_{\mathcal{R}}(L)$ is regular as well, and computable in PTIME from an automaton for L.*

Based on the above theorem and reasoning similarly as for Lemma 6, we also have

Theorem 38 *Let $\mathcal{R}$ be an arbitrary rewrite system and $L \subseteq \Delta^*$ a regular language. Then, the set of prefix rewrite descendants $pdesc_{\mathcal{R}}(L)$ is regular as well, and computable in PTIME from an automaton for L.*

Here we strengthen these results for $\overset{*}{\mapsto}_{\mathcal{R}}$. We show that $\overset{*}{\mapsto}_{\mathcal{R}}$ is rational for any arbitrary rewrite system $\mathcal{R}$.

We do this in order to compute $anc_{\mathcal{R}}(L)$ for a regular language L and a prefix overlapping rewrite system $\mathcal{R}$. It can be easily verified that, $anc_{\mathcal{R}}(L)$ is not equal in general with $(panc_{\mathcal{R}}(L))^{\otimes}$, as we could think at the first glance on Theorem 36.

Let $\mathcal{R}$ be a rewrite system, and (v, w) a pair of words. Then the *prefix lineage through (v, w) induced by $\mathcal{R}$* is defined as

$$plin_{\mathcal{R}}(v, w) = \{(x, y) : x \in panc_{\mathcal{R}}(v) \text{ and } y \in pdesc_{\mathcal{R}}(w)\}.$$

We have the following theorem.

Theorem 39 *Let $\mathcal{S} = \cup\{plin_{\mathcal{R}}(t, u) : (t, u) \in \mathcal{R}\} \cup \{\epsilon, \epsilon\}$. Then, $\overset{*}{\mapsto}_{\mathcal{R}} = \mapsto_{\mathcal{S}}$.*

PROOF. We will prove that, for the rewrite system $\overset{*}{\mapsto}_{\mathcal{R}}$, $w_1 \overset{*}{\mapsto}_{\mathcal{R}} w_2$ if and only if $w_1 = w_2$, or, $w_1 = xz$ and $w_2 = yz$, where $\{x, y, z\} \subset \Delta^*$, and there exists a a pair $(t, u) \in \mathcal{R}$, such that such that $x \overset{*}{\mapsto}_{\mathcal{R}} t$ and $u \overset{*}{\mapsto}_{\mathcal{R}} y$. Clearly from this, the theorem follows.

The *If*-direction is straightforward. For the converse we will use induction on n, for $w_1 \overset{n}{\mapsto}_{\mathcal{R}} w_2$. For $n = 0$ we must indeed have $w_1 = w_2$.

Suppose that the claim is true for n, and let's show it for $n+1$. Let $w_1 \overset{n+1}{\mapsto}_{\mathcal{R}} w_2$. There exists a word v, such that $w_1 \overset{n}{\mapsto}_{\mathcal{R}} v \mapsto_{\mathcal{R}} w_2$. By the induction hypothesis $w_1 = x_1 z_1$ and $v = y_1 z_1$, and for some $(t_1, u_1) \in \mathcal{R}$ we have that $x_1 \overset{*}{\mapsto}_{\mathcal{R}} t_1$ and $u_1 \overset{*}{\mapsto}_{\mathcal{R}} y_1$.

Since $v \mapsto_{\mathcal{R}} w_2$, there exists a pair $(t_2, u_2) \in \mathcal{R}$, such that $v = t_2 z_2$ and $w_2 = u_2 z_2$. So, $v = t_2 z_2 = y_1 z_1$ and we have that either y_1 is a prefix of t_2, or vice versa.

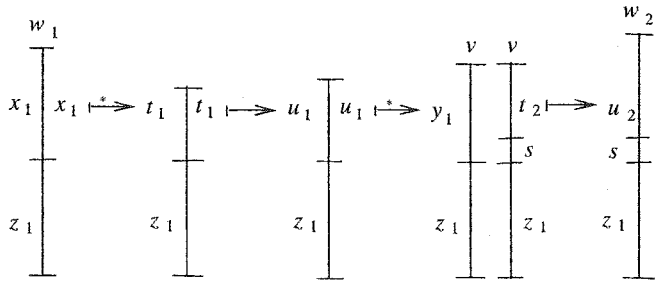

Figure 19: First case of Theorem 39

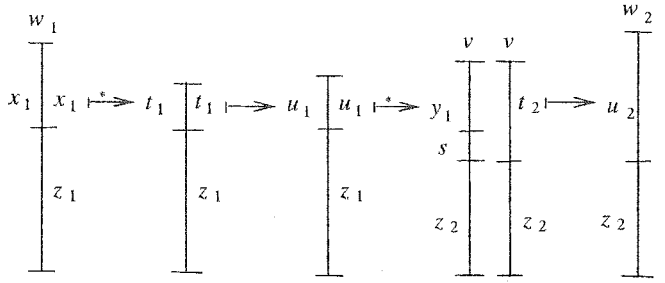

Figure 20: Second case of Theorem 39

In the first case (see Figure 19) t_2 is a prefix of y_1. So, $y_1 = t_2 s$ and $z_2 = s z_1$. Thus, $w_1 = x_1 z_1$ and $w_2 = u_2 z_2 = u_2 s z_1$. By the induction hypothesis ($x_1 \overset{*}{\mapsto}_{\mathcal{R}} t_1$ and $u_1 \overset{*}{\mapsto}_{\mathcal{R}} y_1$), we have that $x_1 \overset{*}{\mapsto}_{\mathcal{R}} t_1 \mapsto_{\mathcal{R}} u_1 \overset{*}{\mapsto}_{\mathcal{R}} y_1$. Since $y_1 = t_2 s$, $u_1 \overset{*}{\mapsto}_{\mathcal{R}} t_2 s \mapsto_{\mathcal{R}} u_2 s$. On the other hand, recall that $w_2 = u_2 s z_1$. So, in total we have $w_1 = x_1 z_1$ and $w_2 = u_2 s z_1$, where $x_1 \overset{*}{\mapsto}_{\mathcal{R}} t_1$ and $u_1 \overset{*}{\mapsto}_{\mathcal{R}} u_2 s$. Hence, in this case we take $x = x_1$, $y = u_2 s$, $z = z_1$, and $(t, u) = (t_1, u_1)$.

In the second case (see Figure 20) we have $t_2 = y_1 s$ and $z_1 = s z_2$. Thus, $w_1 = x_1 z_1 = x_1 s z_2$, and by the induction hypothesis ($x_1 \overset{*}{\mapsto}_{\mathcal{R}} t_1$ and $u_1 \overset{*}{\mapsto}_{\mathcal{R}} y_1$), we have that

$x_1 s \overset{*}{\mapsto}_{\mathcal{R}} t_1 s \mapsto_{\mathcal{R}} u_1 s \overset{*}{\mapsto}_{\mathcal{R}} y_1 s$. Since $y_1 s = t_2$, we have that $x_1 s \overset{*}{\mapsto}_{\mathcal{R}} t_2$ (recall $w_1 = x_1 s z_2$). On the other hand, recall that $w_2 = u_2 z_2$. Hence, in this case we take $x = x_1 s$, $y = u_2$, $z = z_2$, and $(t, u) = (t_2, u_2)$. ■

Theorem 40 *$\overset{*}{\mapsto}_{\mathcal{R}}$ is a rational relation.*

PROOF. Observe that for a pair (v, w), $plin_{\mathcal{R}}(v, w) = panc_{\mathcal{R}}(v) \times pdesc_{\mathcal{R}}(w)$. Since from Theorem 37 and Theorem 38, $panc_{\mathcal{R}}(v)$ and $pdesc_{\mathcal{R}}(w)$ are regular languages, we have from Theorem 39 that the corresponding rewrite system $\mathcal{S}$ is a union of Cartesian products of regular languages.

It is not difficult to construct a transducer for a Cartesian product $L \times M$ of two regular languages. For this, let $A_L = (P_L, \Delta, \delta_L, s_L, F_L)$ and $A_M = (P_M, \Delta, \delta_M, s_M, F_M)$ be finite automata recognizing L and M, respectively. Then construct the transducer $\mathcal{T}_{L \times M} = (P_L \cup P_M, \Delta, \Delta, \delta, s_L, F_M)$, where δ contains all the tuples (p, a, q) of δ_L expanded as (p, a, q, ϵ), all the tuples (p, b, q) of δ_M expanded as (p, ϵ, q, b), the set $\{(f, \epsilon, s_M, \epsilon) : f \in F_L\}$, and nothing else. It is easily seen that $\mathcal{T}_{L \times M}$ recognizes exactly $L \times M$.

Finally, we construct a transducer for $\mapsto_S$ by taking the (finite) union of the transducers recognizing the above Cartesian products, concatenating at the end with a transducer "leaving everything unchanged." ■

From the above theorem and Theorem 36 we have the following corollary.

Corollary 9 *Let $\mathcal{R}$ be prefix overlapping. Then $\overset{*}{\to}_{\mathcal{R}}$ is rational.*

We say that a finite set $\mathcal{C}$ of word constraints is prefix overlapping, if the corresponding rewrite system $\mathcal{R}_{\mathcal{C}}$ is prefix overlapping. We now finally have

Theorem 41 *Let $\mathcal{C}$ be a prefix overlapping finite set of word constraints, and Q_1 and Q_2 regular path queries. Then, $Q_1 \sqsubseteq_{\mathcal{C}} Q_2$ is decidable and complete in PSPACE.*

PROOF. From all the above it follows that the relation $\xrightarrow{*}_c$ is rational and a transducer $\mathcal{T}_\mathcal{C}$ recognizing it can be computed in polynomial time. It is easy now to verify that

$$anc_c(Q) = \mathcal{T}_\mathcal{C}^{-1}(Q).$$

Based on this fact and Theorem 34, we conclude that deciding the containment $Q_1 \sqsubseteq_\mathcal{C} Q_2$, for general path queries, under a prefix overlapping set $\mathcal{C}$ of word constraints, is equivalent with deciding $Q_1 \subseteq \mathcal{T}_\mathcal{C}^{-1}(Q_2)$, and this is in PSPACE. The lower bound is the same because in the absence of constraints, the query containment coincides with the algebraic containment of regular languages, which is PSPACE complete. ■

5.5 Query rewriting using views under constraints

We will now reason about rewriting of regular path queries using views based on the query containment under constraints.

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. Let $\Omega = \{v_1, \ldots, v_n\}$ be the *outer* or *view alphabet.* Finally, let *def* be the substitution that associates with each view name v_i in Ω alphabet the language V_i.

Suppose now that we have a set $\mathcal{C}$ of constraints available. The question is if can we use the constraints to get "bigger and better" rewritings. For example, let $Q = R_1 \cdots R_{99} R_{100} \cdots R_{200}$, and suppose that we have two views, V_1 and V_2, where $V_1 = S_1 \cdots S_{99}$, $V_2 = R_{101} \cdots R_{199}$, and $\mathcal{C} = \{R_i \sqsubseteq S_i : i \in [1, 99]\} \cup \{S_i \sqsubseteq R_i : i \in [1, 99]\}$. Then, $\text{MCPR}_\mathbf{V}(Q) = R_1 \cdots R_{99} R_{100} v_2 R_{200}$. On the other hand, the rewriting $v_1 R_{100} v_2 R_{200}$ is "bigger and better" because $v_1 R_{100} v_2 R_{200}$ is $\mathcal{C}$-equivalent to Q.

In the rest of this section we show how to obtain such "biggest and best" rewritings, using constraints.

Let $\mathcal{C}$ be a finite set of constraints, and let $L \subseteq \Delta^*$. Then we define the *maximization of L under $\mathcal{C}$*, denoted $max_c(L)$ to be the $\subseteq$-largest language M, such that $M \equiv_{\mathcal{C}} L$.

Intuitively, under constraints we can replace more subwords and still have a rewriting. We capture this intuition by enlarging the view languages V_i to $max_c(V_i)$, for $i \in [1, n]$.

We then define the substitution $def_c : \Omega \cup \Delta \to \Delta^*$ as $def_c(v_i) = max_c(V_i)$, for $v_i \in \Omega$, and $def_c(R) = \{R\}$ for $R \in \Delta$.

A *$\mathcal{C}$-constrained contained partial $\mathbf{V}$-rewriting* (CCPR) of Q is a language Q' on $\Omega \cup \Delta$, such that $def_c(Q') \sqsubseteq_{\mathcal{C}} Q$. The rewriting is said to be *$\mathcal{C}$-constrained exact* if $def_c(Q') \equiv_{\mathcal{C}} Q$. As in Chapter 4 we will often drop the adjective "partial" since the non-partial rewritings are simply special sub-cases.

In order to compare different rewritings, we generalize the partial order in [GT2003a] to take constraints into account. With this partial order we want to capture the intuition that the more subwords on the Δ-alphabet that have been replaced by Ω symbols in a rewriting, the "bigger and better" the rewriting is.

Let Q_1 and Q_2 be $\mathcal{C}$-constrained contained $\mathbf{V}$-rewritings (over $\Omega \cup \Delta$) of Q. Then, Q_1 is "smaller" than Q_2, denoted $Q_1 \leq^Q_{\mathbf{V},\mathcal{C}} Q_2$, if it is possible to substitute by $v_{i_1}, \ldots, v_{i_k}$ some (not necessarily all) occurrences of words in $max_c(V_{i_1})$, ..., $max_c(V_{i_k})$ respectively, occurring as subwords in Q_1, and obtain Q_2 as a result. Also, a word in Q_1 can participate in the creation of more than one word in Q_2.

Obviously, $\leq^Q_{\mathbf{V},\mathcal{C}}$ is transitive and reflexive. Is is not antisymmetric, as for instance $\{vRR, vv, RRRR\} \leq^Q_{\mathbf{V},\mathcal{C}} \{vv, RRRR\}$, and $\{vRR, vv, RRRR\} \geq^Q_{\mathbf{V},\mathcal{C}} \{vv, RRRR\}$, when for example there is a single view $V = \{RR\}$, with v as the corresponding representative view symbol, and $\mathcal{C} = \emptyset$. However, if we define $Q_1 \equiv^Q_{\mathbf{V},\mathcal{C}} Q_2$ iff $Q_1 \leq^Q_{\mathbf{V},\mathcal{C}} Q_2$ and $Q_2 \leq^Q_{\mathbf{V},\mathcal{C}} Q_1$, we get a partial order on the equivalence classes.

Notably, we have that if a set Q' is $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal, then its equivalence class is a singleton. For this, observe that we cannot replace just any subword which is in

some word of some $max_c(V_i)$, for $i \in [1,n]$. However, a word $w = w_1 w_2 w_3$, where $w_2 \in max_c(V_i)$ and $def_c(w_1 v_i w_3) \sqsubseteq_C Q$, is not yet an "optimal" word, and we call w_2 a subword *$\mathcal{C}$-eligible for replacement*. On the other hand, we call a word on $\Omega \cup \Delta$, that has no subwords $\mathcal{C}$-eligible for replacement, a *$\mathcal{C}$-optimal* word.

Theorem 42 *Let Q' be a $\mathcal{C}$-CCPR of Q using $\mathbf{V}$. Then, Q' is $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal, if and only if, there does not exist a non $\mathcal{C}$-optimal word in Q'.*

PROOF. *(If.)* This direction is easy to see because, if there is no word in $max_c(V_i)$, for any $i \in [1,n]$, that appears as a subword $\mathcal{C}$-eligible for replacement in any of the words of Q', then it is impossible to obtain any new $\leq^Q_{\mathbf{V},\mathcal{C}}$-larger rewriting from Q'.

(Only if.) Let's suppose that there are subwords $\mathcal{C}$-eligible for replacement in the words of Q'. Then, for each word $w \in Q'$ compute a word $w_{\mathbf{V}}$ by exhaustively replacing the subwords $\mathcal{C}$-eligible for replacement in w, until nothing can be replaced anymore. If there are no subwords $\mathcal{C}$-eligible for replacement in w, then $w_{\mathbf{V}}$ equals w. Clearly, for each word w there is at least one such word $w_{\mathbf{V}}$, and the number of steps for computing it, is bounded by the length of w. Now, consider the rewriting $Q'' = \bigcup_{w \in Q'} \{w_{\mathbf{V}}\}$. For the rewriting Q'' we have that (a) $Q'' \neq Q'$, (b) $Q' \leq^Q_{\mathbf{V},\mathcal{C}} Q''$, and (c) we cannot obtain any new $\leq^Q_{\mathbf{V},\mathcal{C}}$-larger rewriting from Q''. We can see that, from (a) and (c) Q' and Q'' cannot belong to the same equivalence class, and from (b), Q'' is $\leq^Q_{\mathbf{V},\mathcal{C}}$-larger than Q'. All the above show that Q' cannot be a maximal rewriting, and this is a contradiction. ■

Corollary 10 *If Q' is $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal, then its $\equiv^Q_{\mathbf{V},\mathcal{C}}$-equivalence class is a singleton.*

PROOF. Since Q' is $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal, from the above theorem we have that, from Q', there cannot be obtained any new $\leq^Q_{\mathbf{V},\mathcal{C}}$-larger rewriting. This means in turn that the equivalence class of Q' is a singleton. ■

As discussed before a rewriting Q' is (more) useful for query optimization when it is exact. A rewriting, that is both $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal and exact, is the *union* of all

$\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal $\mathbf{V}$-rewritings of Q. We call this rewriting the *$\mathcal{C}$-constrained maximal and contained partial $\mathbf{V}$-rewriting* of Q, and denote it with $\text{CMCPR}_\mathbf{V}(Q)$.

From all the above, we can see that the CMCPR is the set of *all* the words on $\Omega \cup \Delta$ with no subword $\mathcal{C}$-eligible for replacement. Formally, we have:

Theorem 43 *The $\text{CMCPR}_\mathbf{V}(Q)$ is $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximal and $\mathcal{C}$-constrained exact.*

PROOF. The $\leq^Q_{\mathbf{V},\mathcal{C}}$-maximality follows from the fact that $\text{CMCPR}_\mathbf{V}(Q)$ is the union of the sets of words w with no subwords $\mathcal{C}$-eligible for replacement. For $\mathcal{C}$-constrained exactness, observe that by definition we have $def(\text{CMCPR}_\mathbf{V}(Q)) \sqsubseteq_\mathcal{C} Q$. On the other hand, consider a word $w \in Q$. Iterating a finite number of times (function of the length of w) we can find a word $w' \in (\Omega \cup \Delta)^*$ with the smallest possible number of Δ symbols and such that $w \in def_c(w') \sqsubseteq_\mathcal{C} Q$. Clearly, $w' \in \text{CMCPR}_\mathbf{V}(Q)$. ■

Now, let's consider again the contained rewritings without considering constraints. As mentioned before, a contained $\mathbf{V}$-rewriting of Q is any language Q' on $\Omega \cup \Delta$, such that $def(Q') \subseteq Q$. Notably, the definition of the contained $\mathbf{V}$-rewritings of Q is compliant with the definition of the $\mathcal{C}$-constrained contained $\mathbf{V}$-rewritings of Q, and the partial order $\leq^Q_\mathbf{V}$ is also compliant with the partial order $\leq^Q_{\mathbf{V},\mathcal{C}}$. Formally speaking, $\leq^Q_\mathbf{V}$ is a restriction of $\leq^Q_{\mathbf{V},\mathcal{C}}$. This is because any (algebraically contained) rewriting, by the following theorem, is also a $\mathcal{C}$-constrained rewriting.

Theorem 44 *Let Q' be a contained $\mathbf{V}$-rewriting of Q. Then, Q' is also a $\mathcal{C}$-constrained contained $\mathbf{V}$-rewriting of Q.*

Consider now the maximal and contained partial rewriting MCPR of Q. The MCPR is defined as the union of all $\leq^Q_\mathbf{V}$-maximal $\mathbf{V}$-rewritings of Q, and in simple words this means that the MCPR is the set of all "optimal" words w on $\Omega \cup \Delta$, such that $def(w) \subseteq Q$. Naturally, a word $w = w_1 w_2 w_3$, where $w_2 \in V_i$ and $def(w_1 v_i w_3) \subseteq Q$, for some $i \in [1, n]$, is not yet an "optimal" word, and we call w_2 a subword *eligible for replacement*. On the other hand, we call a word on $\Omega \cup \Delta$, that has no subwords eligible for replacement, an *optimal* word.

Since $\leq_{\mathbf{V}}^{Q}$ is a restriction of $\leq_{\mathbf{V},\mathcal{C}}^{Q}$, we have that, in general, $\text{MCPR}_{\mathbf{V}}(Q) \leq_{\mathbf{V},\mathcal{C}}^{Q} \text{CMCPR}_{\mathbf{V}}(Q)$, and this means that, in the presence of constraints, $\text{CMCPR}_{\mathbf{V}}(Q)$ is always a better (never worse) rewriting than $\text{MCPR}_{\mathbf{V}}(Q)$. Finally, when the set $\mathcal{C}$ of constraints is empty, $\text{CMCPR}_{\mathbf{V}}(Q)$ coincides with $\text{MCPR}_{\mathbf{V}}(Q)$.

We will now give a characterization for computing the rewriting $\text{CMCPR}_{\mathbf{V}}(Q)$. Namely, we show that we can compute it by a language theoretic construction. Let $\mathbf{V} = \{V_1, \ldots, V_n\}$. Then $max_c(\mathbf{V}) = \{max_c(V_1), \ldots, max_c(V_n)\}$. Consider for $max_c(\mathbf{V})$ the same alphabet Ω of view symbols.

Theorem 45 $\text{CMCPR}_{\mathbf{V}}(Q) = \text{MCPR}_{max_{\mathcal{C}}(\mathbf{V})}(max_c(Q))$.

PROOF. *($\subseteq$-direction).* Let $w \in \text{CMCPR}_{\mathbf{V}}(Q)$. For w we have that $def_c(w) \sqsubseteq_{\mathcal{C}} Q$, and since $max_c(Q)$ is the $\subseteq$-largest $\mathcal{C}$-equivalent language on Δ, we have that $def_c(w) \subseteq max_c(Q)$. This is the first condition for a word to be in a contained $max_c(\mathbf{V})$-rewriting of $max_c(Q)$. Additionally, we should show that w is optimal with respect to the sets $max_c(\mathbf{V})$ and $max_c(Q)$. Let's suppose that w is not optimal. This means in turn that w can be written as $w = w_1 w_2 w_3$, where $w_2 \in max_c(V_i)$ and $def_c(w_1 v_i w_3) \subseteq max_c(Q)$, for some $i \in [1, n]$. This fact, in other words, means that w is not $\mathcal{C}$-optimal and so, it cannot belong to $\text{CMCPR}_{\mathbf{V}}(Q)$, which is a contradiction.

($\supseteq$-direction). Let $w \in \text{MCPR}_{max_{\mathcal{C}}(\mathbf{V})}(max_c(Q))$. For w we have that $def_c(w) \subseteq max_c(Q)$, which implies that $def_c(w) \sqsubseteq_{\mathcal{C}} Q$. This is the first condition for a word to be in a $\mathcal{C}$-constrained contained $\mathbf{V}$-rewriting of Q. Additionally, we should show that w is $\mathcal{C}$-optimal. Let's suppose that w is not $\mathcal{C}$-optimal. This means in turn that w can be written as $w = w_1 w_2 w_3$, where $w_2 \in max_c(V_i)$ and $def_c(w_1 v_i w_3) \sqsubseteq_{\mathcal{C}} Q$, for some $i \in [1, n]$. Similarly as in the first part of the proof, this implies $def_c(w_1 v_i w_3) \subseteq max_c(Q)$. Finally, the last containment says that in w there are still subwords eligible for replacement, and so w cannot belong to $\text{MCPR}_{max_{\mathcal{C}}(\mathbf{V})}(max_c(Q))$, which is a contradiction. ■

Based on the above theorem and Theorem 34, we have the following corollary.

Corollary 11 *If $\mathcal{C}$ is a set of word constraints, then* $\text{CMCPR}_{\mathbf{V}}(Q) = \text{MCPR}_{anc_{\mathcal{C}}(\mathbf{V})}(anc_{\mathcal{C}}(Q))$, *where* $anc_{\mathcal{C}}(\mathbf{V}) = \{anc_{\mathcal{C}}(V_1), \ldots, anc_{\mathcal{C}}(V_n)\}$.

A set $\mathcal{C}$ of constraints is *regularity preserving* when, for any regular language L on Δ, $max_{\mathcal{C}}(L)$ is regular as well. Similarly, $\mathcal{C}$ is *context-freeness preserving* when, for any context-free language L on Δ, $max_{\mathcal{C}}(L)$ is context-free as well.

Now, a first conclusion from Theorem 45 is that, when $\mathcal{C}$ is regularity preserving, $\text{CMCPR}_{\mathbf{V}}(Q)$ can be effectively computed by the algorithm given in Chapter 4

We can raise the question about rewriting using views when the set $\mathcal{C}$ of constraints is not regularity preserving. We show that in such cases, even when $\mathcal{C}$ is context-freeness preserving, it is undecidable in general to test the existence of a "useful" contained rewriting using views. We define a rewriting as *useful* when it contains at least one word, which has at least one view symbol in it.

Theorem 46 *There exists a query Q, and a set $\mathbf{V}$ of views, such that the existence of a (constrained) useful rewriting of Q by $\mathbf{V}$ is undecidable for the class of context-freeness preserving constraints.*

PROOF. We give a reduction from the universality problem for the context free class CFG of grammars. Let $G = (\Gamma, \Delta, S, \Pi)$ be a grammar in CFG, with Γ and Δ being its sets of non-terminals and terminals respectively ($\Gamma \cap \Delta = \emptyset$), $S \in \Delta$ being the start symbol, and Π being the set of production rules

$$\{(u_i, t_i) : u_i \in \Gamma, t_i \in (\Delta \cup \Gamma)^*, \text{ for } i \in [1, m]\}.$$

Clearly, for the set $\mathcal{C} = \{t_i \sqsubseteq u_i : i \in [1, m]\}$ of word constraints on $\Delta \cup \Gamma$, we have that $L(G) = anc_{\mathcal{C}}(S) \cap \Delta^*$. Also, for any symbol $T \in \Gamma$, $anc_{\mathcal{C}}(T)$ is context-free since it is the set of all the sentential forms of a context-free sub-grammar. On the other hand, for any symbol $R \in \Delta$, $anc_{\mathcal{C}}(R) = \{R\}$. Since the context free languages are closed under union, concatenation and Kleene star, we conclude that the ancestor function $anc_{\mathcal{C}}$ applied to regular languages on $\Delta \cup \Gamma$ does not escape from the class

of context-free languages. Since by Theorem 34, $max_c = anc_c$, we have that $\mathcal{C}$ is context-freeness preserving.

Let \$ be a special symbol not in $\Delta \cup \Gamma$. We take the database alphabet to be $\Delta \cup \Gamma \cup \{\$\}$, $Q = \{\$S\$\}$, $\mathcal{C}$ as above, and a single view $V = \$\Delta^*\$$. From Corollary 11, $\text{CMCPR}(Q) = \text{MCPR}_{anc_{\mathcal{C}}(\mathbf{V})}(anc_c(Q))$, where $anc_c(\mathbf{V}) = \{anc_c(V)\}$. We observe that $anc_c(V) = anc_c(\{\$\Delta^*\$\}) = V$. So, in fact $\text{MCPR}_{anc_{\mathcal{C}}(\mathbf{V})}(anc_c(Q)) = \text{MCPR}(anc_c(Q))$. Now, because of the special symbol \$, testing if there is a useful constrained rewriting is equivalent with testing if $\text{MCPR}(anc_c(Q)) \cap \Omega \neq \emptyset$. The last can happen if and only if $V \subseteq anc_c(Q)$. This is $\$\Delta^*\$ \subseteq anc_c(\$S\$) = \$\, anc_c(S)\$$, which is equivalent to $\Delta^* \subseteq anc_c(S)$. Finally, $\Delta^* \subseteq anc_c(S)$ is equivalent to $\Delta^* \subseteq anc_c(S) \cap \Delta^*$, which is nothing else but $\Delta^* \subseteq L(G)$, i.e. $\Delta^* = L(G)$ since $L(G)$ is a pure Δ language. ■

Let us now analyze the complexity of computing the $\text{CMCPR}_{\mathbf{V}}(Q)$, when $\mathcal{C}$ is a prefix overlapping set of word constraints. As shown in Section 5.4, for any regular language L on Δ, we can polynomially compute its language of ancestors $anc_c(L)$, which is regular as well. Then, by Corollary 11, we conclude that the computation of $\text{CMCPR}_{\mathbf{V}}(Q)$ remains in the same complexity class as that of $\text{MCPR}_{\mathbf{V}}(Q)$. The lower bound can be established by the fact that in the absence of constraints, i.e. when $\mathcal{C} = \emptyset$, $\text{CMCPR}_{\mathbf{V}}(Q)$ coincides with $\text{MCPR}_{\mathbf{V}}(Q)$.

Chapter 6

Epilogue

6.1 Conway's solution for the maximally contained rewriting

The regular path queries are yet another example of the applications of the foundational theory of formal languages. Since the core of the formal language theory has been developed a long time ago, a natural question is whether there is some related work from before.

In the best of our knowledge, there is one more solution to the problem of computing the maximally contained rewriting (MCR) of [CGLV99]. This solution is from Conway [Con71], who is based on a theory of factorizations for obtaining the maximal "approximation" to a regular language. The Conway's approach is very different in nature compared to the automata-oriented solution of [CGLV99]. However, both approaches compute the same language: the maximally contained rewriting (MCR). In the following, we present the Conway's solution.

Let E and F be two languages. We say that $E \leq F$ if $E \subseteq F$. Now, let E and $F_1, \ldots, F_n$ be all regular languages. We will find a regular function f such that $f(F_1, \ldots, F_n)$ is the best "approximation" of E, in terms of $F_1, \ldots, F_n$. The

approximation is such that $f(F_1, \ldots, F_n) \leq E$, and there does not exists another function f' such that $f(F_1, \ldots, F_n) \leq f'(F_1, \ldots, F_n) \leq E$.

Definition 8

1. $F.G \ldots H \ldots J.K$ is a *subfactorization* of E if and only if

$$FG \ldots H \ldots JK \leq E. \quad (\S)$$

2. $F'.G' \ldots H' \ldots J'.K'$ *dominates* $F.G \ldots H \ldots J.K$, if $F \leq F'$, $G \leq G'$, $\ldots$, $H \leq H'$, $\ldots$, $J \leq J'$, $K \leq K'$.

3. A term H is *maximal* if it cannot be increased without violating $(\S)$.

4. A *factorization* of E is a subfactorization with every term maximal.

5. A *factor* of E is any language, which is a term in some factorization of E.

6. A *left* (*right*) factor is one, which can be the leftmost (rightmost) term in a factorization.

It is easy to verify the following theorem.

Theorem 47 *Any left (right) factor is the left (right) factor in some 2-term factorization. Any factor is the central term in some 3-term factorization.*

From the above theorem we can easily prove:

Theorem 48 *The condition that $M.N$ be a factorization of E defines a $1-1$ correspondence between the left and right factors.*

PROOF. The previous theorem shows that each left factor M corresponds to at least one right factor N, and vice versa - but for given M the union of all possible N's is the only possible maximal N. ■

Now, we index the left and right factors as M_i, N_i so that $M_i.N_i$ is a factorization, and define E_{ij} by the condition that $M_i.E_{ij}.N_i$ be a subfactorization of E in which E_{ij} is maximal.

Note. Conway did not give a way to compute the E_{ij} given M_i and N_i. However, we can compute E_{ij} by using our algorithm for the maximal and contained partial rewriting (MCPR). For this, let # be a marker symbol, not in the alphabet of the languages. We compute the $\text{MCPR}_{\{\{\#\}.M_i,\ N_i.\{\#\}\}}(\{\#\}E\{\#\})$, by using as "view" symbols, say m_i and n_i. Clearly, because of the marker #, the symbols m_i and n_i will appear respectively only on the left and the right of the words of the rewriting. It is easy to build now an erasing transducer, which erases m_i and n_i. Finally, if we transduce the above rewriting we get the E_{ij}.

Theorem 49 *Each E_{ij} is a factor of E, and each factor of E is one of the E_{ij}. There exist unique indices l, r such that $E = E_{lr}$, and $M_i = E_{li}$, $N_i = E_{ir}$ for each i.*

PROOF. E_{ij} is maximal in $M_i.E_{ij}.N_i$. From this, it is easy to see that E_{ij} is a factor. Each factor H is the central term in a 3-term factorization M_iHN_j. Since this is a factorization, H is maximal. So, $H = E_{ij}$.

The term E is maximal in the subfactorization $\{\epsilon\}.E$, dominated by $M_l.E$, say, whence $E = N_l$ and $M_l \geq \{\epsilon\}$. Now for any i, $M_l.M_i.N_i$ is a subfactorization with M_i, N_i maximal, and so $M_i = E_{li}$. Defining r similarly by the condition that $E.N_r$ be a factorization, we find that $M_lE \leq E$ and $EN_r \leq E$, so M_lEN_r is a subfactorization with E maximal, whence $E = E_{lr}$. ∎

Hence, the factors naturally form a square matrix among the entries of which is E. Now the question is how we can compute the factors of a language E. The proof of the following theorem is constructive and gives a way to compute the right factors of a language E if it is regular. Symmetrically, we can compute the left factors, if we consider the mirror language of E, that we get if we write the words of E in reverse order.

Theorem 50 *A regular language E has finitely many factors.*

PROOF. The right factors N of E are just the maximal events with $M.N \leq E$, M ranging over all events. But, $M.N \leq E$ if and only if $\{w\}.N \leq E$ for each $w \in M$; that is to say, if and only if $N \leq \cap_{w \in M} E_w$, where E_w is the *left word derivative* of E with respect to w, $\{v : wv \in E\}$. Since N is maximal, we have instead equality $N = \cap_{w \in M} E_w$. Recall that $w \in M$, and M could be any language. Hence, we can obtain the right factors by taking all the possible combinations of intersections of the left word derivatives. It is part of the folklore that the left word derivatives are finitely many, and are the languages accepted by the DFA's that we get by repetitively changing the initial state of a DFA for E. ■

After we compute the left and right factors, we can use the reasoning in the proof of Theorem 48, do find out their 1-1 correspondence. Then as explained before (in the note), we can compute the E_{ij} factors of E. Hence the factor matrix, denoted with $|E|$, is indeed constructible. We will show the following lemma.

Lemma 7

1. *$AB \leq E_{ik}$ if and only if $A \leq E_{ij}$, $B \leq E_{jk}$ for some j.*
2. *$F_a F_b \ldots F_k$ if and only if there are indices l, m, n, $\ldots$, q, r for which $F_a \leq E_{lm}$, $F_b \leq E_{mn}$, $\ldots$, $F_k \leq E_{qr}$ (l, r as usual).*

PROOF. If $AB \leq E_{ik}$, then $M_i A.BN_k \leq E$, and so $M_i A \leq M_j$, $BN_k \leq N_j$ for some j, so that $M_i A N_j \leq E$ and $M_j B N_k \leq E$, whence $A \leq E_{ij}$, $B \leq E_{jk}$, which proves the first part of the lemma. The second part can be inductively proved from the first part. ■

The matrix $|E|$ is a "Kleene matrix." Formally, a *Kleene matrix* $\mathcal{M}$ is a square matrix whose entries $\mathcal{M}[i, j]$ are languages. We can visualize such a (square) matrix by building a "macro" automaton $\mathcal{A}_{\mathcal{M}}$, whose states are labeled by the indices i of the matrix rows (and columns), and whose "macro" transitions are labeled with the languages $\mathcal{M}[i, j]$ if going from state i to j. We define the languages L_{ij}^k as the set of all words taking $\mathcal{A}_{\mathcal{M}}$ from state i to j passing exactly k "macro" transitions. It is

easy to see that $L_{ij}^k = \mathcal{M}^k[i,j]$, where the matrix multiplication is with respect to the language concatenation (".") and union ("+"). Now, let L_{ij} be the set of all words taking $\mathcal{A}_\mathcal{M}$ from the state i to j (regardless of how many "macro" transitions are passed). Clearly $L_{ij} = \mathcal{M}^*[i,j]$, where $\mathcal{M}$ denotes the matrix $\mathcal{M} + \mathcal{M}^2 + \ldots$. Note that we can effectively construct $\mathcal{M}^*$. For this, we compute the languages between any two states of $\mathcal{A}_\mathcal{M}$ by using the standard Kleene algorithm [HU79]. Then we build a new Kleene matrix of the same dimensions as $\mathcal{M}$, where the (i,j) entries are the languages between the i and j states in $\mathcal{A}_\mathcal{M}$. This matrix is $\mathcal{M}^*$.

Let's return now to the factor matrix $|E|$. We can "approximate" it in terms of $F_1, \ldots, F_n$ by "approximating" its entries individually. Let's become more specific on what the "approximation" means in that context. First we consider the special symbols $f_1, \ldots, f_n$[1] for representing the languages $F_1, \ldots, F_n$. For an entry E_{ij} we define the *best linear approximation* as the sum of all the symbols f_t for which $F_t \leq E_{ij}$. Also, we define the *best constant approximation* as the ϵ or $\emptyset$ if $\epsilon \in E_{ij}$ or $\epsilon \notin E_{ij}$ respectively. Naturally, the *best approximation* is the language of all the words $f_a f_b \ldots f_k$ such that $F_a F_b \ldots F_k \leq E_{ij}$. Now, we are ready to state the main theorem.

Theorem 51 *Let $|E|_F$, $|E|_C$, $|E|_L$ denote respectively the best, best constant, best linear approximation matrices for $|E|$ by $F_1, \ldots, F_n$. Then*

$$|E|_F = |E|_C + |E|_L^*.$$

PROOF. The non-empty word $f_a f_b \ldots f_k$ belongs to $|E|_F[i,j]$ if and only if F_a, F_b, $\ldots F_k$ satisfy the conditions of the Lemma 7 (second part), which are precisely the conditions under which it belongs to $|E|_L^*$. The empty word belongs to $|E|_F$ if and only if $|E|_C[i,j] = \{\epsilon\}$. ∎

The above theorem embodies an algorithm for computing the best approximating function. We first compute the factor matrix $|E|$, then replace every entry E_{ij} by the sum of all f_t for which $F_t \leq E_{ij}$, together with ϵ if $\epsilon \in E_{ij}$, and then take the (l,r) entry in the star of the resulting matrix.

[1]These special symbols have the same role as the symbols of the Ω alphabet in the previous chapters.

6.2 More general path queries

In this section we will demonstrate that our data and query framework can be easily adapted to capture some additional aspects not explicitly in the model [AV99]. For example, some languages with path expressions (such as Lorel [AQM+97]) view labels as strings of characters and use regular expressions that work at two levels of granularity: the label (viewed as a string of characters) and the path (viewed as a sequence of labels). For instance, consider the following general path expression:

"doc"("[sS]ections?" "text"+"[pP]aragraph")

which specifies a path starting with an edge labeled *doc*, either followed by an edge labeled *section(s)* (possibly starting with a capital *S*) and a *text*-edge, or followed by an edge labeled *paragraph* (possibly with a capital *P*).

We used here a syntax based on grep E-regular expressions for string patterns, and quotes to separate labels (strings of characters) from paths (sequences of labels).

Call such queries *general path queries*. We claim that these can essentially be captured by our framework, modulo some preprocessing of labels. Let Q be a general path query and let Π be the set of string patterns occurring in Q. We will reduce the problem of the evaluation of the general path query Q on a database DB with possibly infinitely many labels, to the problem of the evaluation of a regular path query $\mu(Q)$ on a database $\mu(DB)$ with finitely many labels. To do this, consider the equivalence relation on strings defined by: $w \equiv w'$ if w and w' satisfy precisely the same patterns in Π. For each equivalence class $[w]$, choose a particular label say $l([w])$ in $[w]$ and let Δ be the set of such labels. Observe that Δ is finite. Now μ is defined as

1. for each label/string w, $\mu(w) = l([w])$ and $\mu(a) = a$ for each node a in DB; this defines $\mu(DB)$.

2. for each string pattern s occurring in Q, let $\mu(s) = l([w_1]) + \ldots + l([w_k])$, where $[w_l], \ldots, [w_k]$ are the equivalence classes of words satisfying s; this defines $\mu(Q)$.

The construction is illustrated next. Consider the general path expression:

$$Q = (\text{“}R^*S\text{”}\ \text{“}SR^*\text{”}) + (\text{“}R^*S\text{”}\ \text{“}T\text{”}) + (\text{“}SR^*\text{”}\ \text{“}T\text{”}) + (\text{“}UU^*\text{”})^+.$$

One can find six equivalence classes:

1. $[S] = R^*S \cap SR^* = S$,
2. $[RS] = R^*S - SR^* = RR^*S$,
3. $[SR] = SR^* - R^*S = SRR$,
4. $[T] = T$,
5. $[U] = UU^*$,
6. $[V] = \Delta^* - ([S] \cup [RS] \cup [SR] \cup [T] \cup [U])$.

Let “S,” “RS,” “SR,” “T,” “U,” “V” be the labels representing the equivalence classes. The query $\mu(Q)$ is

$$\mu(Q) = ((\text{“}S\text{”}+\text{“}RS\text{”})(\text{“}S\text{”}+\text{“}SR\text{”}))+((\text{“}S\text{”}+\text{“}RS\text{”})\text{“}T\text{”}))+((\text{“}S\text{”}+\text{“}SR\text{”})\text{“}T\text{”})+(\text{“}U\text{”})^+.$$

Clearly, a tuple (pair) is in $ans(Q, DB)$, if and only if it is in $ans(\mu(Q), \mu(DB))$.

This allows us to reduce the problem of the evaluation of a general path query involving potentially infinitely many labels to the evaluation of a regular path query on a finite alphabet of labels, via preprocessing of labels.

We want to emphasize here that in the case of the query rewriting using views, which are all expressed as general path queries, we apply exactly the above preprocessing of labels considering in this case *all* the string patterns in the general path expressions of the query and the views. Clearly, after such a preprocessing of the labels we can apply the algorithms of the previous chapters for computing rewritings using views.

6.3 Semistructured data: nodes versus edge labels

Semistructured data is a bare-bones abstraction of irregular, self-describing data found across the Web. It is also motivated by applications such as scientific databases, and integration of heterogeneous data. Several variants of the semi-structured data model have been proposed, with minor differences in formalism. Some of the variants consider the data being organized in graphs where only the edges are labeled, some other consider only the nodes being labeled, while another more general variant considers both the edges and the nodes being labeled. The last variant is more practical for the Web data, especially for networks of linked Web pages, which have also *content*, in the form of text. A node a with "content" w (where w is a word) *can be modeled in our context* by having an edge labeled "content=w" outgoing from a and pointing to a itself. Now content-based selections can be specified using the general path expressions just discussed in Section 6.2. For instance, the reachable nodes that contain the word "SGML" can be retrieved using the general path query ("(.)*")*"content=(.)*SGML(.)*", where "(.)*" indicates some arbitrary sequence of characters.

6.4 Semistructured data: graphs versus trees

A question arises: If our databases were trees instead of general graphs, then does this restriction helps in reasoning about query rewriting using views? This question is related to the fact most of the XML documents are modeled by trees and not by general graphs. Our answer in the above question is negative: The query rewriting using views does not become any easier by considering tree target databases instead of general graphs. We give this answer based on the following observations. Firstly, all the rewritings (except PR) that we presented are based on our ability to reason about the query containment. Deciding or ensuring containment is their most expensive part. We argue here that the query containment does not become any easier if we restrict ourselves to the tree databases only. For this, consider as an example the containment of two queries $Q_1 \sqsubseteq Q_2$. Take now the family $\mathcal{F}_{lin}$ of all the linear databases on the

Δ alphabet, i.e. for any word $w \in \Delta^*$ ($w = R_1 \dots R_k$) construct a database DB_w, by introducing $k+1$ nodes, $a_1, \dots, a_{k+1}$, and the edges $(a_1, R_1, a_2), \dots, (a_k, R_k, a_{k+1})$. Clearly, a linear database is a special case of a tree database. It is easy to see that by considering the family $\mathcal{F}_{lin}$ of databases, if $ans(Q_1, DB) \subseteq ans(Q_2, DB)$ for each $DB \in \mathcal{F}_{lin}$, then $Q_1 \subseteq Q_2$ as formal languages. Hence, at the end we conclude that even being restricted to linear databases only, we still need to decide "pure" formal language containment and nothing less than that.

6.5 Regular path queries versus XPATH

The most popular incarnation of regular path queries is in the XPATH query language that is mainly used to elegantly access different parts of an XML document. As a typical example consider the following XPATH query

/publications//journal/article[author/name="Emil Bouret"]/ref/software/category,

where the "/" semantically stands for concatenation, "//" for Δ^*, and [author/name= "Emil Bouret"] is a condition with a regular expression which is to be evaluated starting from the nodes, that we reach from a "root" node by following a path spelling a word in "/publications//journal/article."

The above query exemplifies almost all the constructs with regular expressions that we can meet in XPATH. Suppose now that we have evaluated the above query on the database, and are interested in what can be cached for optimizing future queries. Observe that we can break the above XPATH query into three parts:

1. Q_1 = "/publications//journal/article",
2. Q_2 = "author/name", and
3. Q_3 = "/ref/software/category".

Reasoning similarly as in Section 4.4, we can say that Q_1 is locally complete, with respect to the node from where we started evaluating it (we start with the root of the

document in this case), Q_2 is not locally complete unless there is no article authored by "Emil Bouret" (because in such a case the "cut" like constructs could not be used), and finally Q_3 is indeed locally complete with respect to all the nodes from where we started evaluating it.

Clearly, we should cache the answers of the locally complete parts of the XPATH queries. In our example we cache the answer to Q_1 and Q_3. As mentioned above some partial answer to Q_2 could also be cached if it happens, as explained above, to be locally complete. [2] After a few cachings we will have a useful view graph, which we could use to optimize the future XPATH queries.

Let $\mathbf{V} = \{V_1, \ldots, V_n\}$ be the regular languages of those locally complete parts of the previously answered XPATH queries. Let $\mathcal{V}$ be the associated cached view graph. Now, consider an XPATH query with regular expressions $Q_1, \ldots, Q_k$. We rewrite each one of them into $Q'_1, \ldots, Q'_k$, where Q'_i, for $i \in [1, k]$, is an exact partial rewriting of Q_i using $\mathbf{V}$. Then, we evaluate on $\mathcal{V}$ and DB, the new XPATH query, which is obtained by replacing Q_i with Q'_i, for $i \in [1, k]$, in the initial XPATH query. When we need to evaluate Q'_i, for some $i \in [1, k]$, starting from a node, say a, we use exactly the same algorithm as in Section 4.4.

6.6 Conclusions

We have throroughly studied the rewriting of regular path queries using views. We study the problem in two realistic settings. The first is the optimization of the information integration from different data sources represented as views, and the second is the query optimization using a set of cached views. Our optimizations are in fact minimizations of the expensive direct data accesses. As such they are guarantied to be always better than direct access methods.

Our contributions are sumarized as follows.

1. We reason about two different kinds of rewritings: the *relevance* and *contained*

[2] See also Section 4.4 for a more detailed analysis of such cases.

rewritings. The contained rewritings are also relevant rewritings, but they have the desired property to be contained in the original query after the substitution of the view symbols (or names) by the corresponding view languages. This property makes the contained rewritings ideal choices for query optimization, because by evaluating them we do not get false tuples (pairs). On the other hand the relevance rewritings are in general bigger languages, and if the path history of the view graph computation has been cached as well, then the relevance rewritings become very useful in query optimization.

2. We present a polynomial time algorithm for obtaining all the certain answers in a LAV data integration system, under the most realistic assumption, where the data sources are modeled as sound views over a global schema. We achieve the polynomial time by computing the most relevant (or possibility) rewriting (PR). Although we can get some more tuples, not in the certain answer, those tuples are still useful in the sense that they are *possible*, i.e. there exist databases, which are consistent with the views and on which the query evaluation produces the afore mentioned tuples.

3. We introduce a general framework for comparing rewritings. We define two partial orders, one for the relevance rewritings and another for the contained rewritings. Then, we compute the best known (partial) rewritings according to the corresponding orders. They are the *exhaustive possibility partial rewriting* (EPPR) and the *maximal and contained partial rewriting* (MCPR).

4. We present algorithms which by making use of the rewritings minimize as much as possible the direct data accesses. The contained rewritings are the choice when there is no cached path history. On the other hand, the relevance rewriting are more profitable should we have cached the path histories.

5. We are the first to present a "cache-and-optimize" algorithm for conjunctive regular path queries (CPRQ) using view based rewritings. Our opinion is that the rewriting of the CPRQ's should be approached by rewriting each atom in particular instead of rewriting the whole query at once. Notably, we show that the ideas for optimizing CPRQ's using views, could be naturally extended to the optimization of the XPATH queries, which have become so popular nowadays for querying XML databases.

6. We introduce the *general path constraints*, which can capture a lot of knowledge about the database(s) we are going to navigate with regular path queries. Our constraints can be detected either manually by browsing or by automated agents and database spiders (like Web spiders).

7. We explore the inherent relationship between word constraints and semi-Thue rewrite systems. This relationship is particularly important because it provides the bridge between the world of databases and the world of the formal languages.

8. The relationship between word constraints and semi-Thue rewrite systems tells us that the word query containment under word constraints is decidable whenever the corresponding class of semi-Thue systems has a decidable word problem. Notably, there are a lot of useful classes of semi-Thue systems with decidable word problem. Now, the question is whether the (general) query containment is decidable for a class of word constraints with decidable word query containment. Our answer is negative based on a reduction from the universality problem for context-free grammars, which is known to be undecidable.

9. We present a wide class of word constraints under which the (general) query containment is decidable. We call this class "prefix overlapping" to convey the type of the allowed explicit overlapping between the left and right hand sides of the constraints. However, other implicit overlappings are allowed, or no overlappings at all, making our class capable of capturing almost all the constraints in real world examples.

10. We are the first to reason about the query rewriting under constraints. By considering constraints we are to obtain bigger and better rewritings through which we can optimize the queries even more. Notably, we transform the query rewriting under constraints into a pure language theoretic construct.

11. We discuss related work from the formal language research comunity about rewriting using views (Conway's solution for the MCR). We also discuss the rewritings when the target databases are trees versus graphs, and when there are node contents (labels) as well. We argue that considering tree databases does not make the rewriting problem any easier, while a database model having

node labels can be easily transformed into our initial database model via a preprocessing of the labels.

Closing, we present the containment relationship between the relevance and contained rewritings, a Hasse diagram (with respect to the partial order $\leq_{\mathbf{V}}^{Q}$, Figure 21) and a sumarizing table (Table 2) with the optimality (with respect to the partial orders $\leq_{\mathbf{V}}$ and $\leq_{\mathbf{V}}^{Q}$), exactness (with respect to the query) and the computational complexity (with respect to query and view expression lengths) of the rewritings.

$$\text{PR} \subseteq \text{PPR}$$

$$\text{MCR} \subseteq \text{ECPR} \subseteq \text{MCPR} \subseteq \text{GCPR}$$

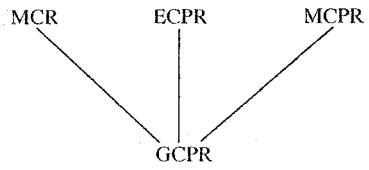

Figure 21: Hasse diagram for the contained rewritings

	$\leq_{\mathbf{V}}$ or $\leq_{\mathbf{V}}^{Q}$ -maximality	Exactness	Complexity
PR [GT2000]	YES	NO	PTIME
PPR [GT2001a]	YES	NO	EXPTIME
MCR [CGLV99]	YES	NO	2EXPTIME
GCPR [CGLV99]	NO	YES	2EXPTIME
ECPR [GT2001a]	YES	NO	2EXPTIME
MCPR [GT2003a]	YES	YES	3EXPTIME

Table 2: Maximality, exactness and complexity of the rewritings

Future work. As an extension of our work we see the introduction of weighted regular path queries and weighted database graphs. The weights (over closed semirings) can represent the cost of traversing a link to access a Web page, the link (page) reputation, the price we want to pay for traversing specific links and many other quantitative characteristics about the way, in which the user wants the regular path

queries to be evaluated. We are working toward optimal algorithms for query answering, query containment and query rewriting using views in this "weighted world" setting.

Bibliography

[Abi97] S. Abiteboul. Querying Semistructured Data. *Proc. of the 6th Int'l Conference on Database Theory* 1997 pp. 1-18.

[AD98] S. Abiteboul, O. M. Duschka. Complexity of Answering Queries Using Materialized Views. *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1998 pp. 254-263

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

[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'l Journal on Digital Libraries* 1997 1(1) pp. 68-88.

[AV99] S. Abiteboul, V. Vianu. Regular Path Queries with Constraints. *Journal of Computing and System Sciences* 58(3) 1999, pp. 428-452

[BN81] L. Boasson, M. Nivat. Centers of Languages. *Proc. of Theoretical Computer Science, 5th GI-Conference* 1981, LNCS 104, pp. 245-251.

[BO93] R. Book, F. Otto *String Rewriting Systems* Springer Verlag, 1993.

[Bun97] P. Buneman. Semistructured Data. *Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1997, pp. 117-121.

[BDFS97] P. Buneman, S. B. Davidson, M. F. Fernandez and D. Suciu. Adding Structure to Unstructured Data. *Proc. of the 6th Int'l Conference on Database Theory* 1997, pp. 336-350.

[BFW98] P. Buneman. W. Fan, S. Weinstein. Path Constraints in Semistructured and Structured Databases. *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1998, pp. 129-138.

[BFW99] P. Buneman. W. Fan, S. Weinstein. Query Optimization for Semistructured Data Using Path Constraints in a Deterministic Data Model. *Proc. Int'l Workshop on Database Programming Languages* 1999, pp. 208-223.

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

[BL80] J. A. Brzozowski and E. L. Leiss. On Equations for Regular Languages, Finite Automata, and Sequential Networks. *Theoretical Computer Science* 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 the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1999, pp. 194-204.

[CGLV2000a] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of the 16th Int'l Conference on Data Engineering* 2000, pp. 389-398.

[CGLV2000b] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing for Regular Path Queries with Inverse. *Proc. of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 2000, pp. 58-66.

[CGLV2000c] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. What is View-Based Query Rewriting? *Proc. of the 7th International Workshop on Knowledge Representation meets Databases* 2000, pp. 17-27.

[CGLV2002] D. Calvanese, G. Giacomo, M. Lenzerini and M. Y. Vardi. Lossless Regular Views. *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 2002, pp. 247-258.

[Ca2001] D. Caucal. On the Transition Graphs of Turing Machines. *Proc. of Machines, Computations, and Universality, Third Int'l Conference.* 2001, pp. 177-189

[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.

[DT2001] A. Deutsch, V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. *Proc. of Int'l Workshop on Database Programming Languages* 2001, pp. 21-39.

[DG97] O. Duschka and M. R. Genesereth. Answering Recursive Queries Using Views. *Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1997, pp. 109-116.

[FS98] M. F. Fernandez and D. Suciu. Optimizing Regular path Expressions Using Graph Schemas *Proc. of the 14th International Conference on Data Engineering* 1998, pp. 14-23.

[FLS98] D. Florescu, A. Y. Levy, D. Suciu Query Containment for Conjunctive Queries with Regular Expressions *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1998, pp. 139-148.

[GW97] R. Goldman and J. Widom. DataGuides: Enabling query Formulation and Optimization in Semistructured Databases. *Proc. of the 23rd International Conference on Very Large Data Bases* 1997 pp. 436-445.

[GM99] G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. *Proc. of 7th Int'l Conference on Database Theory* 1999 pp. 332-347.

[GT2000] G. Grahne and A. Thomo. An Optimization Technique for Answering Regular Path Queries. *Proc. of Web and Databases* 2000 pp. 99-104, and Lecture Notes in Computer Science 1997 Springer 2001 pp. 215-225.

[GT2001a] G. Grahne and A. Thomo. Algebraic Rewritings for Optimizing Regular Path Queries. *Proc. of the 8th Int'l Conference on Database Theory* 2001, Lecture Notes in Computer Science 1973 Springer 2001 pp. 301-315.

[GT2001b] G. Grahne and A. Thomo. Approximate Reasoning in Semistructured Data. *Proc. of the 8th International Workshop on Knowledge Representation meets Databases* 2001.

[GT2003a] G. Grahne and A. Thomo. New Rewritings and Optimizations for Regular Path Queries. *Proc. of the 9th Int'l Conference on Database Theory*, Lecture Notes in Computer Science 2572 Springer 2002 pp. 242-258.

[GT2003b] G. Grahne and A. Thomo. Query Containment and Rewriting Using Views for Regular Path Queries under Constraints *Proc. of the 22d ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 2003 pp. 111-122.

[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.

[Lev2000] A. Y. Levy. *Answering queries using views: a survey.* Technical Report, Computer Science Dept., Washington Univ., 2000.

[LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *Proc. of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* 1995, pp. 95-104.

[MW95] A. O. Mendelzon and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing* 24:6, 1995.

[MMM97] A. O. Mendelzon, G. A. Mihaila and T. Milo. Querying the World Wide Web. *Int'l Journal on Digital Libraries* 1(1), 1997 pp. 54-67.

[MS99] T. Milo and D. Suciu. Index Structures for Path Expressions. *Proc. of the 7th Int'l Conference on Database Theory*, 1999, pp. 277-295.

[NUWC97] S. Nestorov, J. D. Ullman, J. L. Wiener, S. S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchical Data. *Proc. of the 13th International Conference on Data Engineering*, 1997, pp. 79-90.

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

[Sen90] G. Senizergues. Some Decision Problems about Controlled Rewriting Systems. *Theoretical Computer Science* 71(3), 1990, pp. 281-346

[Sip96] M. Sipser. *Introduction to the Theory of Computation* PWS Pub. Co., 1996.

[Tho2000] A. Thomo. *Query Processing Using Views in Semistructured Databases* Master Thesis, Concordia University Pub., 2000.

[Ull97] J. D. Ullman. Information Integration Using Logical Views. *Proc. of the 6th Int'l Conference on Database Theory* 1997, pp. 19-40.

[Var88] M. Y. Vardi. The universal-relation model for logical independence. *IEEE Software* 5(2), 1988, pp. 80-85.

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