To appear in **ICDT'99 WS on Query Processing for SSD and Non-standard Data Formats**

Rewriting Regular Expressions in Semi-Structured Data

D. Calvanese[1], G. De Giacomo[1], M. Lenzerini[1], M. Vardi[2]

[1] *Dip. di Informatica e Sistemistica*
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

[2] *Dept. of Computer Science*
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A
vardi@cs.rice.edu

Abstract

In this paper we address the problem of query rewriting in the context of semi-structured data. We present a method for computing the rewriting of a regular expression E in terms of other regular expressions. The method computes the exact rewriting (the one that defines the same regular language as E) if it exists, or the rewriting that defines the maximal language contained in the one defined by E, othwerwise. We present a complexity analysis of both the problem and the method, showing the latter is essentially optimal. Finally, we illustrate how to exploit the above mentioned method in order to devise an algorithm for rewriting regular path queries for semi-structured data using views. The complexity results established for the rewriting of regular expressions apply also to the case of regular path queries.

1 Introduction

The basic querying mechanism over semi-structured data is the one that retrieves all pairs of nodes of the graph representing the database that are connected by a path conforming to a given pattern. Since a user may ignore the precise structure of the graph, the mechanism for specifying path patterns should be flexible enough to allow for expressing regular path queries, i.e. queries that provide the specification of the requested paths through a regular language [AQM+97, BDHS96, FFK+98]. For example, the regular path query $(\_^* \cdot (\mathit{rome} + \mathit{jerusalem}) \cdot \_^* \cdot \mathit{restaurant})$ specifies all the paths having at some point an edge labeled *rome* or *jerusalem*, followed by any number of other edges and by an edge labeled with a restaurant.

In semi-structured data, as well as in data integration, data warehousing, and query optimization, the problem of query rewriting using views is receiving much attention [Ull97, AD98]: Given a query Q and k queries $Q_1, \ldots, Q_k$ associated to the symbols $q_1, \ldots, q_k$, respectively, generate a new query Q' over the alphabet $q_1, \ldots, q_k$ such that, first interpreting each q_i as the result of Q_i, and then evaluating Q' on the basis of such interpretation, yields the same result as evaluating Q. Several papers investigates this problem for the case of conjunctive queries (with or without arithmetic comparisons) [LMSS95, RSU95], queries with aggregates [SDJL96, CNS98], recursive queries [DG97], and queries expressed in Description Logics [BLR97]. Rewriting techniques for query optimization are described, for example, in [CKPS95, ACPS96, TSI96], and in [FS98, MS99] for the case of path queries in semi-structured data.

None of the above papers provides a method for rewriting regular path queries. Observe that computing the of rewriting regular expressions is a special case of the above problem, and is still an open problem.

In this paper we present the following contributions:

- We describe a method for computing the rewriting of a regular expression E_0 in terms of other regular expressions. The method computes the exact rewriting (the one that defines

the same regular language as E_0) if it exists, or the rewriting that defines the maximal language contained in the one defined by E_0, otherwise.

- We provide a complexity analysis of the problem of rewriting regular expressions. We show that our method computes the rewriting in 2EXPTIME, and is able to check whether the computed rewriting is exact in 2EXPSPACE. We also show that the problem of checking whether there is a nonempty rewriting is EXPSPACE-complete, and demonstrate that our method for computing the rewriting is essentially optimal. Finally, we show that the problem of verifying the existence of an exact rewriting is 2EXPSPACE-complete.
- We illustrate how to exploit the above mentioned method in order to devise an algorithm for the rewriting of regular path queries for semi-structured databases. The complexity results established for the rewriting of regular expressions apply to the new algorithm as well. Also, we show how to adapt the method in order to compute rewritings with specific properties. In particular, we consider partial rewritings (which are rewritings that, besides $E_1, \ldots, E_k$, may use also symbols in E_0), in the case where an exact one does not exist.

This paper is a shorter version of [CDGLV99], to which we refer for more details.

2 Rewriting of regular expressions

We present a technique for the following problem: Given a regular expression E_0 and a (finite) set $\mathcal{E} = \{E_1, \ldots, E_k\}$ of regular expressions over an alphabet Σ, re-express, if possible, E_0 by a suitable combination of $E_1, \ldots, E_k$.

We assume that associated to $\mathcal{E}$ we always have an alphabet $\Sigma_{\mathcal{E}}$ containing exactly one symbol for each regular expression in $\mathcal{E}$, and we denote the regular expression associated to the symbol $e \in \Sigma_{\mathcal{E}}$ with $re(e)$. Given any language ℓ over $\Sigma_{\mathcal{E}}$, we denote by $expand_{\Sigma}(\ell)$ the language over Σ defined as follows

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

where $L(e)$ is the language defined by the regular expression e.

Definition 1 Let E_0 be a regular expression over Σ, and $\mathcal{E} = \{E_1, \ldots, E_k\}$ a set of regular expressions over Σ. Let R be any formalism for defining a language $L(R)$ over $\Sigma_{\mathcal{E}}$. We say that R is a *rewriting of* E_0 wrt $\mathcal{E}$ if $expand_{\Sigma}(L(R)) \subseteq L(E_0)$. ∎

We are interested in maximal rewritings, i.e. rewritings that capture in the best possible way the language defined by the original regular expression E_0.

Definition 2 A rewriting R of E_0 wrt $\mathcal{E}$ is Σ-*maximal* if for each rewriting R' of E_0 wrt $\mathcal{E}$ we have that $expand_{\Sigma}(L(R')) \subseteq expand_{\Sigma}(L(R))$. A rewriting R of E_0 wrt $\mathcal{E}$ is $\Sigma_{\mathcal{E}}$-*maximal* if for each rewriting R' of E_0 wrt $\mathcal{E}$ we have that $L(R') \subseteq L(R)$. ∎

Intuitively, when considering Σ-maximal rewritings we look at the languages obtained after substituting each symbol in the rewriting by the corresponding regular expression over Σ, whereas when considering $\Sigma_{\mathcal{E}}$-maximal rewritings we look at the languages over $\Sigma_{\mathcal{E}}$. Observe that by definition all Σ-maximal rewritings define the same language (similarly for $\Sigma_{\mathcal{E}}$-maximal rewritings).

Theorem 3 *Let R be a rewriting of E_0 wrt $\mathcal{E}$. If R is $\Sigma_{\mathcal{E}}$-maximal then it is also Σ-maximal.*

Given E_0 and $\mathcal{E}$, we are interested in deriving the maximal rewriting of E_0 wrt $\mathcal{E}$, i.e., the rewriting that defines the maximal language contained in the one defined by E_0. The algorithm we propose takes E_0 and $\mathcal{E}$ as input, and returns an automaton $R_{\mathcal{E},E_0}$ built as follows:

1. Construct a deterministic automaton $A_d = (\Sigma, S, s_0, \rho, F)$ such that $L(A_d) = L(E_0)$.
2. Define the automaton $A' = (\Sigma_{\mathcal{E}}, S, s_0, \rho', S - F)$, where $s_j \in \rho'(s_i, e)$ iff $\exists w \in L(re(e))$ such that $s_j \in \rho^*(s_i, w)$.
3. $R_{\mathcal{E},E_0} = \overline{A'}$, i.e. the complement of A'.

Theorem 4 *The automaton $R_{\mathcal{E},E_0}$ is a $\Sigma_{\mathcal{E}}$-maximal rewriting of E_0 wrt $\mathcal{E}$.*

Notably, although we do not constrain in any way the form of the rewritings, Theorem 4 shows that the language defined by the maximal rewritings is in fact regular (indeed, $\overline{A'}$ is a finite automaton).

To verify whether the $R_{\mathcal{E},E_0}$ is an exact rewriting of E_0 wrt $\mathcal{E}$ we proceed as follows:

1. We construct an automaton $B = (\Sigma, S_B, s_{B0}, \rho_B, F_B)$ that accepts $expand_{\Sigma}(L(R_{\mathcal{E},E_0}))$, by replacing each edge labeled by e_i in $R_{\mathcal{E},E_0}$ by an automaton A_i such that $L(A_i) = L(re(e_i))$ for $i = 1, \ldots, k$. (Each edge labeled by e_i is replaced by a fresh copy of A_i. We assume, without loss of generality, that A_i has unique start state and accepting state, which are identified with the source and target of the edge, respectively.) Observe that, since $R_{\mathcal{E},E_0}$ is a rewriting of E_0, $L(B) \subseteq L(A_d)$.
2. We check whether $L(A_d) \subseteq L(B)$, that is, we check whether $L(A_d \cap \overline{B}) = \emptyset$.

Theorem 5 *The automaton $R_{\mathcal{E},E_0}$ is an exact rewriting of E_0 wrt $\mathcal{E}$ iff $L(A_d \cap \overline{B}) = \emptyset$.*

We analyze now the computational complexity of both the problem of rewriting regular expressions, and the method described above. By considering the cost of the various steps in computing $R_{\mathcal{E},E_0}$, we immediately derive the following upper bounds.

Theorem 6 *The problem of generating the $\Sigma_{\mathcal{E}}$-maximal rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ of regular expressions is in 2EXPTIME.*

Theorem 7 *The problem of verifying the existence of an exact rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ of regular expressions is in 2EXPSPACE.*

The established upper bounds are essentially optimal, as shown by the following theorems.

Theorem 8 *The problem of checking whether there is a nonempty rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ of regular expressions is EXPSPACE-complete.*

Theorem 9 *For each $n > 0$ there is a regular expression E_0^n and a set $\mathcal{E}^n$ of regular expressions such that the combined size of E_0^n and $\mathcal{E}^n$ is polynomial in n, but the shortest nonempty rewriting (expressed as either a regular expression or an automaton) of E_0^n wrt $\mathcal{E}^n$ is of length 2^{2^n}.*

Note that Theorem 8 already implies that the upper bound established in Theorem 6 is essentially optimal. If we can generate maximal rewritings in, say, EXPTIME, then we could test emptiness in PSPACE, which is impossible by Theorem 8. However, Theorem 9 shows an even sharper lower bound on the size of rewritings.

Theorem 10 *The problem of verifying the existence of an exact rewriting of a regular expression E_0 wrt a set $\mathcal{E}$ of regular expressions is 2EXPSPACE-complete.*

3 Query rewriting in semi-structured data

All semi-structured data models share the characteristic that data are organized in a labeled graph, where the nodes represent objects, and the edges represent links between objects [BDFS97, Bun97, Abi97, QRS+95]. From a formal point of view we can consider a *(semi-structured) database* as a graph DB whose edges are labeled by elements from a given domain $\mathcal{D}$ which we assume finite.

In order to define queries over a semi-structured database we start from a decidable, complete[1] first-order theory $\mathcal{T}$ over the domain $\mathcal{D}$. We assume that the language of $\mathcal{T}$ includes one distinct constant for each element of $\mathcal{D}$ and one unary predicate of the form $\lambda z.z = a$ (or simply a), for each constant a in $\mathcal{D}$. Finally, following [BDFS97], we consider both the size of $\mathcal{T}$, and the time needed to check validity of any formula in $\mathcal{T}$ to be constant.

We consider *regular path queries* (which we call simply queries) i.e., queries that denote all the paths corresponding to words of a specified regular language. The regular language is defined over a (finite) set $\mathcal{F}$ of formulae of $\mathcal{T}$ with one free variable. Such formulae are used to describe properties that the labels of the edges of the database must satisfy. Regular path queries are the basic constituents of queries in semi-structured data, and are typically expressed by means of regular expressions [BDHS96, Abi97, FS98, MS99]. Another possibility to express regular path queries is to use finite automata.

When evaluated over a database, a query Q returns the set of pairs of nodes connected by a path that conforms to the regular language $L(Q)$ defined by Q, according to the following definitions.

Definition 11 Given an $\mathcal{F}$-word $\varphi_1 \cdots \varphi_n$, a $\mathcal{D}$-word $a_1 \cdots a_n$ *matches* $\varphi_1 \cdots \varphi_n$ (wrt $\mathcal{T}$) iff $\mathcal{T} \models \varphi_i(a_i)$, for $i = 1, \ldots, n$. ∎

We denote the set of $\mathcal{D}$-words that match an $\mathcal{F}$-word w by $match(w)$, and given a language ℓ over $\mathcal{F}$, we denote $\bigcup_{w \in \ell} match(w)$ by $match(\ell)$.

Definition 12 The *answer to a query Q over a database DB* is the set $answer(L(Q), DB)$, where for a language ℓ over $\mathcal{D}$

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

∎

In order to apply the results on rewriting of regular expressions to query rewriting in semi-structured data we need to take into account that the alphabet over which queries (the one we want to rewrite and the views to use in the rewriting) are expressed, is the set $\mathcal{F}$ of formulae of the underlying theory $\mathcal{T}$, and not the set of constants that appear as edge labels in graph databases.

Let Q_0 be a regular path query and $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ be a finite set of views, also expressed as regular path queries, in terms of which we want to rewrite Q_0. Let $\mathcal{F}$ be the set of formulae of $\mathcal{T}$ appearing in $Q_0, Q_1, \ldots, Q_k$, and let $\mathcal{Q}$ have an associated alphabet $\Sigma_{\mathcal{Q}}$ containing exactly one symbol for each view in $\mathcal{Q}$. We denote the view associated to the symbol $q \in \Sigma_{\mathcal{Q}}$ with $rpq(q)$.

Given any language ℓ over $\Sigma_{\mathcal{Q}}$, we denote by $expand_{\mathcal{F}}(\ell)$ the language over $\mathcal{F}$ defined as follows

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

[1] The theory is complete in the sense that for every closed formula φ, either $\mathcal{T} \models \varphi$, or $\mathcal{T} \models \neg\varphi$ [BDFS97].

Definition 13 Let R be any formalism for defining a language $L(R)$ over $\Sigma_{\mathcal{Q}}$. R is a *rewriting of* Q_0 wrt $\mathcal{Q}$ if for every database DB, $answer(expand_{\mathcal{F}}(L(R)), DB) \subseteq answer(L(Q_0), DB)$, and is said to be (i) *maximal* if for each rewriting R' of Q_0 wrt $\mathcal{Q}$ we have that $answer(expand_{\mathcal{F}}(L(R')), DB) \subseteq answer(expand_{\mathcal{F}}(L(R)), DB)$, (ii) *exact* if $answer(expand_{\mathcal{F}}(L(R)), DB) = answer(L(Q_0), DB)$. ∎

Theorem 14 *R is a rewriting of Q_0 wrt $\mathcal{Q}$ iff $match(expand_{\mathcal{F}}(L(R))) \subseteq match(L(Q_0))$. Moreover, it is maximal iff for each rewriting R' of Q_0 wrt $\mathcal{Q}$ we have that $match(expand_{\mathcal{F}}(L(R'))) \subseteq match(expand_{\mathcal{F}}(L(R)))$, and it is exact iff $match(expand_{\mathcal{F}}(L(R))) = match(L(Q_0))$.*

We say that R is $\Sigma_{\mathcal{Q}}$-*maximal* if for each rewriting R' of Q_0 wrt $\mathcal{Q}$ we have that $L(R') \subseteq L(R)$. By arguing as in Theorem 3, and exploiting Theorem 14, it is easy to show that a $\Sigma_{\mathcal{Q}}$-maximal rewriting is also maximal.

Next we show how to compute a $\Sigma_{\mathcal{Q}}$-maximal rewriting, by exploiting the construction presented in Section 2. Applying the construction literally, considering $\mathcal{F}$ as the base alphabet Σ, we would not take into account the theory $\mathcal{T}$, and hence the construction would not give us the maximal rewriting in general. As an example, suppose that $\mathcal{T} \models \forall x.A(x) \supset B(x)$, $Q_0 = B$, and $\mathcal{Q} = \{A\}$. Then the maximal rewriting of Q_0 wrt $\mathcal{Q}$ is A, but the algorithm would give us the empty language.

In order to take the theory into account, we can proceed as follows: For each query $Q \in \{Q_0\} \cup \mathcal{Q}$ we construct the automaton Q^g accepting the language $match(L(Q))$. This can be done by viewing the query Q as a (possibly nondeterministic) automaton $Q = (\mathcal{F}, S, s_0, \rho, F)$ and construct Q^g as $(\mathcal{D}, S, s_0, \rho^g, F)$, where $s_j \in \rho^g(s_i, a)$ iff $s_j \in \rho(s_i, \varphi)$ and $\mathcal{T} \models \varphi(a)$. Observe that the set of states of Q and Q^g is the same. We denote $\{Q_1^g, \ldots, Q_k^g\}$ with $\mathcal{Q}^g$. Then we proceed as before:

1. Construct a deterministic automaton $A_d = (\mathcal{D}, S_d, s_0, \rho_d^g, F_d)$ such that $L(A_d) = L(Q_0^g)$.
2. Define the automaton $A' = (\Sigma_{\mathcal{Q}}, S_d, s_0, \rho', S_d - F_d)$, where $s_j \in \rho'(s_i, q)$ iff $\exists w \in match(L(rpq(q)))$ such that $s_j \in \rho_d^{g*}(s_i, w)$.
3. Return $R_{\mathcal{Q},Q_0} = R_{\mathcal{Q}^g,Q_0^g} = \overline{A'}$.

Theorem 15 *The automaton $R_{\mathcal{Q},Q_0}$ is a maximal rewriting of Q_0 wrt $\mathcal{Q}$.*

To check that $R_{\mathcal{Q},Q_0}$ is an exact rewriting of Q_0 wrt $\mathcal{Q}$ we can proceed as in Section 2, by constructing an automaton B that accepts $expand_{\mathcal{D}}(L(R_{\mathcal{Q}^g,Q_0^g}))$, and checking for the emptiness of $L(A_d \cap \overline{B})$.

Observe that both the size of Q_0^g and $\mathcal{Q}^g$ and the time needed to construct them from Q_0 and $\mathcal{Q}$ are linearly related to the size of Q_0 and $\mathcal{Q}$. It follows that the same upper bounds as established in Section 2 hold for the case of regular path queries.

In fact, the construction of $\mathcal{Q}^g$ can be avoided in building $R_{\mathcal{Q},Q_0}$, since we can verify whether there exists a $\mathcal{D}$-word $w \in match(L(rpq(q)))$ such that $s_j \in \rho_d^{g*}(s_i, w)$ (required in Step 2 of the algorithm above) as follows. We consider directly the automaton $Q = rpq(q)$ (which is over the alphabet $\mathcal{F}$) and the automaton $A_d^{i,j} = (\mathcal{D}, S_d, s_i, \rho_d^g, \{s_j\})$ obtained from A_d by suitably changing the initial and final states. Then we construct from Q and $A_d^{i,j}$ the product automaton K, with the proviso that K has a transition from (s_1, s_2) to (s_1', s_2') (whose label is irrelevant) iff (i) there is a transition from s_1 to s_1' labeled a in $Q_{i,j}$, (ii) there is a transition from s_2 to s_2' labeled φ in Q, and (iii) $\mathcal{T} \models \varphi(a)$. Finally we check whether K accepts a non-empty language. This allows us to instantiate the formulae in $\mathcal{Q}$ only to those constants that are actually necessary to generate the transition function of A'.

On the other hand, the construction of Q_0^g seems unavoidable, since formulae that satisfy more that one constant in $\mathcal{T}$ and that appear as labels of the transitions of Q_0, may hide a nondeterminism that is instead revealed when we consider Q_0^g.

4 Properties of rewritings

In the case where the rewriting $R_{\mathcal{Q},Q_0}$ is not exact, the only thing we know is that such rewriting is the best one we can obtain by using only the views in $\mathcal{Q}$. However, one may want to know how to get an exact rewriting by adding to $\mathcal{Q}$ suitable views. Here we consider the case where such views are *atomic*, i.e., have the form $\lambda z.P(z)$, where P is a predicate of $\mathcal{T}$. Notice that atomic views include views of the form $\lambda z.z = a$, (abbreviated by a), which we call *elementary*.

Example 16 Let $Q_0 = a \cdot (b + c)$, $\mathcal{Q} = \{a, b\}$, and $\Sigma_{\mathcal{Q}} = \{q_1, q_2\}$, where $rpq(q_1) = a$, and $rpq(q_2) = b$. Then $R_{\mathcal{Q},Q_0} = q_1 \cdot q_2$, which is not exact. On the other hand, by adding c to $\mathcal{Q}$ and q_3 to $\Sigma_{\mathcal{Q}}$, with $rpq(q_3) = c$, we obtain $q_1 \cdot (q_2 + q_3)$ as an exact rewriting of Q_0. ∎

The intuitive idea is to choose a subset $\mathcal{P}'$ of the set $\mathcal{P}$ of predicates of $\mathcal{T}$, and to construct an exact rewriting of Q_0 wrt $\mathcal{Q}_+$, where $\mathcal{Q}_+$ is obtained by adding to $\mathcal{Q}$ an atomic view for each symbol in $\mathcal{P}'$. An exact rewriting R of Q_0 wrt $\mathcal{Q}_+$ is called a *partial rewriting* of Q_0 wrt $\mathcal{Q}$, provided that $\mathcal{Q}_+ \neq \mathcal{Q}$.

The method we have presented can be easily adapted to compute partial rewritings. Indeed, if we compute $R_{\mathcal{Q}_+,Q_0}$, we obtain a partial rewriting of Q_0 wrt $\mathcal{Q}$, provided that $R_{\mathcal{Q}_+,Q_0}$ is an exact rewriting of Q_0 wrt $\mathcal{Q}_+$. Observe that it is always possible to choose a subset $\mathcal{P}'$ of $\mathcal{P}$ in such a way that $R_{\mathcal{Q}_+,Q_0}$ is exact (e.g., by choosing the set of all elementary views).

Typically, one is interested in using as few symbols of $\mathcal{P}$ as possible to form $\mathcal{Q}_+$, and this corresponds to choose the minimal subsets $\mathcal{P}'$ such that $R_{\mathcal{Q}_+,Q_0}$ is exact. More generally, one can establish various preference criteria for choosing rewritings. For instance, we may say that a (partial) rewriting R is *preferable* to a (partial) rewriting R' if:

1. $match(expand_{\mathcal{F}}(L(R'))) \subseteq match(expand_{\mathcal{F}}(L(R)))$,
2. $match(L(R)) = match(L(R'))$ and R uses less additional elementary views than R',
3. $match(L(R)) = match(L(R'))$, R uses the same number of additional elementary views as R', and less additional nonelementary views.
4. $match(L(R)) = match(L(R'))$, R uses the same number of additional atomic views as R', and less views than R'.

Under this definition an exact rewriting is preferable to a nonexact one. Moreover, the definition reflects the fact that the cost of materializing additional atomic views (in particular the elementary ones) is higher than the cost of using the available ones. Finally, since a certain cost is associated to the use of each view, when comparing two rewritings defining the same language and using (if any) the same number of additional atomic views, then the one that uses less views is preferable.

The rewriting algorithm presented above can be immediately exploited to compute the most preferable rewritings according to the above criteria. It easy to see that the problem of computing the most preferable rewritings remains in the same complexity class.

References

[Abi97] Serge Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 1–18, 1997.

[ACPS96] S. Adali, K.S. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 137–148, 1996.

[AD98] Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-98)*, pages 254–265, 1998.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[BDFS97] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 336–350, 1997.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization technique for unstructured data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516, 1996.

[BLR97] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 99–108, 1997.

[Bun97] Peter Buneman. Semistructured data. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 117–121, 1997.

[CDGLV99] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Vardi. Rewriting of regular expressions and regular path queries. Submitted for publication, 1999.

[CKPS95] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE-95)*, Taipei, Taiwan, 1995.

[CNS98] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. Technical report, The Hebrew University, Jerusalem, 1998. Submitted for publication.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 109–116, 1997.

[FFK+98] Mary F. Fernandez, Daniela Florescu, Jaewoo Kang, Alon Y. Levy, and Dan Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 414–425, 1998.

[FS98] Mary F. Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE-98)*, pages 14–23, 1998.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-95)*, pages 95–104, 1995.

[MS99] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT-99)*, 1999. To appear.

[QRS+95] D. Quass, A. Rajaraman, I. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, pages 319–344. Springer-Verlag, 1995.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-95)*, 1995.

[SDJL96] D. Srivastava, S. Dar, H.V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB-96)*, pages 318–329, 1996.

[TSI96] O.G. Tsatalos, M.H. Solomon, and Y.E. Ioannidis. The gmap: A versatile tool for phyisical data independence. *Very Large Database J.*, 5(2):101–118, 1996.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, number 1186 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 1997.