Queries with Incomplete Answers over Semistructured Data

Yaron Kanza
Institute for Computer Science
The Hebrew University
Jerusalem 91904, Israel
yarok@cs.huji.ac.il

Werner Nutt
German Research Center for
Artificial Intelligence GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
Werner.Nutt@dfki.de

Yehoshua Sagiv
Institute for Computer Science
The Hebrew University
Jerusalem 91904, Israel
sagiv@cs.huji.ac.il

Abstract

Semistructured data occur in situations where information lacks a homogeneous structure and is incomplete. Yet, up to now the incompleteness of information has not been reflected by special features of query languages for semistructured data. Our goal is to investigate the *principles* of queries that allow for incomplete answers. We do not present, however, a concrete query language.

Queries over classical structured data models contain a number of variables and conditions on these variables. An answer is a binding of the variables by elements of the database such that the conditions are satisfied. In the present paper, we loosen this concept in so far as we allow also answers that are *partial,* that is, not all variables in the query are bound by such an answer.

Partial answers make it necessary to refine the model of query evaluation. The first modification relates to the satisfaction of conditions: under some circumstances we consider conditions involving unbound variables as satisfied. Second, in order to prevent a proliferation of answers, we only accept answers that are *maximal* in the sense that there are no assignments that bind more variables and satisfy the conditions of the query.

Our model of query evaluation consists of two phases, a *search* phase and a *filter* phase. Semistructured databases are essentially labeled directed graphs. In the search phase, we use a query graph containing variables to match a maximal portion of the database graph. We investigate three different semantics for query graphs, which give rise to three variants of matching. For each variant, we provide algorithms and complexity results.

In the filter phase, the maximal matchings resulting from the search phase are subjected to constraints, which may be *weak* or *strong.* Strong constraints require all their variables to be bound, while weak constraints do not. We describe a polynomial algorithm for evaluating a special type of queries with filter constraints and assess the complexity of evaluating other queries for several kinds of constraints.

In the final part, we investigate the containment problem for queries consisting only of search constraints under the different semantics.

1 Introduction

The growing need to integrate data from heterogeneous sources and to access data sources with irregular or incomplete contents is the main motivation for research into semistructured data models and query languages for them. Semistructured data do not comply with a strict schema and are inherently incomplete. Query languages for such data should reflect these characteristics.

Semistructured data models have been intensively studied recently [Abi97, Bun97]. They originated with work on heterogeneous data integration [QRS+94, PGMW95, RU96]. Several models for representing semistructured data have been proposed together with query languages for those models such as Lorel [AQM+97, MAG+97, QWG+96] and UnQL [BDHS96]. Further topics of research have been the design of schemas for semistructured data [BDFS97] and the extraction of schemas from the data [GW97, NAM98].

A particular motivation for this research has been to allow one to access heterogeneous sources on the World Wide Web in an integrated fashion by providing a view of the web as a semistructured database [AV97a, KMSS98]. For the purpose of querying the World Wide Web several query languages and Web site management tools have been proposed, such as W3QL [KS95, KS97], WebSQL [MMM97], Strudel [FFK+98], Araneus [MAM+98], and others [LSS96, AM98]. The growing use of the Web emphasizes the need of querying semistructured data and retrieving partial answers when complete answers are not found.

We define a simple data model that is similar to OEM [PGMW95, AQM+97], where databases are labeled directed graphs. A node represents an object, a label an attribute, and a labeled edge links a node to another one if the second node is an attribute filler for the first node. Our queries, too, are defined by graphs, which are to be matched by the database. The idea to base a query language on graphs appeared already in [CMW82]. In this paper we apply it to semistructured data.

In an abstract view, database queries consist of a set of variables and constraints on the variables. A solution to a query is a binding of the variables to objects in the database such that the constraints are satisfied. In order to be able to accept as solutions also assignments that do not bind all variables, we refine the structure of a query. We divide the constraints into *search constraints* and *filter constraints.* The search constraints form a labeled directed graph whose nodes are variables: they are a pattern that has to be matched by some part of the database. We are only interested in *maximal matchings,* because they contain maximal information. The maximal matchings are then filtered: those that satisfy the filter constraints are called *solutions.* We distinguish between *strong* and *weak* filter constraints. A strong constraint is satisfied by an assignment if all variables of the constraint are bound and their values comply with the constraint, while a weak constraint is already satisfied if one of its variables is not bound by the assignment. Hence, it is important that only maximal matchings are filtered, since this enhances the applicability of the weak constraints. Finally, solutions are restricted to a subset of their variables, the *output variables.* Those restrictions are called *answers.* The roles of the different components of a query in our model can be illustrated by setting up an analogy with SQL-queries. The FROM-clause is the analogue of the search constraints, the WHERE-clause the analogue of the filter constraints, while the SELECT-clause specifies the output variables. Thus, with its search and filter constraints, a query is conceptually evaluated in two phases: the first being structural matching for the retrieval, and the second filtering. Query evaluation in this model is therefore non-monotonic: the fuller an assignment, the more filter constraints it has to satisfy.

We introduce different semantics for search constraints and investigate query evaluation under them. Our queries are essentially conjunctive queries. There is no explicit disjunction or negation, although some semantics give queries a disjunctive flavor. Our language is also restricted in that constraints on edges are only labels and not regular expressions, as in Lorel [AQM+97] and other query languages for semistructured data. The generalization to regular path expressions as edge constraints will be a topic for further research.

In this paper, we first define databases and queries in our model. We examine different semantics for the search phase of query answering, give algorithms for computing the maximal matchings of a query over a database with respect to those semantics, and study the evaluation of filter constraints. As a basis for query optimization, we give criteria for checking equivalence and containment of queries under the various semantics.

2 Data Model

Our data model is a simplified version of the Object Exchange Model (OEM) of [PGMW95, AQM+97]. Both data and queries are represented by labeled directed graphs. The nodes in a database graph are either complex or atomic. Complex nodes have outgoing edges, while atomic nodes do not. Atomic nodes have values. In each database there is one distinguished complex node, the root. It is the entry point for browsing and querying the database. Therefore, every node in the database must be reachable from it.

We assume that there is an infinite set $\mathcal{A}$ of *atoms* and an infinite set $\mathcal{L}$ of *labels*. Atoms can be of type *integer*, *real*, *string*, *gif*, etc. Each type comes with a set of decidable relations on the elements of the types, like comparisons on numbers and strings. For simplicity of exposition, we assume here that there is just a single type.

We give formal definitions and introduce the necessary notation. A *labeled directed graph* or *ldg* over a set of nodes N is a pair $G = (N, \cdot^G)$, where $\cdot^G$ associates with each label $l \in \mathcal{L}$ a binary relation $l^G \subseteq N \times N$ between the nodes. We will often view a binary relation l^G as a function $l^G\colon N \to 2^N$. Note that in a labeled directed graph there can be two nodes u and v such that in the graph there are two distinct edges from u to v labeled with different labels.

The *skeleton* of G is the union of the binary relations in G. Under the view of relations as functions, the skeleton is defined as the function $\sigma_G\colon N \to 2^N$ satisfying

$$\sigma_G(n) = \{n' \in N \mid n' \in l^G(n) \text{ for some } l \in \mathcal{L}\}.$$

The skeleton can be seen as the ldg where we ignore the labeling.

A labeled directed graph G over N is *rooted* if there is a designated node $r_G \in N$, the *root*, such that every node in N is reachable from r_G in σ_G. We denote a rooted ldg G as a triple $G = (N, r_G, \cdot^G)$. A node n in G is a *terminal node* if $\sigma_G(n) = \emptyset$, and an *inner node* otherwise. We say that G is *finitely branching* if $\sigma_G(n)$ is finite for every node n.
A *database* consists of

1. a rooted finitely branching ldg $(O, r_D, \cdot^D)$ over a set of objects O, and

2. a function α that maps each terminal node to an atom.

We denote a database as a 4-tuple $D = (O, r_D, \cdot^D, \alpha)$.
The graph in Figure 1 depicts a database containing information about university departments, courses and staff at the Hebrew University. The nodes are the database objects, and edges are annotated with their labels. We will use the example database of Figure 1 later on in our examples.

Generally, databases may be infinite. However, since they are finitely branching, there is only a finite number of nodes that can be reached from the root in a given number of steps. In particular, for queries without regular path expressions, a database has to be explored only up to a finite depth when answering the query, and therefore only a finite portion is relevant.

3 Overview of the Query Language

This section we formally define an abstract syntax of queries and define their semantics. The definitions will be illustrated by example queries over the university database.

We assume that there is an infinite set of *variables.* A query is a triple $Q = (G, F, \bar{x})$, where

1. $G = (V, r_G, \cdot^G)$ is a labeled directed graph, called the *query graph,* whose nodes are variables;

2. F is a set of *filter constraints,* whose syntax will be precisely defined later on; and

3. $\bar{x}$ is a tuple of variables occuring in V.

3.1 Search Constraints and Matchings

Let $G = (V, r_G, \cdot^G)$ be a query graph. We can view G also as a set of constraints $\mathit{Cons}(G)$ over V. There is a *root constraint,* singling out the root node, and for each pair of variables u, v, such that $v \in l^G(u)$, there is an *edge constraint* ulv. Conversely, a set S of edge constraints over V determines an ldg over V. If $v_0 \in V$ is singled out by the root constraint, then S determines a rooted ldg if every element of V is reachable from v_0 by a sequence of edge constraints. The constraints in $\mathit{Cons}(G)$ are called *search constraints* and are used for searching data in the database according to the structure they impose. Representing the query graph as a set of constraints allows us to view the entire query as a set of constraints. This makes the representation of queries more uniform and relates our queries to

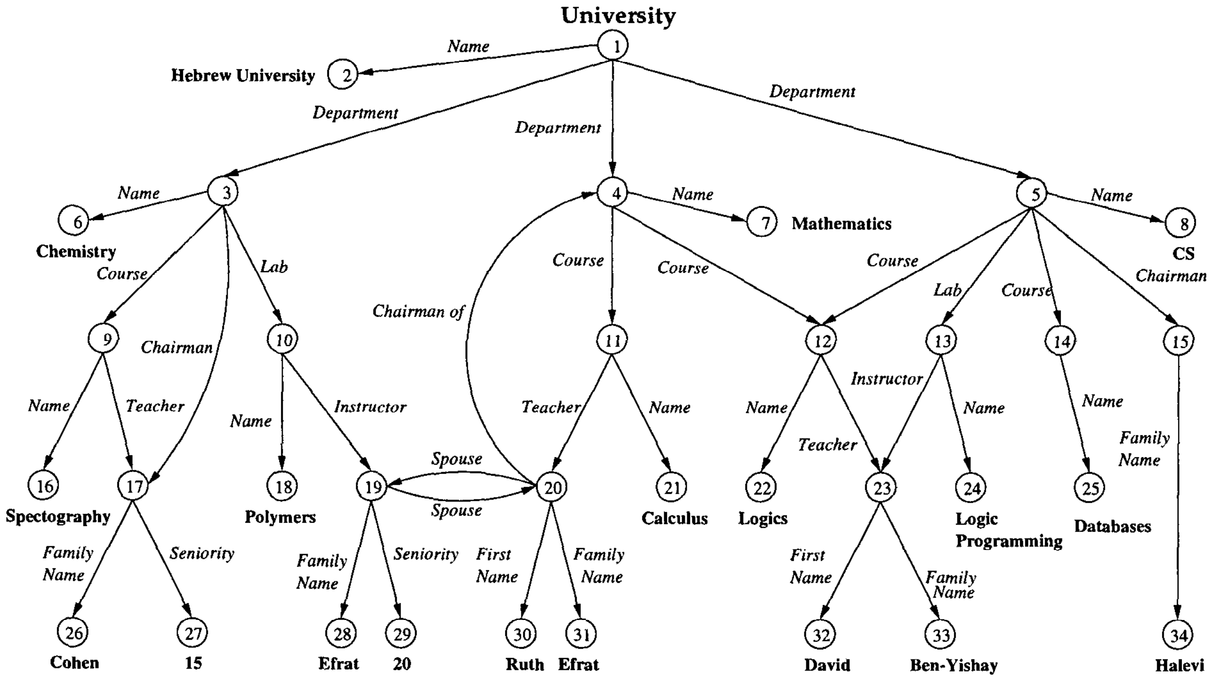

Figure 1: A university database

the well-known conjunctive queries. We also found that representing queries as sets of constraints eases the development and the descriptions of algorithms for query evaluation.

Let D be a database over O and V be the set of node variables occurring in the query graph G. We say that a *D-assignment over V* is a mapping $\mu: V \to O \cup \{\bot\}$, where $\bot$ is a new symbol, called *null.* If $\mu(v) \neq \bot$, we say that the assignment μ is *defined* for the variable v, or that v is *bound.* An assignment is *total* if it is defined for all variables in V and *partial* otherwise.

We say that an assignment μ *satisfies* an edge constraint ulv if $\mu(v) \in l^D(\mu(u))$, i.e., the relation l^D in the database contains the pair $(\mu(u), \mu(v))$. Note that μ has to be defined for the variables u, v in a constraint ulv in order to satisfy the constraint.

The assignment μ is a *strong matching* for G if it satisfies the following two conditions

1. $\mu(r_G) = r_D$, i.e., μ maps the root of G to the root of D, and
2. μ satisfies every edge constraint in G.

Thus, a strong matching is a total assignment.

The definition reflects the classical view of a conjunctive query and an answer to it. In order to generalize it to non-total matchings, we first single out assignments where, intuitively, the defined variables are connected to the root. Under this restriction, only connected pieces of the database can be matched against the query. Matchings, thus, contain only pieces of information that are related to each other, and can be computed by traversing the database.

Let μ be a D-assignment over V, and let C_μ be the set of edge constraints in $Cons(G)$ satisfied by μ. Then μ is a *prematching* of Q if

1. $\mu(r_G) = r_D$, and
2. every variable to which μ assigns a non-null value is reachable from r_G by a path in the graph C_μ.

We now define matchings of the query to the database which are prematchings that only partially satisfy the search constraints of the query. Let μ be a prematching of Q. We say that μ is a *weak matching* if, whenever μ is defined for u and v, and G contains a constraint ulv, then μ satisfies ulv. That is, a weak matching has to satisfy every constraint whenever it is defined for the variables of that constraint. Weak matchings are a natural generalization of strong matchings to the case of partial assignments. The rigid requirement that the assignment satisfy *all* constraints is relaxed in so far as we expect a partial assignment only to satisfy those constraints for which it binds all variables.

In addition to weak matchings, we give two other definitions of matchings that take into account the graph structure of a query. In a query graph G, the constraints of the form ulv are called the *incoming constraints* of the variable v. The prematching μ is an AND-*matching* if it satisfies *all* incoming constraints of v whenever $\mu(v) \neq \bot$. We also say that a prematching μ is an OR-*matching* of Q, since it satisfies *some* incoming constraint of v whenever $\mu(v) \neq \bot$.

Intuitively, when computing OR-matchings, we view the query graph as a description of a set of possible paths along which to explore the database and to collect as much information as possible. By an OR-matching, a query variable can be bound if there exists *some* path in the query from the root to the variable such that the matching binds all the variables on the path and satisfies all edge constraints on the path. Contrary to an OR-matching, an AND-matching can only bind a query variable if for *all* paths from the root to that variable the matching binds all variables on the paths and satisfies all edge constraints.

The sets of all strong, weak, AND, and OR-matchings of

Q over D are denoted as

$$Mat^{s}_{D}(Q),\ Mat^{w}_{D}(Q),\ Mat^{\wedge}_{D}(Q),\ Mat^{\vee}_{D}(Q),$$

respectively. Obviously, we have

$$Mat^{s}_{D}(Q) \subseteq Mat^{\wedge}_{D}(Q) \subseteq Mat^{w}_{D}(Q) \subseteq Mat^{\vee}_{D}(Q). \qquad (1)$$

If the query graph is a tree, then there is no distinction between weak, AND, and OR-matchings. For $* \in \{s, \wedge, w, \vee\}$ we also refer to $Mat^{*}_{D}(Q)$ as matchings under strong, AND, weak, and OR-semantics, respectively.

Let A be a set of assignments that range over the same set of variables V and let μ, $\mu' \in A$. We say that μ' *subsumes* μ, and write $\mu \sqsubseteq \mu'$, if, whenever $\mu(v)$ is defined, then $\mu'(v)$ is defined and $\mu(v) = \mu'(v)$. Intuitively, the subsuming assignment contains more information than the subsumed one. An assignment μ is a *maximal* element of A if for any element $\mu' \in A$ we have that $\mu \sqsubseteq \mu'$ implies $\mu = \mu'$.

Given a query Q and a database D, we are interested in matchings that have maximal information content. For each $* \in \{s, \wedge, w, \vee\}$ we thus define $MMat^{*}_{D}(Q)$ as the set of maximal elements of $Mat^{*}_{D}(Q)$. Obviously, $MMat^{s}_{D}(Q)$ is the same as $Mat^{s}_{D}(Q)$, but for the other types of matchings not all matchings are maximal. Hence, the analogue of Equation (1) does not hold for the sets of maximal matchings. However, if $Mat^{*}_{D}(Q) \subseteq Mat^{\dagger}_{D}(Q)$, then for every $\mu \in MMat^{*}_{D}(Q)$ there is a $\mu' \in MMat^{\dagger}_{D}(Q)$ such that μ is subsumed by μ'. To illustrate our definitions, we consider an example query graph for which we compute the different kinds of maximal matchings over the university database.

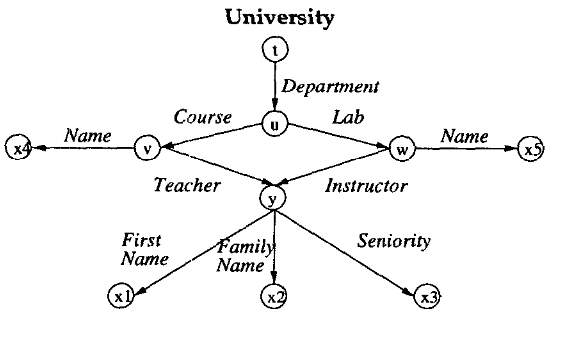

Figure 2: Query graph G_1 asking for course teachers and lab instructors

Example 3.1 (Matchings under AND, Weak and OR-Semantics) Figure 2 depicts the query graph G_1. We evaluate G_1 over the university database. Table 1 contains the maximal matchings under AND, weak, and OR-semantics. Note that all maximal matchings in the table are partial assignments. Thus, under strong semantics we would not retrieve any answer to our example query.

Under the different semantics, the query in Figure 2 has different meanings. Under AND-semantics, it requests the people that are both a course teacher and a lab instructor. Under OR-semantics, we can use it to find those people that are either a course teacher or a lab instructor. We can use here also weak semantics instead of OR-semantics in order to achieve more intuitive results as will be explained in Example 3.2.

Semantics	t	u	v	w	y	$x1$	$x2$	$x3$	$x4$	$x5$
AND	1	3	9	10	⊥	⊥	⊥	⊥	16	18
	1	4	11	⊥	⊥	⊥	⊥	⊥	21	⊥
	1	4	12	⊥	⊥	⊥	⊥	⊥	22	⊥
	1	5	12	13	23	32	33	⊥	22	24
	1	5	14	13	⊥	⊥	⊥	⊥	25	24
Weak	1	3	9	10	⊥	⊥	⊥	⊥	16	18
	1	3	9	⊥	17	⊥	26	27	16	⊥
	1	3	⊥	10	19	⊥	28	29	⊥	18
	1	4	11	⊥	20	30	31	⊥	21	⊥
	1	4	12	⊥	23	32	33	⊥	22	⊥
	1	5	12	13	23	32	33	⊥	22	24
	1	5	14	13	⊥	⊥	⊥	⊥	25	24
OR	1	3	9	10	17	⊥	26	27	16	18
	1	3	9	10	19	⊥	28	29	16	18
	1	4	11	⊥	20	30	31	⊥	21	⊥
	1	4	12	⊥	23	32	33	⊥	22	⊥
	1	5	12	13	23	32	33	⊥	22	24
	1	5	14	13	23	32	33	⊥	25	24

Table 1: Maximal matchings for the search graph G_1 in Figure 2 over the university database

3.2 Filter Constraints and Solutions

Filter constraints reduce the set of maximal matchings to a set of *solutions*. We now define the syntax and semantics of filter constraints.
A *type* is a pair $T = (\mathcal{D}, \mathcal{R})$, where

1. $\mathcal{D}$ is a set of values, called the *domain* of T, and
2. $\mathcal{R}$ is a set of decidable relations $r \subseteq \mathcal{D} \times \ldots \times \mathcal{D}$ over the domain $\mathcal{D}$.

The elements of $\mathcal{D}$ are the *atomic values* or *atoms* of T. Part of a database in our model is a function α that maps the terminal nodes to atoms and each atom is of some type. For simplicity we assume for now that there is only one type T, and that all atomic values are of this type.

We distinguish between three kinds of filter constraints: atomic constraints, object comparisons, and existence constraints. An *atomic constraint* of arity n is an expression $r(\bar{s})$, where r is a relation of arity n, and $\bar{s} = (s_1, \ldots, s_n)$ is a tuple of variables and atomic values of $\mathcal{D}$. Equality and inequality of the atomic values attached to terminal nodes are special atomic constraints and are denoted as

$$u \doteq_v v \qquad \text{and} \qquad u \not\doteq_v v,$$

respectively. *Object comparisons* are constraints of the form

$$u \doteq_o v \qquad \text{and} \qquad u \not\doteq_o v.$$

The first constraint is an *object equality* while the second is an *object inequality*. Unlike equality and inequality constraints that are used for comparing atomic values, object comparisons are used for comparing database objects for identity and distinctness. Two objects can have equal values without being equal themselves. An existence constraint has the form

$$!v.$$

The existence constraint $!v$ requires the variable v to be bound.

Filter constraints are applied to maximal matchings, in which certain variables may be undefined. We take this into account by distinguishing between two kinds of satisfaction of a constraint by an assignment, which we call strong and

Semantics	First Name	Family Name	Seniority	Course Name	Lab Name
And	David	Ben-Yishay		Logics	Logic Programming
Weak		Cohen	15	Spectography	
		Efrat	20		Polymers
	Ruth	Efrat		Calculus	
	David	Ben-Yishay		Logics	Logic Programming

Table 2: Maximal answers of Q_1 over the university database under AND and weak semantics

weak satisfaction. An assignment μ *strongly satisfies* an atomic constraint $r(s_1, \ldots, s_n)$ if for every s_i that is a variable

1. $\mu(s_i)$ is defined and equal to an atomic database node; and
2. the tuple $(\alpha(\mu(s_1)), \ldots, \alpha(\mu(s_n)))$ is in the relation r.

The assignment μ *strongly satisfies* the object equality $u \doteq_o v$ if $\mu(u)$ and $\mu(v)$ are defined, and $\mu(u) = \mu(v)$. For object inequalities the definition is analogous.

Thus, in order for an assignment to strongly satisfy a constraint, it must be defined for all the variables occurring in the constraint, and the values must be in the constraint relation. An assignment *weakly satisfies* a constraint if it is undefined for one of the variables of the constraint, or otherwise, if it strongly satisfies it. If μ strongly (weakly) satisfies a set of constraints C, we write $\mu \models_s C$ ($\mu \models_w C$).

We say that μ satisfies the existence constraint $!v$ if $\mu(v) \neq \bot$. Here, we do not distinguish between strong and weak satisfaction. Existence constraints can be used to turn weak satisfaction into strong satisfaction. If c_f is a filter constraint with variables $v_1, \ldots, v_n$ and μ is an assignment, then $\mu \models_s \{c_f\}$ if and only if $\mu \models_w \{c_f, !v_1, \ldots, !v_n\}$.

The set F of filter constraints in a query $Q = (G, F, \bar{x})$ is partitioned into sets F_s and F_w, to which we refer as strong and weak filter constraints. For $* \in \{s, \wedge, w, \vee\}$ we define the set of strong, AND, weak and OR-*solutions* of Q over D as

$$Sol^*_D(Q) := \left\{\mu \in MMat^*_D(Q) \;\middle|\; \mu \models_s F_s \text{ and } \mu \models_w F_w\right\}.$$

3.3 Answers

Up to now we have defined the role of the search and filter constraints in the evaluation of a query $Q = (G, F, \bar{x})$ over a database D. The result is the set of solutions $Sol^*_D(Q)$, which depends on the semantics $*$.

Instead of all solutions, however, we are in general only interested in the bindings of some of the variables in the query. We select these variables as the output variables $\bar{x}$. We define the set of strong, AND, weak and OR-*answers* of Q over D as consisting of the tuples $\mu(\bar{x})$, where μ is a strong, AND, weak, or OR-solution of Q respectively, and denote it as $Ans^*_D(Q)$, where $* \in \{s, \wedge, w, \vee\}$.

3.4 Examples

We give now some examples to illustrate the previously defined concepts.

Example 3.2 (Solutions and Answers) In Example 3.1 we saw the sets of maximal matchings over the university database for the query in Figure 2. Since we intend to find information about people with this query, we require the variable y to be bound by adding the existence constraint $!y$. As a result, we want the information about the person contained in the variables $x1$, $x2$, $x3$, the information about the course the teacher teaches in $x4$, and the information about the lab for which the person is the instructor in $x5$. Thus, we declare $x1$, $x2$, $x3$ $x4$, $x5$ as output variables.

Let $Q_1 = (G_1, F_1, \bar{x})$ be the query, where G_1 is the query graph in Figure 2, $F_1 = \{!y\}$, and $\bar{x} = (x1, x2, x3, x4, x5)$. Table 2 shows the maximal answers to Q_1 under AND and weak semantics. We see that under AND-semantics, we receive the one person who teaches a course and is also a lab instructor, together with the information about the course and the information about the lab. (Instead of the terminal objects in the answer, the attached values appear in the table.)

Under weak semantics we get information about all those persons who teach a course or instruct a lab, and we get the information about the lab they instruct or the course they teach. If the person only teaches a course, we only get the information about the course he teaches. If the person only instructs a lab, we only get information about the lab he instructs. If the person does both, we get the information about the course and the lab that correspond to him.

If here we were using OR-semantics instead of weak semantics, we would still get the information about people teaching a course or instructing a lab, but in addition, we may also get information about a person together with the information about the course he *does* teach, but with information about a lab in his department which he *does not* instruct. For instance, under OR-semantics, David Ben-Yishai together with the Logic Programming Lab and the course on Databases may be returned as an answer, although David does not teach Databases. A motivation for using OR-semantics will be given in Example 3.4.

Example 3.3 (Weak Filter Constraints) Suppose, we are interested only in those staff members with a seniority of at least 20 years. We can add to query Q_1 a filter constraint which requires that the atomic value of the database object that is bound to the variable $x3$ will be greater or equal to 20.

However, some matchings are not defined for the variable $x3$. It may be the case that some person has a seniority greater than 20, but this information is not present in the database. It may therefore be rash to dismiss them as solutions.

In order to admit them as solutions, we treat the filter *constraint* $x3 \geq 20$ *as a weak constraint.* Then $x3 \geq 20$ is satisfied if either we have positive information that the seniority is at least 20, or if there is no information about seniority.

The idea underlying OR-semantics is to use the query graph as a pattern of paths for traversing the database. During the search phase, we retrieve as many matchings that are as full as possible, and we filter the unneeded matchings using the filter constraints.

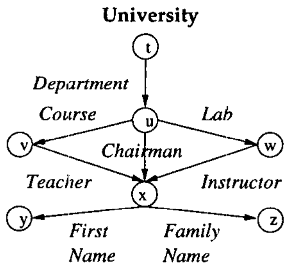

Figure 3: Query graph G_2 asking for the staff in the university database

Semantics	t	u	v	w	x	y	z
	1	3	9	10	17	$\perp$	Cohen
	1	3	9	10	19	$\perp$	Efrat
	1	4	11	$\perp$	20	Ruth	Efrat
OR	1	4	12	$\perp$	23	David	Ben-Yishay
	1	5	12	13	23	David	Ben-Yishay
	1	5	14	13	23	David	Ben-Yishay
	1	5	12	13	15	$\perp$	Halevi
	1	5	14	13	15	$\perp$	Halevi

Table 3: Maximal OR-matchings for query graph G_2 over the university database

Example 3.4 (OR-**Semantics**) If we evaluate the query graph G_2 in Figure 3 under OR-semantics, we retrieve the staff in the university database. We find all the people that are either course teacher, lab instructor, or chairman of a department. We cannot use weak semantics for that purpose, since in weak semantics, the variable u must be bound if x is bound, and the binding must satisfy the edge constraint u *Chairman* x. Thus, whenever x is bound in a weak matching, then it is bound to the chairman of the department.

Table 3 shows the set of maximal OR-matchings for the query graph in Figure 3. For terminal nodes, we have listed the attached atoms instead of the identity of the object.

4 Computing Matchings

This section summarizes results about the computation of sets of maximal matchings under the three semantics allowing for partial matchings, i.e., weak, AND, and OR-semantics.

Matchings depend only on the query graph, which contains the search constraints. They are the result of the (conceptually) first phase of query evaluation. We distinguish between three cases where the search graph of the query is either a tree, an acyclic ldg, or a general ldg. According to the form of the search graph, we call a query a *tree query* or a *dag query*.

It will turn out that for acyclic graphs, the set of maximal matchings can be computed in time polynomial in the size of the query, the database, and the result.

4.1 Tree Queries

We have seen that there is no difference between the matchings under the three semantics if the query graph is a tree. Consequently, the sets of maximal matchings are the same. We sketch an algorithm *EvalTreeQuery* that computes them.

The algorithm proceeds by first topologically sorting the nodes of the query graph, that is by establishing a linear ordering $v_0 < v_1 < \ldots < v_n$ on the nodes with the property that $v_i < v_j$ if there is an edge from v_i to v_j in the graph. Obviously, the first node v_0 in such an ordering is the root. The algorithm then performs an iteration over the nodes according to this order. It maintains a set S of assignments of database objects to the variables visited. It starts by assigning the root of the database to the root of the query. Consider the step in which it processes variable v_i with $i > 0$, and suppose that the incoming edge of v_i is $v'lv_i$. Then each assignment $\mu \in S$ is extended to v_i by assigning objects $o \in l^D(\mu(v'))$ to it, if there are any, and by assigning $\perp$ to v_i otherwise. When all variables are processed, S is the set of maximal matchings under AND, OR, and weak semantics. It is easy to derive an upper bound for the runtime of the algorithm that is polynomial in the size of the input and the output.

Theorem 4.1 (Complexity of Tree Queries) *Suppose the algorithm EvalTreeQuery is called with a tree query Q and a database D then it outputs the set of maximal matchings under* AND, OR, *and weak semantics, and has runtime which is* $O(|D| * |Q| * |S|)$.

4.2 Acyclic Queries

The algorithm described above can be generalized to algorithms for dags. In a dag query, there may be more than one incoming constraint for a variable. Consequently, the sets of matchings may vary according to the semantics.

We describe algorithms *EvalOrDag* and *EvalAndDag* for AND and OR-semantics. Under OR-semantics, a variable can be bound if in the query *some* path from the root to the variable is bound, and under AND-semantics it can be bound if *every* path from the root to the variable is bound. The algorithms differ from the one for trees in the iteration step where objects are assigned to the variable v_i. Suppose the incoming edge constraints for a variable v_i are the constraints $u_1 l_1 v_i, \ldots, u_k l_k v_i$. Suppose in addition that we have an assignment μ that has already been processed for the variables $v_0, \ldots, v_{i-1}$, and that we want to extend it to v_i. Now, *EvalOrDag* binds o to v_i if there is *some* u_j such that $o \in l_j^Q(\mu(u_j))$.

By the algorithm *EvalAndDag*, an object o is bound to v_i if $o \in l_j^Q(\mu(u_j))$ for *all* $j \in 1..k$, and we consider $l_j^Q(\mu(u_j))$ to be empty if μ is not defined for u_j.

Since the nodes in the query are topologically sorted, the two policies for binding v_i guarantee that the algorithms compute the sets of maximal OR and AND-matchings, respectively. Again, it is easy to check that the computation takes only time polynomial in the size of the input and output.

Theorem 4.2 (Complexity of AND **and of** OR-**Dag Queries)** *The algorithms EvalOrDag and EvalAndDag, when called with a dag query Q and a database D, output the set of maximal matchings under* AND *and* OR-*semantics, respectively. They have runtime* $O(|D|^2 * |Q|^2 * |S|)$.

An algorithm for computing the maximal weak matchings is more sophisticated. However, it is still possible to compute the set of maximal weak matchings in time polynomial in the size of the input and the output.

Theorem 4.3 **(Complexity of Weak Dag-Queries)** *There is an algorithm that, when called with a dag query Q and a database D, outputs the set of maximal weak matchings. It has runtime $O(|D|^2 * |Q|^3 * |S|)$.*

4.3 General Queries

Clearly, the sets of maximal matchings under any of our three semantics are computable in exponential time. The interesting question is whether it is still possible to do it in time polynomial in the size of the input and output. For AND-semantics we can show that the existence of such an algorithm is highly unlikely.

Theorem 4.4 *If there is an algorithm that computes the set $MMat^{\wedge}_{D}(Q)$ for arbitrary query graphs Q in time polynomial in the input and the output, then* PTIME = NP.

Proof. We only provide a sketch. The proof is based on a reduction of the Hamiltonian path problem. More precisely, for a given graph G, we construct a query Q and a database D_G such that $MMat^{\wedge}_{D}(Q)$ contains only the assignment μ_0 that assigns the root of D to the root of Q and $\perp$ to all other variables if and only if G does not have a Hamiltonian path. If there were an algorithm that computes $MMat^{\wedge}_{D}(Q)$ in time polynomial in the size of input and output, one could derive a polynomial algorithm that returns μ_0 if and only if G does not have a Hamiltonian path. □

5 Computing Solutions

In the previous section we saw how to compute the set of maximal matchings for a query over a database. In this section we examine the additional effect of filter constraints on the complexity of query evaluation, considering separately existence, inequality, and equality constraints.

5.1 *Adding Existence Constraints*

Existence constraints require query nodes to be bound in the filtered solutions. This causes parts of the query graph to be evaluated under strong semantics and turns some filter constraints into strong constraints. We show that evaluating tree queries with existence constraints is polynomial in the size of input and output, while for dag queries no polynomial algorithm exists, provided PTIME $\neq$ NP.

Let $Q = (G, F, \bar{x})$ be a query, where G is a tree and F contains only existence constraints. A node v in Q is *enforced* if there is an existence constraint $!v$ in F. In a tree query, for a node to be bound by an assignment, all its ancestors must be bound. Let V'' be the set of enforced nodes of Q, and V' be the set of nodes such that there is a path from the root to an enforced node in V''. Then we call the restriction of G to the set of nodes V' the *strongly evaluated subgraph* of G.

The strongly evaluated subgraph of a query tree is the part that must be evaluated under strong semantics. We now sketch an algorithm *EvalStrongTree* that evaluates tree queries under strong-semantics. The algorithm is similar to the computation of an acyclic join in relational algebra using a semijoin program (full reducer) as in [BG81]. It consists of the following steps:

1. find a topological order for the edges of the tree;
2. traverse the tree according to the topological order and compute for each edge constraint a relation by collecting the pairs of database objects satisfying it;
3. reduce the relations using semijoins backwards with respect to the topological order;
4. join the reduced relations;
5. create the set of assignments from the tuples in the join.

Theorem 5.1 **(Complexity of Strong Tree Queries)** *The algorithm EvalStrongTree, when called with a tree query Q and a database D, outputs the set S of strong solutions. It has runtime $O(|Q| * (|D|^2 + |S|))$.*

The set of filtered maximal solutions can now be computed by first finding the strongly evaluated subtree of the query, computing the set of strong solutions for this subtree using *EvalStrongTree*, and extending the strong solutions to the rest of the graph in a similar way as maximal matchings are computed for a tree query (cf. Section 4).

Theorem 5.2 **(Polynomiality of Tree Queries)** *When adding only existence constraints as filter constraints to tree-queries, the set of maximal filter solutions under any of strong,* AND, *weak, and* OR *semantics can be computed in time polynomial in the size of the query, the database, and the solution set.*

There is no polynomial algorithm for evaluating dag-queries with existence constraints, provided PTIME $\neq$ NP. We say that the *evaluation problem* for a fixed semantics and a class of queries is to decide whether for a given query and database the set of solutions under that semantics is nonempty.

Theorem 5.3 *The evaluation problem for dag queries with existence constraints is* NP*-complete.*

Proof. (Sketch) There is a classical reduction of 3SAT to the problem of evaluating a conjunctive query over a database. The reduction can be modified such that all the predicates are binary. Hence, the query has the form of a graph. It turns out, that the graph is even a dag. Evaluation of such a conjunctive query is the same as evaluation under strong semantics. Evaluation under strong semantics is enforced by decorating all nodes of a query with existence constraints. □

Under AND-semantics, for a node to be bound all its incoming constraints must be satisfied, and therefore all its parent nodes must be bound. Hence, decorating only the terminal nodes of a dag with existence constraints has the same effect as decorating all nodes. Under weak and OR-semantics, however, some nodes without existence constraints may remain unbound. One may ask whether this has an effect on the difficulty of the evaluation problem.

Theorem 5.4 *The evaluation problem for dag queries with existence constraints under both, weak and* OR*-semantics, is* NP*-complete, even if existence constraints occur only at the terminal nodes of the query.*

Proof. (Sketch) The proof is again by a reduction of 3SAT. The disjunction in 3SAT is encoded into the different choices that can be made among the paths leading to the constrained terminal nodes. □

5.2 Adding Weak Equality and Inequality Constraints

Allowing weak equality or inequality constraints in the filter constraints makes the evaluation of a query NP-hard. It is interesting to note that this is already the case for tree queries. Thus, the results do not depend on the semantics under which the search constraints are evaluated.

Theorem 5.5 (Equalities and Inequalities) *The evaluation problem for tree-queries with either weak equality constraints or weak inequality constraints is* NP*-complete. The* NP*-hardness holds for equalities and inequalities, both for values and objects.*

Proof. (Sketch) The proof for queries with equality constraints is by a reduction of One-In-All-Pos-3SAT, while the proof for inequality constraints is by a reduction of 3-colorability. □

6 Containment of Search Queries

In this section, we examine containment and equivalence between queries without filter constraints, that is queries of the form $Q = (G, \emptyset, \bar{x})$. For the sake of brevity, we denote such a query as $Q = (G, \bar{x})$. We leave the study of more complex queries to future research. Our definitions, however, apply to the general case.

Since we are dealing with answers that are partial, that is, which may contain the value $\perp$, we define containment and equivalence modulo subsumption of answers. Let $\bar{a}$ and $\bar{a}'$ be tuples of the same length that contain either atoms or $\perp$. We say that $\bar{a}$ is *subsumed* by $\bar{a}'$, and write $\bar{a} \sqsubseteq \bar{a}'$, if $a_i = a_i'$ whenever $a_i \neq \perp$ for each component a_i of $\bar{a}$ and a_i' of $\bar{a}'$. An answer in the set $Ans^*_D(Q)$ is *maximal* if it is a maximal element of $Ans^*_D(Q)$ w.r.t. to the ordering "$\sqsubseteq$".

The definition of containment is parameterized by the semantics under which the search constraints are evaluated. Consider two queries $Q = (G, F, \bar{x})$ and $Q' = (G', F', \bar{x})$ that have the same output variables. Let σ be one of the semantics defined before—strong, AND, weak, or OR-semantics. Then Q and Q' are *equivalent* under σ if for every database they return the same maximal answers w.r.t. σ. We say that Q is *contained* in Q' under σ if over every database D, and for every $\bar{a} \in Ans^{\vee}_D(Q)$ there is an $\bar{a}' \in Ans^{\vee}_D(Q')$ such that $a \sqsubseteq a'$. Obviously, two queries are equivalent under σ iff they subsume each other w.r.t. σ. We write $Q \sqsubseteq_* Q'$, where $* \in \{s, \wedge, w, \vee\}$, if Q is contained in Q' under strong, AND, weak, or OR-semantics, respectively.

Under a semantics that allows for solutions with the value $\perp$, not all variables in the query graph are needed for computing answers. More precisely, a variable from which there is no path to an output variable never needs to be bound. By eliminating such nodes, we can cut down the query graph so that it contains only nodes that are needed.

Let G be a query graph and $\bar{x}$ be a tuple of variables appearing in G. By $Pr_{\bar{x}}(G)$ we denote the restriction of G to those nodes which are on a path from r_G to one of the variables in $\bar{x}$. We call $Pr_{\bar{x}}(G)$ the *pruned version* of G. The pruned version of a query graph G has r_G as its own root, and every node in it is reachable from the root. Thus, it is a legal query graph. Given a tuple of variables $\bar{x}$, we say that a query graph is *pruned* if $Pr_{\bar{x}}(G) = G$. By extension, if $Q = (G, \bar{x})$ is a query, we call the query $Q' = (Pr_x(G), \bar{x})$ the *pruned version* of Q, and we say that a query is pruned if its query graph is pruned.

Proposition 6.1 *Let $Q = (G, \bar{x})$ be a query and let $Q' = (Pr_{\bar{x}}(G), \bar{x})$ be its pruned version. Then for any database D and for $* \in \{\wedge, \vee, w\}$ we have*

$$Ans^*_D(Q) = Ans^*_D(Q').$$

By the above proposition, in order to decide containment, it is sufficient to check the pruned versions of queries for containment.

To characterize containment of queries, we need the concept of a query homomorphism. Let $Q_1 = (G_1, \bar{x})$ and $Q_2 = (G_2, \bar{x})$ be two queries. A mapping φ from the variables of Q_1 to the variables of Q_2 is a *homomorphism* from Q_1 to Q_2 if

1. it maps roots to roots, that is, $\varphi(r_{G_1}) = r_{G_2}$;
2. it maps output variables to output variables, that is, $\varphi(x_j) = x_j$ for each $x_j \in \bar{x}$;
3. it maps edge constraints to edge constraints, that is, $\varphi(u) l \varphi(v)$ is a constraint in G_2 for each ulv in G_1.

We start by examining containment for the case of two queries that are evaluated under AND-semantics.

Theorem 6.2 (Containment under AND-Semantics) *Let $Q_1 = (G_1, \bar{x})$ and $Q_2 = (G_1, \bar{x})$ be two pruned queries. Then $Q_1 \subseteq_\wedge Q_2$, i.e., Q_1 is contained in Q_2 under AND-semantics, if and only if there is a homomorphism from Q_2 to Q_1.*

Since the existence of a graph homomorphism is NP-complete, it follows that containment of queries under AND-semantics is NP-complete. For tree queries, AND, weak, and OR-semantics coincide. Thus Theorem 6.2 implies that the existence of a homomorphism is a sufficient and a necessary condition for containment among pruned tree queries under AND, weak and OR-semantics.

We now want to check containment under OR-semantics. The basic idea in checking containment under OR-semantics is to reduce containment of arbitrary queries to containment of tree queries. Let $G = (V, r_G, \cdot^G)$ be a query graph. A *spanning tree* of G is a subgraph $T = (V, r_T, \cdot^T)$ of G that has the same nodes and the same root as G, and whose skeleton is a tree. If $Q = (G, \bar{x})$ is a query, then we define the set of queries

$$\mathcal{T}_Q := \Big\{ (T, \bar{x}) \;\Big|\; T \text{ is a pruned spanning tree of } G \Big\}.$$

We call $\mathcal{T}_Q$ the *tree expansion* of Q. Under OR-semantics, a query can be evaluated by evaluating each query in its tree expansion and taking the union of the results. We will use this fact in order to characterize containment among queries under OR-semantics.

Proposition 6.3 *Let $Q = (G, \bar{x})$ be a query and $\mathcal{T}_Q$ its tree expansion. Then we have for any database D that*

$$Ans^{\vee}_D(Q) = \bigcup_{Q' \in \mathcal{T}_Q} Ans^{\vee}_D(Q').$$

Theorem 6.4 (Containment under OR-Semantics) *For two queries $Q_1 = (G_1, \bar{x})$ and $Q_2 = (G_2, \bar{x})$ the following are equivalent:*

- *$Q_1 \sqsubseteq_\vee Q_2$;*

- *for every query $Q_1' \in \mathcal{T}_{Q_1}$ there is a query $Q_2' \in \mathcal{T}_{Q_2}$ such that $Q_1' \sqsubseteq Q_2'$.*

Theorem 6.5 (Complexity of OR-Containment)

1. *Containment of queries under* OR*-semantics is in Π_2^{P}.*
2. *The problem is* NP*-complete if the containee is a tree.*
3. *It is polynomial if the container is a tree.*

Proof. That containment is in Π_2^{P}, is clear from the characterization in Theorem 6.4. The third statement is also clear, since it can be decided in polynomial time whether there is a homomorphism from a tree to an arbitrary graph. The second claim can be shown by a reduction of the Hamiltonian path problem. □

For weak semantics we have a characterization of containment resembling the one for OR-semantics that is given in Theorem 6.4. The idea is to replace the set of spanning trees by a set of pruned graph fragments, where a graph fragment is a restriction of the query graph to a subset of the variables in the query such that this subset contains the root of the query and such that all the variables in the fragment are reachable from the root.

Also, for weak semantics complexity results analogous to those in Theorem 6.5 hold.

7 Conclusion and Related Work

Semistructured data models are distinguished from classical "structured" data models by two characteristics: they do not assume that data have a homogeneous structure, and they do not assume that data are complete.

The query languages proposed for semistructured data so far take only the first characteristic into account. In most models for semistructured data, databases are essentially labeled directed graphs. Typically, one can formulate in the languages proposed thus far navigational queries with regular path expressions, which apply to a wide range of graph structures in a database, and are therefore not restricted to one prespecified schema, see [AV97b, FLS98].

In the present paper, we have concentrated on the second characteristic. In our opinion, a congenial query language for incomplete data must allow incomplete answers as query results. In this paper, we have presented some theoretical principles for such languages. We believe that maximal partial answers can contain useful information in situations where complete answers are not available.

The work on full disjunctions [GL94, RU96] is related to our notion of maximal matchings under OR-semantics. However, the work on full disjunctions was couched in the relational model, and the results are not the same as those we have obtained for the semistructured data model. For one, we have established a polynomial-time complexity in the size of the input and output even for dag queries, while from the results of [RU96] it only follows that full disjunctions can be computed in polynomial time in the size of the input and output when the relations are γ-acyclic. Moreover, we have also investigated other semantics, and introduced a two-phase evaluation process, consisting of search constraints and filter constraints which is more expressive than outerjoins. We have also investigated containment of search queries under the various semantics.

In a project at Hebrew University, we have designed and implemented a language based on the ideas expounded here. The language is part of a system to facilitate the access to the World Wide Web. As described abstractly in this paper, queries are based on so-called query graphs, which have to be matched against the database graph. In our implementation, query graphs are edited with a graphical user interface. Thus, they allow for more intuitive query formulation than the text based query languages proposed so far for semistructured data.

We have deliberately limited our investigation to queries that do not allow regular path expressions. Regular expressions present an additional difficulty, one of the reasons being that they cannot be modeled in first order logic. As a consequence, reasoning problems like equivalence and containment for such a language have a significantly higher complexity than in the case studied here. However, any practical query language for semistructured data will need regular path expression. How to use them in a language that allows for incomplete answers is an important and challenging research problem.

Acknowledgments

This research was supported in part by the Esprit Long Term Research Project 22469 "Foundations of Data Warehouse Quality" (DWQ), and by Grants 8528-95-1 and 9481-1-98 of the Israeli Ministry of Science.

References

[Abi97] S. Abiteboul. Querying semi-structured data. In F.N. Afrati and Ph. Kolaitis, editors, *International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18, Delphi (Greece), January 1997. Springer-Verlag.

[AM98] G.O. Arocena and A.O. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proc. 14th International Conference on Data Engineering*, pages 24–33, Orlando (Florida, USA), February 1998. IEEE Computer Society.

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To weave the Web. In *Proc. 23nd International Conference on Very Large Data Bases*, pages 206–215, Athens (Greece), August 1997. Morgan Kaufmann Publishers.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[AV97a] S. Abiteboul and V. Vianu. Queries and computation on the web. In F.N. Afrati and Ph. Kolaitis, editors, *International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 122–133, Delphi (Greece), January 1997. Springer-Verlag.

[AV97b] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. 16th Symposium on Principles of Database Systems*, pages 122–133, Tucson (Arizona, USA), May 1997. ACM Press.

[BDFS97] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In F.N. Afrati and Ph. Kolaitis, editors, *International Conference on Database Theory*, pages 336–350, Delphi (Greece), January 1997. Springer-Verlag.

[BDHS96] P. Buneman, S.B. Davidson, G.G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal (Canada), June 1996.

[BFW98] P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured and structured databases. In *Proc. 17th Symposium on Principles of Database Systems*, pages 129–138, Seattle (Washington, USA), June 1998. ACM Press.

[BG81] P.A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981.

[Bun97] P. Buneman. Semistructured data. In *Proc. 16th Symposium on Principles of Database Systems*, pages 117–121, Tucson (Arizona, USA), May 1997. ACM Press.

[CAW98] S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proc. 14th International Conference on Data Engineering*, pages 4–13, Orlando (Florida, USA), February 1998. IEEE Computer Society.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th Annual ACM Symposium on Theory of Computing*, 1977.

[CMW82] I.S. Cruz, A.O. Mendelzon, and P.T. Wood. A graphical query language supporting recursion. In *Proc. 1982 International Conference on Management of Data*, pages 323–330, Orlando (Florida, USA), June 1982.

[FFK+98] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data*, pages 414–425, Seattle (Washington, USA), June 1998. ACM Press.

[FLS98] D. Florescu, A.Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. 17th Symposium on Principles of Database Systems*, pages 139–148, Seattle (Washington, USA), June 1998. ACM Press.

[FPS97] M.F. Fernandez, L. Popa, and D. Suciu. A structure-based approach to querying semi-structured data. In *6th International Workshop on Database Programming Languages*, Estes Park (Colorado, USA), August 1997. Springer-Verlag.

[GL94] C.A. Galindo-Legaria. Outerjoins as disjunctions. In *Proc. 1994 ACM SIGMOD International Conference on Management of Data*, pages 348–358, Minneapolis (Minnesota, USA), May 1994. ACM Press.

[GW97] R. Goldman and J. Widom. Dataguides: enabling query formulation and optimization in semistructured databases. In *Proc. 23nd International Conference on Very Large Data Bases*, Athens (Greece), August 1997. Morgan Kaufmann Publishers.

[HMV96] M.Z. Hasan, A.O. Mendelzon, and D. Vista. Applying database visualization to the world wide web. *SIGMOD Record*, 25(4):45–49, 1996.

[KMSS97] Y. Kogan, D. Michaeli, Y. Sagiv, and O. Shmueli. Utilizing the multiple facets of WWW contents. In *Proc. 2nd International Workshop on Next Generation Information Technologies and Systems*, Neve Ilan (Israel), July 1997.

[KMSS98] Y. Kogan, D. Michaeli, Y. Sagiv, and O. Shmueli. Utilizing the multiple facets of WWW contents. *Data and Knowledge Engineering*, 28(3):255–275, 1998.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the world-wide web. In *Proc. 21st International Conference on Very Large Data Bases*, pages 54–65. Morgan Kaufmann Publishers, August 1995.

[KS97] D. Konopnicki and O. Shmueli. W3QS—A system for WWW querying. In *Proc. 13th International Conference on Data Engineering*, page 586, Birmingham (United Kingdom), April 1997. IEEE Computer Society.

[LSS96] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. A declarative language for querying and restructuring the web. In *Proc. 6th International Workshop on Research Issues on Data Engineering - Interoperability of Nontraditional Database Systems*, pages 12–21, New Orleans (Louisiana, USA), February 1996. IEEE Computer Society.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 3(26):54–66, 1997.

[MAM+98] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. The Araneus web-base management system. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data*, pages 544–546, Seattle (Washington, USA), June 1998. ACM Press.

[MM97] A.O. Mendelzon and T. Milo. Formal models of web queries. In *Proc. 16th Symposium on Principles of Database Systems*, pages 134–143, Tucson (Arizona, USA), May 1997. ACM Press.

[MMM97] A.O. Mendelzon, G.A. Mihaila, and T. Milo. Querying the world wide web. *International Journal on Digital Libraries*, 1(1):54–67, 1997.

[NAM98] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data*, pages 295–306, Seattle (Washington, USA), Seattle (Washington, USA) 1998. ACM Press.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In P.S.Yu and A.L.P. Chen, editors, *Proc. 11th International Conference on Data Engineering*, pages 251–260, Taipei, March 1995. IEEE Computer Society.

[QRS+94] D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, and J. Widom. Querying semistructured heterogeneous information. Unpublished memorandum, CSD, Stanford University, 1994.

[QWG+96] D. Quass, J. Widom, R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, A. Rajaraman, H. Rivero, S. Abiteboul, J.D. Ullman, and J.L. Wiener. Lore: A lightweight object repository for semistructured data. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, page 549, Montreal (Canada), June 1996.

[RU96] A. Rajaraman and J.D. Ullman. Integrating information by outerjoins and full disjunctions. In *Proc. 15th Symposium on Principles of Database Systems*, pages 238–248, Montreal (Canada), June 1996. ACM Press.