Quering Incomplete Information in Semistructured Data

Yaron Kanza* Werner Nutt† Yehoshua Sagiv*

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. 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 constraints on these variables. An answer is a binding of the variables by elements of the database such that the constraints 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 constraints: under some circumstances we consider constraints 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 constraints 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.

*Institute for Computer Science, The Hebrew University, Jerusalem 91904, Israel, {yarok, sagiv}@cs.huji.ac.il

†Department of Computing and Electrical Engineering Heriot-Watt University Riccarton Campus Edinburgh EH14 4AS nutt@cee.hw.ac.uk

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, CGd99] 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. If the pair of nodes (n_1, n_2) is in the relation l^G, then we say that there is an *edge* with label l from n_1 to n_2, and we call n_1 the *source* and n_2 the *target* of that edge. We will often view a binary relation l^G as a function $l^G: N \to 2^N$. 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 $G = (N, r_G, \cdot^G)$. In the examples, the root node is always highlighted by a name. 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 $D = (O, r_D, \cdot^D, \alpha)$.

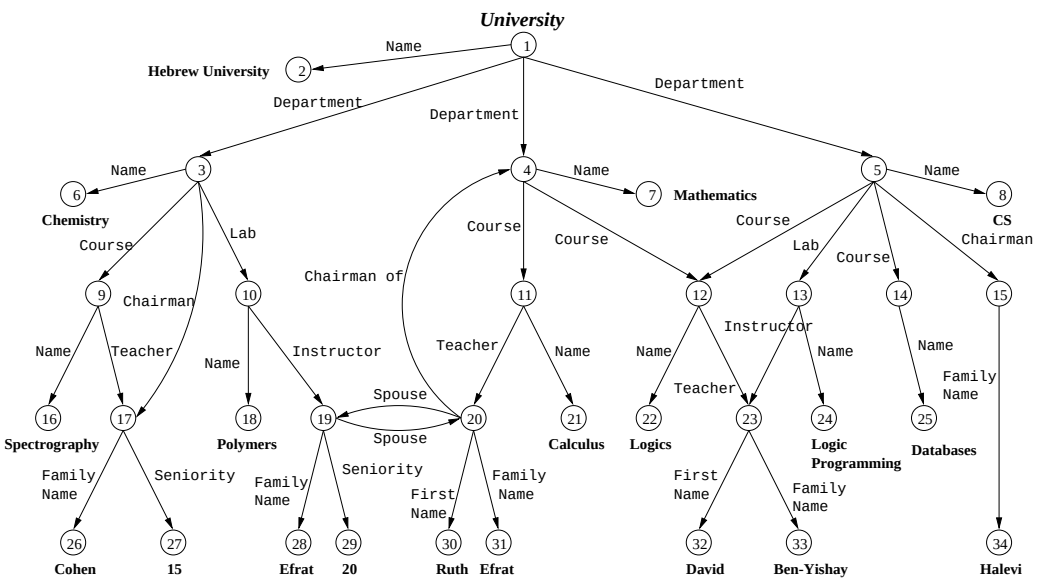

Figure 1: A university database

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.

In general, 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

In 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 occurring 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 $Cons(G)$ 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 $Cons(G)$ 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 the well-known conjunctive queries. We also found that representing queries as sets of constraints eases the development and the description 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.

It is convenient for our formalism to extend the edge functions l^D from database nodes to the value $\bot$ by defining $l^D(\bot) := \emptyset$.

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 $\mu(v) \neq \perp$. By analogy, we call a prematching also an OR-*matchings,* since it satisfies *some* incoming constraint of v whenever $\mu(v) \neq \perp$.

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 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

$$Mat^s_D(Q) \subseteq Mat^\wedge_D(Q) \subseteq Mat^w_D(Q) \subseteq Mat^\vee_D(Q). \tag{1}$$

If the query graph is a tree, then there is no distinction between weak, AND, and OR-matchings. For $\sigma \in \{s, \wedge, w, \vee\}$ we also refer to $Mat^\sigma_D(Q)$ 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 $\sigma \in \{s, \wedge, w, \vee\}$ we thus define $MMat^\sigma_D(Q)$ as the set of maximal elements of $Mat^\sigma_D(Q)$. 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 $Mat^\sigma_D(Q) \subseteq Mat^\tau_D(Q)$, then for every $\mu \in MMat^\sigma_D(Q)$ there is a $\mu' \in MMat^\tau_D(Q)$ 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.

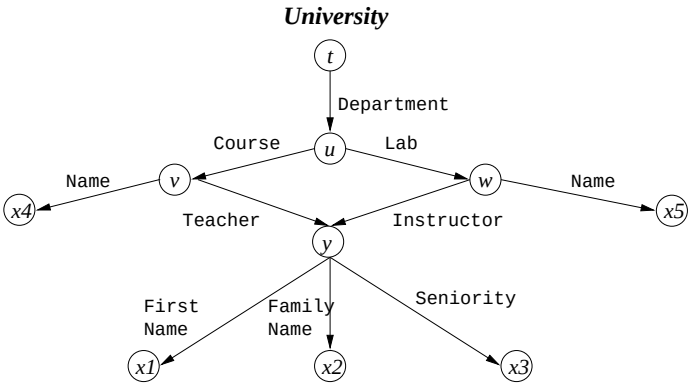

Figure 2: Query graph G_1 asking for course teachers and lab instructors

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

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

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

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.

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 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 a terminal 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. For weak satisfaction, we no longer insist that an assignment be defined for all variables occurring in the constraint. However, if the assignment is defined for all variables in a constraint, then the values must be in the constraint relation. Formally, 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 $\mu(v) \neq \perp$. 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 $\sigma \in \{s, \wedge, w, \vee\}$ we define the set of strong, AND, weak and OR-*solutions* of Q over D as

$$Sol_D^\sigma(Q) := \Big\{\mu \in MMat_D^\sigma(Q) \;\Big|\; \mu \models_s F_s \text{ and } \mu \models_w F_w\Big\}.$$

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 $Sol_D^\sigma(Q)$, which depends on the semantics σ.

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 $Ans_D^\sigma(Q)$, where $\sigma \in \{s, \wedge, w, \vee\}$.

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 in his responsibility.

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, as before, but in addition, we would receive two more kinds of answers. We would also get information about a person together with information about a course he *does* teach, and a lab

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

in his department which he *does not* instruct, or, likewise, together with information about a lab he *does* instruct, and a course in his department which he *does not* teach. For instance, under OR-semantics, David Ben-Yishai together with the Logic Programming Lab and the course on Databases would 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.

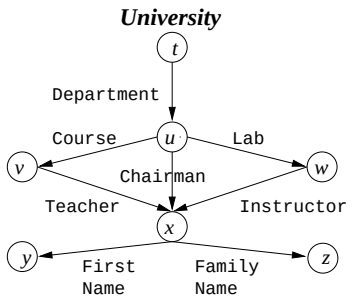

Figure 3: Query graph G_2 asking for the staff in the university database

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.

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 a prematching, the variable u must be bound if x is bound, and under weak semantics, 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.

Semantics	t	u	v	w	x	y	z
OR	1	3	9	10	17	$\perp$	Cohen
	1	3	9	10	19	$\perp$	Efrat
	1	4	11	$\perp$	20	Ruth	Efrat
	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

3.5 Incomplete Answers Versus Regular Path Expressions

Some query languages for semistructured data use regular expressions to deal with incompleteness in the data (e.g. [FFK+98, MAG+97, KS95]). In this section, we discuss the similarities and differences between our approach and the approach of using regular expressions to deal with incompleteness in the data.

First, we give some definitions. The *l*-sequence of a given path ϕ is the sequence of labels along the path. A *regular-path query* is a query that consists of a regular expression. Suppose that Q_ϵ is a regular-path query for the regular expression ϵ. The result of applying Q_ϵ to a database D is the set of all database nodes o, such that there is a path ϕ from the root of D to o and the *l*-sequence of ϕ is in the language of ϵ. A discussion of regular-path queries and their evaluation can be found in [ABS00].

There is some resemblance between regular-path queries and queries in the OR-semantics. In particular, a query Q expressed in the OR-semantics can be emulated by a regular-path query Q', provided that Q has only one distinguished variable, say x. To construct Q', we first construct a regular expression ϵ, such that the language of ϵ is exactly the set of the *l*-sequences of all the paths from the root of Q to x

(see [MW95] for a discussion on how to construct ϵ). Q' is the regular-path query Q_ϵ for the regular expression ϵ. The evaluation of Q and Q_ϵ over any given database will produce the same result.

Regular-path queries can reveal the set of database nodes that will be assigned to each variable of a given query under the OR-semantics. However, given a query in the OR-semantics with more than one distinguished variable, the usage of regular-path queries does not reveal the OR-matchings. Note that if we have for each query variable v, the set of database nodes that are assigned to v, then we may take the Cartesian product of all these sets. But the set of OR-matchings is a subset of this product, and regular-path queries cannot find this exact subset. Thus, even with regular-path queries, one cannot fully express queries in either the AND, OR, or weak semantics.

4 Computing Matchings

In this section we study how to compute 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.

For any syntactical object X, we denote by $|X|$ the size of X. Thus, $|Q|$, $|D|$, and $|S|$, stand for the size of the query Q, the database D, and the set of assignments S, respectively.

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,

Algorithm *EvalDagQuery*(Q, D, σ);
Input dag query Q, database D, semantics $\sigma \in \{\wedge, \vee\}$;
Output the set of maximal σ-matchings of Q over D;

let $v_0 < v_1 < \ldots < v_k$ be a topological order on the variables of Q;
let $\mu_0 := [v_0/r_D]$;
let $S_0 := \{\mu_0\}$;
for $i = 1$ to k **do**
    $S_i := \emptyset$;
    for each $\mu \in S_{i-1}$ **do**
        let $E := Ext^{\sigma}_{\mu}(v_i)$
        if $E = \emptyset$
        then
            $S_i := S_i \cup \{\mu \oplus [v_i/\perp]\}$
        else
            $S_i := S_i \cup \{\mu \oplus [v_i/o] \mid o \in E\}$
return S_k

Figure 4: Computing maximal matchings for a dag query

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) *If the algorithm EvalTreeQuery is called with a tree query Q and a database D, then its output S is the set of maximal matchings under* AND, OR, *and weak semantics. The runtime of the algorithm is $O(|D| * |Q| * |S|)$.*

4.2 Acyclic Queries

We now generalize the algorithm *EvalTreeQuery* to algorithms for dag queries. In a dag query, there may be more than one incoming constraint for a variable, and consequently, the matchings vary according to the different semantics under which the query is interpreted. We first describe an algorithm *EvalDagQuery*, which has an argument specifying whether the query is to be evaluated under AND or OR-semantics. Then we give an algorithm for weak semantics.

4.2.1 AND- and OR-Semantics

The algorithm *EvalDagQuery* is presented in Figure 4. We give an overview of the algorithm and explain the notation used in writing it up.

In the course of computation, *EvalDagQuery* extends and reduces assignments. If μ is an assignment, v is a variable that is not yet bound by μ, and o is a database object, then $\mu \oplus [v/o]$ is the assignment that binds v to o in addition to the variables already bound by μ. If μ is an assignment, v is a variable and μ binds v to an object o then $\mu \ominus [v/o]$ is the assignment that binds all variables, except v, the same as μ and binds v to $\perp$.

Similarly as the algorithm for tree queries, *EvalDagQuery* starts out by arranging the variables in the query in a topological order $v_0 < v_1 < \ldots < v_k$. With Q_i we denote the query whose graph is the subgraph of the query graph of Q that has only the nodes $\{v_0, \ldots, v_i\}$. In an iteration over the variables, *EvalDagQuery* computes for each $i \in 1..k$ the set of maximal matchings for the query Q_i by extending the maximal matchings for the query Q_{i-1}.

The central operation is to compute for a maximal matching μ for Q_{i-1} the set of database objects o such that $\mu \oplus [v_i/o]$ is a maximal matching of Q_i. Of course, this operation differs, depending on whether AND or OR-matchings are to be computed. Suppose the database D over which we evaluate Q ranges over the set of objects O. Moreover, let μ be a D-assignment over $\{v_0, \ldots, v_{i-1}\}$. Then for $\sigma \in \{\wedge, \vee\}$, we define $Ext^{\sigma}_{\mu}(v_i)$ as the set of objects o such that extending μ by binding v_i to o yields a σ-matching of Q_i, that is, formally,

$$Ext^{\sigma}_{\mu}(v_i) := \Big\{ o \in O \;\Big|\; \mu \oplus [v_i/o] \in Mat^{\sigma}_{D}(Q_i) \Big\}.$$

The following proposition, which follows immediately from the definitions, provides characterizations of the sets $Ext^{\sigma}_{\mu}(v_i)$.

Recall that if D is a database and l is a label, then $l^D(\perp) = \emptyset$.

Proposition 4.2 *Let Q be a dag query, and let $v_0 < v_1 < \ldots < v_k$ be a topological order on the variables of Q. Moreover, let D be a database and let μ be a D-assignment over $\{v_0, \ldots, v_{i-1}\}$. Then*

$$\begin{aligned} Ext^{\wedge}_{\mu}(v_i) &= \bigcap_{vlv_i \in Cons(Q)} l^D(\mu(v)) \\ Ext^{\vee}_{\mu}(v_i) &= \bigcup_{vlv_i \in Cons(Q)} l^D(\mu(v)). \end{aligned}$$

Proof. For a AND-matching μ, an object o is in $Ext^{\wedge}_{\mu}(v_i)$ iff $\mu' := \mu \oplus [v_i/o]$ satisfies all incoming edge constraints of $\mu(v_i)$, that is, μ' satisfies all edge constraints in $Cons(Q)$ of the form vlv_i, that is, o is an element of all sets $l^D(\mu(v))$, where $vlv_i \in Cons(Q)$. This proves the first equality.

The second equality is proved with an analogous argument. □

Corollary 4.3 *For a dag query Q, a database D, an assignment μ and a variable v, the set $Ext^{\sigma}_{\mu}(v_i)$ can be computed in time $O(|Q| \cdot |D| \cdot \log |D|)$ for $\sigma \in \{\wedge, \vee\}$.*

Proof. When computing $Ext^{\sigma}_{\mu}(v_i)$, each edge in D is touched at most as often as there are edge constraints in Q. Intersections and unions of sets of database objects can be computed in time $O(|D| \cdot \log |D|)$. □

Proposition 4.4 (Soundness and Completeness) *If called with a dag query Q, a database D, and a type $\sigma \in \{\wedge, \vee\}$, then the algorithm EvalDagQuery(Q, D, σ) returns $MMat^{\sigma}_{D}(Q)$.*

Proof. Let $v_0 < v_1 < \ldots < v_k$ be a topological order on the variables of Q. We define Q_i to be the query graph with node variables $v_0, \ldots, v_i$, with root v_0, and such that two variables are connected by label l in Q_i if and only if they are connected by l in Q. For Q_0 the set S_0 contains only the root assignment and hence S_0 is $MMat^{\sigma}_{D}(Q_0)$. We now continue by induction. We assume that for Q_i the set S_i equals $MMat^{\sigma}_{D}(Q_i)$. Then for Q_{i+1} the set S_{i+1} includes all possible extensions of the assignments in S_i to the variable v_{i+1}, since the extensions are based on the set $Ext^{\sigma}_{\mu}(v_i)$. Hence, S_{i+1} equals $MMat^{\sigma}_{D}(Q_{i+1})$. □

As for *EvalTreeQuery*, the run time of *EvalDagQuery* is polynomial in the size of the input and the result.

Proposition 4.5 (Complexity) *Suppose EvalDagQuery is called with a dag query Q and a database D, and outputs the set of assignments S. Then the runtime of the algorithm is $O(|D| \cdot \log |D| \cdot |Q|^2 \cdot |S|)$.*

Proof. The algorithm essentially runs $O(|Q|)$ times through the outer loop. We show that each run through the loop takes time at most $O(|D| \cdot \log |D| \cdot |Q| \cdot |S|)$.

Obviously, $|S_{i-1}| \leq |S_i|$ for all $i \in 1..k$, and thus $|S_i| \leq |S|$. In the i-th run through the loop, for each assignment $\mu \in S_{i-1}$, the set $Ext^{\sigma}_{\mu}(v_i)$ is computed. By Corollary 4.3, this can be done in time $O(|D| \cdot \log |D| \cdot |Q|)$. Since there are at most $|S|$ assignments, this yields the claim. □

4.2.2 Weak Semantics

In order to compute the maximal matchings of a dag query in weak semantics, we use the algorithm *EvalWeakDagQuery* in Figure 5.

The algorithm calls the function *ExtendAssignment* described in Figure 6. Given an assignment μ, *ExtendAssignment* assigns a new database object to a previously unbound variable, say v, producing an assignment μ'. Since the newly bound variable can participate in constraints wlv whose variables are bound, but are not satisfied by μ', such constraints must be "deactivated" by changing the binding of those w's to $\perp$. The changed assignment may no more be a prematching. Therefore, variables that are no more reachable from the root through satisfied constraints are also reset to $\perp$.

Algorithm *EvalWeakDagQuery*(Q, D);
Input dag query Q, database D;
Output the set of maximal weak matchings of Q over D;

let $v_0 < v_1 < \ldots < v_k$ be a topological order on the variables of Q;
let $\mu_0 := [v_0/r_D]$;
let $S_0 := \{\mu_0\}$;
for $i = 1$ to k **do**
    $S_i := \emptyset$;
    for each $\mu \in S_{i-1}$ **do**
        $S_i' := S_i \cup \{\mu \oplus [v_i/\perp]\}$
        $S_i'' := S_i' \cup \{ExtendAssignment(Q, D, \mu, v_i, o) \mid o \in Ext_\mu^\vee(v_i)\}$
    $S_i :=$ the maximal elements of S_i'';
return S_k

Figure 5: Computing maximal matchings of a dag query in weak semantics

Algorithm *ExtendAssignment*(Q, D, μ, v, o);
Input dag query Q, database D, partial assignment μ,
    query variable v, database node o;
Output a weak matching created from μ by adding $[v/o]$;

let $\mu' := \mu \oplus [v/o]$;
for each edge ulv in Q **do**
    if μ' does not satisfy ulv **then**
        $\mu' := \mu' \ominus [u/\mu'(u)] \oplus [u/\perp]$;
while there is a node w in Q, where $w \neq r_Q$
        such that μ' is undefined for all parent nodes of w **do**
    $\mu' := \mu' \ominus [w/\mu'(w)] \oplus [w/\perp]$;
if $\mu'(r_Q) = \perp$ **then**
    $\mu'(r_Q) := r_D$;
return μ';

Figure 6: Adding a node assignment to a dag query assignment in weak semantics

Proposition 4.6 (Soundness and Completeness) *If called with a dag query Q and a database D, the algorithm EvalWeakDagQuery(Q, D) returns $MMat_D^w(Q)$.*

Proof. The proof is by induction over the number of node variables that have been

processed by the algorithm. Let $v_0, \ldots, v_k$ be the nodes of Q ordered in a topological order. As in the proof of Proposition 4.4, we denote by Q_i the query restricted to the variables $v_0, \ldots, v_i$.

EvalWeakDagQuery starts by assigning the database root to the query root. This is the only element in the set of maximal weak matchings for the one node query Q_0. By induction, we assume that the set S_{i-1} computed by the algorithm is the set of weak maximal matchings of Q_{i-1} over D, that is $S_{i-1} = MMat^w_D(Q_{i-1})$.

Then the algorithm computes S_i by processing each assignment μ in S_{i-1} separately. First, μ is extended by assigning $\perp$ to the variable v_i. Furthermore, for each $o \in Ext^{\vee}_{\mu}(v_i)$, the assignment $ExtendAssignment(Q, D, \mu, v_i, o)$ is added to S_i.

When $\perp$ is assigned to v_i, clearly, the extension $\mu \oplus [v_i/\perp]$ is a weak matching of Q_i, since, by the induction hypothesis, μ is a weak matching of Q_{i-1}, and thus, all edge constraints in Q_i, for which both variables are not null, are satisfied by the extension.

We examine the second case and show that for each object $o \in Ext^{\vee}_{\mu}(v_i)$ also $\mu' = ExtendAssignment(Q, D, \mu, v_i, o)$ is a weak matching of Q_i. A quick check of *ExtendAssignment* reveals that μ' assigns the database root to the query root. This is true when μ' is assigned $\mu \oplus [v/o]$, since μ already does so. If $\mu'(r_Q)$ is changed to $\perp$ during the execution of the function, then it is restored to the root of D at the end of the computation.

We also need to show that μ' is a prematching of Q_i. This means that in Q_i each node that is bound by μ' is reachable in the graph of Q_i by a path from the root such that each constraint in this path is satisfied by μ'. If μ is defined for a node u of Q_{i-1}, then, by the induction hypothesis, u is reachable in Q_{i-1} by a path from the root such that each constraint on this path is satisfied by μ. However, μ' need not be defined for u because *ExtendAssignment* assigns $\perp$ to certain variables that were bound by μ. In the first phase, *ExtendAssignment* undoes assignments to those parent nodes of v_i that participate in edge constraints which were not satisfied by μ'. In the second phase, it recursively undoes bindings of variables that, due to previous undoing, do not have bound parents and therefore are no more reachable from the query root. Thus, after the second phase is completed, μ' is a prematching.

We now show that μ' is a weak matching. To see this, let xly be any constraint in Q_i for which $\mu(x)$ and $\mu(y)$ are not null. If y is not v_i, then both x and y are in Q_{i-1} (x cannot be v_i because of the topological order). In this case, μ satisfies xly and hence μ' satisfies xly. If y is v_i, then xly must be satisfied, otherwise μ' would assign $\perp$ to x.

Summarizing, at the end of a loop, S_i is a set of weak matchings of Q_i over D. Although we obtain S_i by removing subsumed matchings, this does not show yet that the elements of S_i are maximal. Soundness will follow from this only together with completeness.

We prove completeness by showing inductively that $S_i \supseteq MMat^w_D(Q_i)$. For S_0 the statement is obvious. For $i > 0$, suppose that μ' is a maximal weak matching of Q_i. There are two cases, depending on whether $\mu'(v_i)$ is $\perp$ or not.

If $\mu'(v_i) = \perp$, let μ be the restriction of μ' to the variables $v_0, \ldots, v_{i-1}$. Then μ is a weak matching of Q_{i-1}. Since μ' is maximal, so is μ. By the induction hypothesis, μ is in S_{i-1}, and thus *EvalWeakDagQuery* includes $\mu' = \mu \oplus [v_i/\perp]$ in S_i.

Suppose now that $\mu'(v_i) = o$ for some object o. The restriction of μ' to the variables $v_0, \ldots, v_{i-1}$ yields a weak matching $\tilde{\mu}$ of Q_{i-1}, which however may not be maximal. Let μ be a maximal weak matching of Q_{i-1} that subsumes $\tilde{\mu}$. Since μ' is a prematching, there is an edge constraint $v_j l v_i$ that is satisfied by μ'. Because the order on variables is topological, we have $j < i$, and thus $\mu(v_j) = \mu'(v_j)$. We infer that $o \in l^D(\mu(v_j))$ and hence, $o \in Ext^{\vee}_{\mu}(v_i)$. Thus, the algorithm calls *ExtendAssignment*(Q, D, μ, v_i, o).

Essentially, *ExtendAssignment* undoes bindings of variables that were bound by μ. We show that no bindings of μ are undone for variables for which μ' is defined. In the first phase, the algorithm undoes all bindings of parent nodes u of v_i that are not compatible with the binding $[v_i/o]$. Those variables are not bound by μ' because μ' is a weak matching. Due to this erasing of bindings, other nodes of the query graph may lose their connection to the root. Their bindings are erased in the second phase of *ExtendAssignment*. Of course, μ' is not defined for these variables because it is a prematching. We conclude that, since μ' is maximal, the function call *ExtendAssignment*(Q, D, μ, v_i, o) returns in fact μ'.

Thus, each maximal weak matching μ' of Q_i over D is returned. This shows the completeness of the algorithm.

To finish the soundness proof, recall that S_i contains all maximal matchings because of the completeness of the algorithm, and that all elements of S_i are weak matchings by the first part of this proof. Both properties together yield the soundness. □

As for AND and OR-semantics, the run time of *EvalWeakDagQuery* is polynomial in the size of the input and the result.

Proposition 4.7 (Complexity) *If EvalWeakDagQuery is called with a dag query Q and a database D, and outputs the set of assignments S, then the runtime of the algorithm is $O(|D| \cdot \log |D| \cdot |Q|^3 \cdot |S|)$.*

Proof. Let $S_i := MMat^w_D(Q_i)$ be the set of assignments computed during the i-th loop. In the proof we use the fact that $|S_i| \leq |S_k| = |S|$. We show this at the end of the proof.

The algorithm runs $O(|Q|)$ times through the outer loop. For each assignment $\mu \in S_{i-1}$, there is an inner loop during which (1) μ is extended by $\perp$, (2) $Ext^{\vee}_{\mu}(v_i)$ is computed, and (3) the function *ExtendAssignment* is called. The extension of μ takes time $O(|Q|)$, the computation of $Ext^{\vee}_{\mu}(v_i)$ takes time $O(|Q| \cdot |D| \cdot \log |D|)$ by Corollary 4.3, and a call to *ExtendAssignment* takes time $O(|Q|^2)$. Thus, each run through the inner loop takes time at most $O(|D| \cdot \log |D| \cdot |Q|^2)$. Since there are $O(S)$ runs, this yields the claim.

It remains to prove that $|S_i| \leq |S_k| = |S|$. We do so by proving that $|S_{i-1}| \leq |S_i|$. Let S_i'' be the set of all assignments constructed during the i-th loop, that is, before reducing them to the maximal elements. For each $\mu \in S_{i-1}$, the set S_i'' contains $\tilde{\mu} := \mu \oplus [v_i/\bot]$. The assignments $\tilde{\mu}$ need not be elements of S_i, but for each $\tilde{\mu}$, there is a $\hat{\mu} \in S_i$ that subsumes it.

Suppose there is a $\nu \in S_{i-1}$ such that $\tilde{\nu} \sqsubseteq \hat{\mu}$, where $\tilde{\nu} = \nu \oplus [v_i/\bot]$. We show that $\mu = \nu$. To see this, suppose first that $\hat{\mu} = \tilde{\mu}$. Then $\tilde{\nu} \sqsubseteq \hat{\mu}$ implies $\nu \sqsubseteq \mu$ and thus $\nu = \mu$ because S_{i-1} contains only maximal elements. Next, suppose that $\hat{\mu} \neq \tilde{\mu}$. Then $\hat{\mu} = \mu \oplus [v_i/o]$ for some database object o (in particular, *ExtendAssignment* did not reset any bindings to null). Again, $\tilde{\nu} \sqsubseteq \hat{\mu}$ implies $\tilde{\nu} \sqsubseteq \tilde{\mu}$ and thus $\nu = \mu$. □

Proposition 4.5 and Proposition 4.7 can be combined into one statement.

Theorem 4.8 (Polynomiality) *Computing the set of maximal matchings of a dag query Q over a given database D can be done in runtime polynomial in the input and the output, for all three semantics—*AND, OR, *and weak semantic.*

4.3 General Queries under AND-semantics

Clearly, for any query, 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. The proof uses a reduction of the Hamiltonian path problem. We formalize this problem as follows: a directed graph is pair $G = (N, \gamma)$ where N is a finite set, the nodes of G, and $\gamma: N \to 2^N$ is the function that associates to each node the set of its *successor nodes.* A *Hamiltonian path* for G is a sequence of nodes $n_1, n_2, \ldots, n_k, n_1$ such that

1. each node of N occurs *exactly once* among $n_1, \ldots, n_k$;
2. n_i is a successor of n_{i-1} for each $i \in 2..k$ and n_1 is a successor of n_k.

In other words, the Hamiltonian path is a closed circuit through the graph that touches each node exactly once. Deciding whether a finite directed graph has a Hamiltonian path is an NP-complete problem [GJ79].

Lemma 4.9 *There is a polynomial time algorithm that constructs for each finite directed graph G a database D_G and a query graph Q_G such that the following are equivalent:*

- *G does* not *have a Hamiltonian path;*
- *$MMat^{\wedge}_D(Q)$ contains only the* root assignment *μ_0 that assigns the root of D_G to the root of Q_G and $\bot$ to all the other variables.*

Proof. We construct for a directed graph $G = (N, \gamma)$ a database D_G and a query graph Q_G and show that they have the desired properties. In our construction, we use the three labels `node`, `neql`, and `succ`. Let $N = \{n_1, \ldots, n_k\}$.

For the database D_G, we choose a set of objects $O = \{o_0, o_1, \ldots, o_k\}$. The root of D_G is $r_{D_G} = o_0$. For each o_i, where $i \in 1..k$, we draw an edge labeled `node` from the root to o_i. For every pair o_i, o_j, where $i, j \in 1..k$, $i \neq j$, we draw an edge labeled `neql` from o_i to o_j. We draw a `succ`-edge from o_i to o_j if and only if n_j is a successor of n_i in G. There are no other edges in the database.

For the query graph Q_G, we choose a set of variables $V = \{v_0, v_1, \ldots, v_k\}$. The root of Q_G is $r_{Q_G} = v_0$. For each v_i, where $i \in 1..k$, we draw an edge labeled `node` from the root to v_i. For every pair v_i, v_j, where $i, j \in 1..k$, $i \neq j$, we draw an edge labeled `neql` from v_i to v_j. We draw a `succ`-edge from v_{i-1} to v_i for $i \in 2..k$ and a `succ`-edge from v_k to v_1. In short, the `succ`-edges in the query describe a circuit through the query variables $v_1, \ldots, v_k$.

Obviously, the root mapping μ_0 is an AND-matching for Q_G over D_G. It is also easy to see that a Hamiltonian path in G gives rise to a strong matching, and therefore to an AND-matching different from the root mapping.

Conversely, suppose μ is an AND-matching that is different from μ_0. Then $\mu(v_i) \neq \perp$ for some $i \in 1..k$. Since v_i lies on the circuit of successors, the predecessor of v_i has to be bound to a non-null node as well. Continuing the argument this way, we see that all variables have to be bound to a non-null value, because they are all on the circuit.

Since μ is defined for all variables, and since it is an AND-matching, it satisfies all constraints in Q_G, in particular all `neql`-constraints. As a consequence, distinct variables are bound to distinct nodes. Hence, the sequence of objects $\mu(v_1), \ldots, \mu(v_k), \mu(v_1)$ corresponds to a Hamiltonian path in G. □

From the lemma, we conclude our claim.

Theorem 4.10 *If there is an algorithm that, for arbitrary query graphs Q, computes $MMat^{\wedge}_D(Q)$ in time polynomial in the input and the output, then* PTIME = NP.

Proof. Suppose that there is an evaluation algorithm and a polynomial p such that for every database D and query Q the algorithm computes the set of AND-matchings $S := MMat^{\wedge}_D(Q)$ in time $p(|D|, |Q|, |S|)$. We show that this would yield a polynomial algorithm to decide the Hamiltonian path problem, which is NP-complete.

Let G be a finite directed graph and D_G, Q_G be the database and the graph constructed according to Lemma 4.9. If G does not have a Hamiltonian path, the evaluation algorithm computes $S_0 = \{\mu_0\}$ in time $p(|D_G|, |Q_G|, |S_0|)$. Since all of $|D_G|$, $|Q_G|$, and $|S_0|$ are polynomially related to $|G|$, the size of G, there is a polynomial p' such that our algorithm returns $\{\mu_0\}$ in time $p'(|G|)$ if G does not have a Hamiltonian path.

This would allow us to solve the Hamiltonian path problem in PTIME. For a graph G, we construct in polynomial time D_G and Q_G. Then we start our algorithm

to evaluate Q_G over D_G. After time $p'(|G|)$ we stop the evaluation. If the algorithm has not finished by itself, then we know that $MMat^{\wedge}_{D_G}(Q_G) \neq \{\mu_0\}$, and thus, by Lemma 4.9, that there is a Hamiltonian path. In case it has finished, we inspect the result set S that it has returned. If $S \neq \{\mu_0\}$, we know, again by Lemma 4.9, that there is a Hamiltonian path, and if $S = \{\mu_0\}$, we know, that there is none. □

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 constraints, equality constraints and inequality constraints.

5.1 Adding Existence Constraints

Existence constraints require query nodes to be bound to non-null values in the filtered solutions. This causes parts of the query graph to be evaluated under strong semantics. We show that evaluating *tree queries* with existence constraints remains polynomial in the size of the input and output. Since AND, weak, and OR-semantics coincide for tree queries, this result is independent of any particular semantics. For *dag queries* with existence constraints, however, evaluation is NP-hard under any of the three semantics.

5.1.1 Queries with Existence Constraints

Let $Q = (G, F, \bar{x})$ be a query, where G is a tree and F contains only existence constraints. We say that a node v in G is *enforced* if there is an existence constraint $!v$ in F. Enforced nodes must be bound to a non-null value in any filtered solution. Obviously, in a tree query, for a node to be bound to a non-null value by a matching, all the ancestors of the node must also be bound to non-null values. Let V_s be the set of nodes in G from which there is a—possibly empty—path to an enforced node, i.e., V_s is the set of enforced nodes and their ancestors. Any matching that satisfies F must bind all variables in V_s, but need not bind variables in the complement of V_s. We call the restriction of G to V_s the *strongly evaluated subtree* of G. The reason is that, intuitively, this subgraph of the query graph has to be evaluated under strong semantics.

The set of solutions of Q can now be computed in the following steps:

1. find the strongly evaluated subtree of G;
2. compute the set of strong matchings for the strongly evaluated subtree;
3. extend the strong matchings to the rest of the graph as described in the algorithm for tree-queries in Section 4.

In order to compute the set of strong matchings for the strongly evaluated subtree of a tree query, we need an algorithm that evaluates tree queries under strong-semantics.

We sketch here such an algorithm, which we call *EvalStrongTree*. The algorithm evaluates a tree query in two phases. During the first phase, it creates a binary relation for each edge of the query. The binary relation created for an edge e by *EvalStrongTree* contains the pairs of database objects such that there is an edge in the database between these two objects, and this database edge satisfies the constraint imposed by e. If the source of e is the root of the query, then in every pair in the relation created for e, the first object, obviously, must be the root of the database. The relation for an edge e, thus, can be viewed as the set of all database edges that satisfy e.

The second phase of *EvalStrongTree* computes the join of the relations that were created in the first phase. We compute that join with the algorithm for computing an acyclic join in relational algebra by a semijoin program (full reducer) as in [BG81]. Note that since the query is a tree, the join is acyclic.

Assume that we evaluate a tree query Q over a database D in strong semantics using the algorithm *EvalStrongTree*, and assume that S is the output set of strong matchings. The runtime of *EvalStrongTree* has an upper bound of $O(|Q| * (|D| * \log|D| + |S| * \log|D|))$. This result follows directly from the cost of computing joins using full reducers (see [Ull89]). Thus, the algorithm *EvalStrongTree* has runtime polynomial in the size of the input and output.

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

Theorem 5.1 (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.*

5.1.2 Dag Queries with Existence Constraints

Adding existence constraints did not make the evaluation of *tree queries* any harder. However, as we will show now, it is highly unlikely that there is an efficient algorithm for evaluating *dag queries* with existence constraints.

The *evaluation problem* for a fixed semantics and a class of queries is to decide whether for a database and a query from that class the set of solutions under that semantics is nonempty. We prove in the following that the evaluation problem for dag queries with existence constraints is NP-hard for any of AND, OR and weak semantics. This implies, by an argument as in the proof of Theorem 4.10, that there

is no algorithm that computes the solutions of such queries in time polynomial in the size of the input and output, unless PTIME = NP.

We first show NP-hardness for AND-semantics. We use a reduction of 3SAT. A formula ϕ is in 3CNF if ϕ is a conjunction of clauses $C_1, \ldots, C_m$, where each clause C_i is a disjunction of three literals l_{i1}, l_{i2}, l_{i3}, and a literal is either a propositional letter or the negation of a propositional letter. To decide whether a formula in 3CNF has a satisfying assignment is NP-complete [GJ79].

Lemma 5.2 *Given a 3CNF formula ϕ over a set of propositional letters P, one can construct in polynomial time a dag query Q and a database D such that the set of filter constraints in Q contains only existence constraints, and such that the following are equivalent:*

1. *there is a maximal* AND*-solution for Q over D;*
2. *there is an assignment for the propositional letters in P that satisfies ϕ.*

Proof. Let ϕ be a formula in 3CNF, let $P = \{p_1, \ldots, p_n\}$ be the set of propositional letters appearing in ϕ, and let $\{C_1, \ldots, C_m\}$ be the set of clauses in ϕ. We construct a database D and a query Q in such a way that evaluating the query over the database simulates the assignment of truth values to the propositional letters, and such that maximal AND-solutions are in a one-to-one correspondence with satisfying assignments. As an example, Figure 7 shows the database and the query constructed for $\phi = (p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_4)$.

We first construct D. Let r_D be the root of D. Below r_D, there are three levels of objects, the first representing the clauses, the second the truth assignments satisfying the clauses, and the third the truth values assigned to the propositional letters. More precisely, for each clause C_i, there is an edge labeled $\texttt{clause}_i$ from r_D to a unique object ocl_i. From each object ocl_i there are seven outgoing edges labeled $\texttt{truth-ass}_i$ to seven different objects $oass_{i1}, \ldots, oass_{i7}$, which stand for the seven truth assignments that satisfy the clause C_i. For each letter p_j, there are objects $otrue_j$ and $ofalse_j$, which stand for the truth values that can be assigned to p_j.

We now link the assignment objects $oass_{ik}$ to the truth value objects $otrue_j$ and $ofalse_j$. Let $p_{l_1}, p_{l_2}, p_{l_3}$ be the propositional letters that appear in the literals of C_i. We consider an assignment to $p_{l_1}, p_{l_2}, p_{l_3}$ that satisfies C_i and choose the object $oass_{ik}$ to represent it. If this assignment maps p_{l_m} to TRUE then we draw an edge with label $\texttt{ass-val}_{l_m}$ from $oass_{ik}$ to $otrue_{l_m}$. If it maps p_{l_m} to FALSE then we draw such an edge to $ofalse_{l_m}$. There are no other objects and edges in D.

Next, we construct a dag query Q with existence constraints. The root of Q is r_Q. Similar to the database, the query has three levels of variables. For each clause C_i, there are variables vcl_i and $vass_i$, standing for the clause and the assignment that satisfies it. There is an edge labeled $\texttt{clause}_i$ from r_Q to vcl_i, and an edge labeled $\texttt{truth-ass}_i$ from vcl_i to $vass_i$. For each propositional letter p_j there is a variable $vprop_j$ in Q. We link the assignment variables $vass_i$ to the letter variables $vprop_j$.

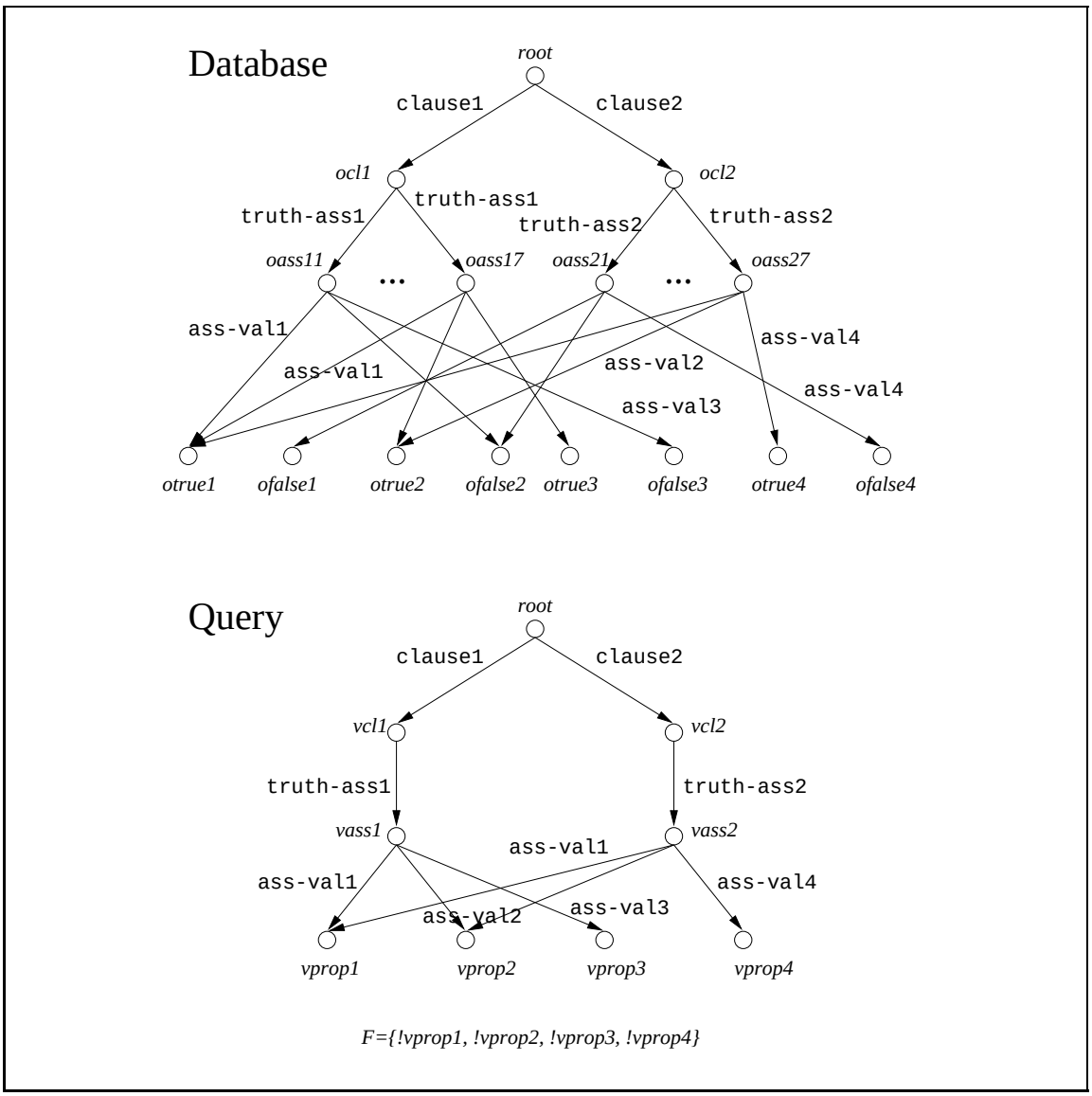

Figure 7: The construction of Lemma 5.2 for $\phi = (p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_4)$

Let $p_{l_1}, p_{l_2}, p_{l_3}$ be the propositional letters that appear in the clause C_i. Then we draw an edge labeled $\texttt{ass-val}_{l_m}$ from $vass_i$ to each variable $vprop_{l_m}$ for $m = 1, 2, 3$. There are no other variables and edges in Q. From the construction it is easy to see that Q is a dag. The set F of filter constraints contains an existence constraint $!vprop_i$ for each variable $vprop_i$ in the query.

We show that there is a one-to-one correspondence between, on the one hand, solutions of Q over D and, on the other hand, truth assignments for the propositions in P that satisfy ϕ. Because of the existence constraints, a solution for Q over D binds each variable $vprop_j$ to a database object. This object is either $otrue_j$ or $ofalse_j$. Thus, a solution defines an assignment of truth values to the propositional letters in

P. Moreover, since the query graph of Q is a dag and the variables $vprop_j$ are the terminal nodes of that dag, an AND-solution for Q binds *all* variables. Because of the structure of D and Q, the variable vcl_i is bound to the clause object ocl_i and $vass_i$ is bound to one of the assignment objects $oass_{ik}$. Since all three constraints $vass_i$ `ass-val`$_j$ $vprop_j$ issuing from $vass_i$ are satisfied by the solution, the clause C_i is in fact satisfied by the corresponding truth assignment. This means that the solution determines an assignment that satisfies ϕ.

Conversely, suppose ρ is a truth assignment that satisfies ϕ. Then ρ gives rise to a solution μ_ρ as follows. The variable $vprop_i$ is mapped to the object $otrue_i$ or $ofalse_i$, depending on whether $\rho(p_i)$ is TRUE or FALSE. The restriction of ρ to the letters in the clause C_i is one of the seven assignments that satisfy C_i. Thus, there is an object $oass_{ik}$ that corresponds to it, and we map $vass_i$ to $oass_{ik}$. The remaining variables in Q are mapped in the obvious way. Obviously, μ_ρ satisfies all edge constraints and the existence constraints of Q, and thus is a solution. □

Theorem 5.3 *Evaluation of dag queries with existence constraints under* AND*-semantics is* NP*-complete.*

Proof. NP-hardness follows from Lemma 5.2. Membership in NP follows because in polynomial time one can guess an assignment to the query variables, verify that it is a matching, verify that it cannot be extended, that is, that it is maximal, and check that it satisfies the existence constraints. Such an assignment is a solution. □

We now show that the evaluation of dag queries in OR and in weak semantics is NP-hard. Again, the proof is based on a reduction of 3SAT, albeit this time different from the one in Lemma 5.2, to take into account the difference in the semantics.

Lemma 5.4 *Given a 3CNF formula ϕ over a set of propositional letters P, one can construct in polynomial time a dag query Q and a database D such that the set of filter constraints in Q contains only existence constraints, and such that the following are equivalent:*

1. *there is a maximal* OR*-solution for Q over D;*
2. *there is an assignment for the propositional letters in P that satisfies ϕ.*

Proof. Let ϕ be a formula in 3CNF, let $P = \{p_1, \ldots, p_n\}$ be the set of propositional letters appearing in ϕ, and let $\{C_1, \ldots, C_m\}$ be the set of clauses in ϕ. We construct a database D and a query Q in such a way that evaluating the query over the database simulates the assignment of truth values to the propositional letters, and such that maximal OR-solutions are in a one-to-one correspondence with satisfying assignments. As an example, Figure 8 shows the database and the query constructed for $\phi = (p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_4)$.

In the database D, there are two levels of objects below the root. The first level represents the literals over P, and the second level the clauses. Links from the first

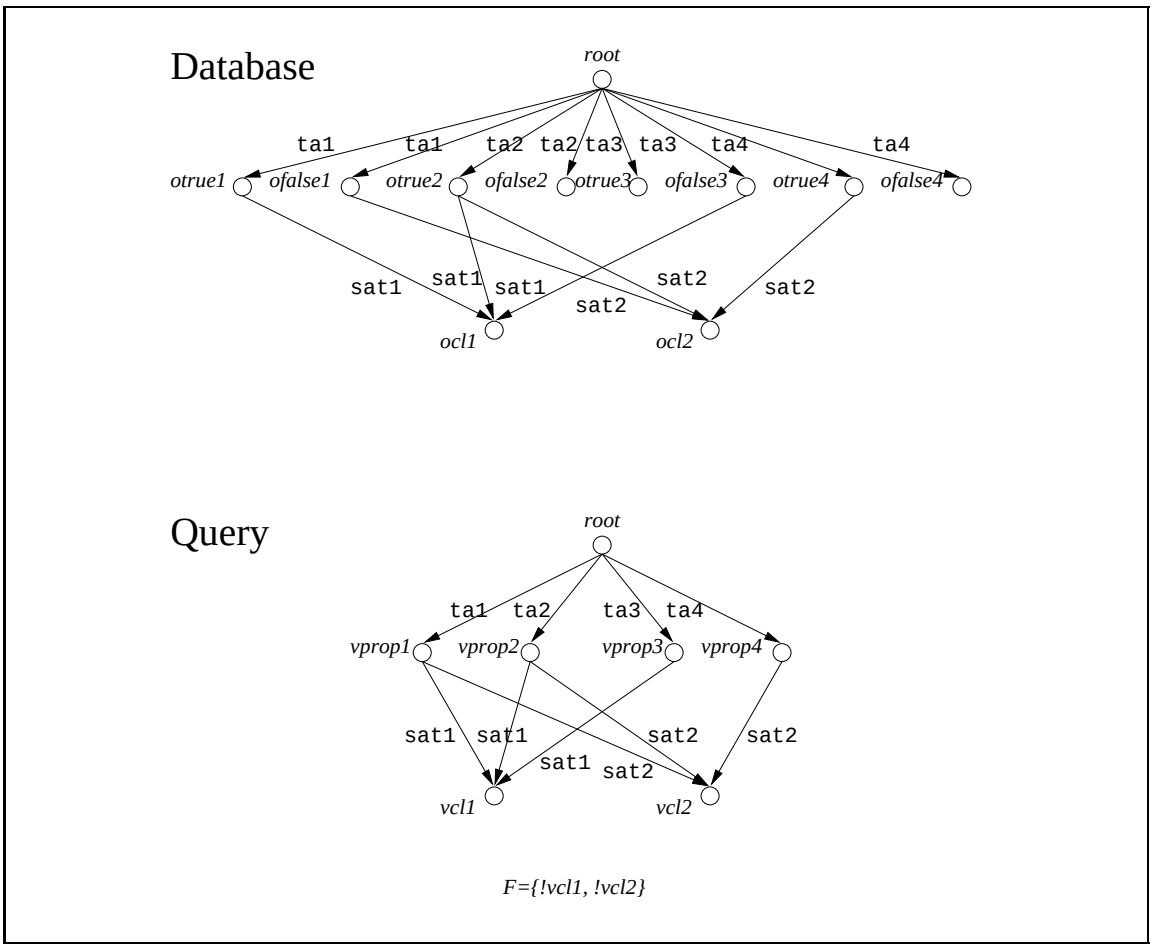

Figure 8: The construction of Lemma 5.4 for $\phi = (p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_4)$

to the second level indicate which literal is present in which clause. Formally, let r_D be the root of D. For each propositional letter p_j, there are objects $otrue_j$ and $ofalse_j$, and edges labeled $\mathtt{ta}_j$ from r_D to these objects. The objects $otrue_j$ and $ofalse_j$ represent the truth values that can be assigned to p_j. For each clause C_i there is a unique object ocl_i in D. If the letter p_j appears as a positive literal in C_i, then we draw an edge labeled $\mathtt{sat}_i$ from $otrue_j$ to ocl_i to indicate that the clause C_i can be satisfied by mapping p_j to TRUE. Similarly, if p_j appears as a negative literal, we draw such an edge from $ofalse_j$ to ocl_i. There are no other edges and nodes in D.

The query Q consists of two similar levels. The root of Q is r_Q. For each propositional letter $p_j \in P$ there is an edge labeled $\mathtt{ta}_j$ from r_Q to a variable $vprop_j$. Because of the label $\mathtt{ta}_j$, the variable can only be bound to $otrue_j$ or to $ofalse_j$. This alternative corresponds to the choice between two truth values for the proposition. For each clause C_i there is a unique variable vcl_i in Q, and for each variable $vprop_j$, such that p_j appears either as a positive or as a negative literal in C_i, there is an edge labeled $\mathtt{sat}_i$ from p_j to vcl_i. There are no other variables and edges in Q. The

set F of filter constraints contains an existence constraint $!vcl_i$ for all the variables vcl_i in the query.

In a solution for Q over D, one of the objects $otrue_j$ or $ofalse_j$ is assigned to the variable $vprop_j$. That corresponds to an assignment of TRUE or FALSE to the propositions in P. Because of the existence constraints, there is a solution for Q over D only if a non-null value is assigned to vcl_i for $i = 1, \ldots, m$. Due to the label $\mathtt{sat}_i$ leading to vcl_i, the only object that can be assigned to it is ocl_i. If vcl_i is bound to ocl_i, then at least one incoming constraint $vprop_j\ \mathtt{sat}_i\ vcl_i$ must be satisfied. That is equivalent to satisfying at least one literal in the clause C_i.

We conclude that there is a one-to-one correspondence between truth assignments to the letters in P and maximal matchings, which assign $otrue_j$ and $ofalse_j$ database objects to $vprop_j$ in Q. A truth assignment to P satisfies ϕ iff every clause of ϕ is satisfied by the assignment iff the corresponding solution assigns a non-null value to each node vcl_i in Q iff there is a solution to Q over D. □

Theorem 5.5 *Evaluation of dag queries with existence constraints under* OR*-semantics is* NP*-complete.*

Proof. NP-hardness follows from Lemma 5.4. Membership in NP follows with arguments similar to those in the proof of Theorem 5.3. □

The same result holds also for weak semantics.

Theorem 5.6 *Evaluation of dag queries with existence constraints under weak semantics is* NP*-complete.*

Proof. For the hardness proof, we modify the construction in Lemma 5.4. That reduction does not work for weak semantics for the following reason. An OR-solution binds all variables in the query, but need not satisfy all edge constraints involving the edges with label $\mathtt{sat}_i$. This only happens if the solution corresponds to an assignment that satisfies all literals of all clauses.

In the modified reduction, we replace each edge labeled $\mathtt{sat}_i$ in the database by a path consisting of two edges, each with that label, and an intermediate new object between the edges. In the query, we similarly replace each edge labeled $\mathtt{sat}_i$ by a path consisting of two edges with the label $\mathtt{sat}_i$ and an intermediate new variable. When not all the incoming constraints of a variable vcl_i are satisfied, that is, when an assignment for a 3CNF formula does not satisfy all the literals in a clause, a null value is assigned to such a new intermediated variable. We then conclude that there is a weak solution to the query over the database iff there is a satisfying assignment to the given 3CNF formula.

Membership in NP is proved with arguments as in Theorem 5.5. □

5.2 Weak Equalities and Inequalities

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.

5.2.1 Adding Weak Equality Constraints

We show that the evaluation problem for tree queries with weak equality constraints is NP-complete by reducing the One-In-All-Pos-Three-3SAT problem. In order to define this problem, we need some notation. Let $P = \{p_1, \ldots, p_n\}$ be a set of propositional letters. A *positive clause* over P is a subset of $\{p_1, \ldots, p_n\}$. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of positive clauses over P. A *traversal* of $\mathcal{C}$ is a truth assignment for P, such that each clause in $\mathcal{C}$ has exactly one true literal.

Now, the One-In-All-Pos-Three-3SAT problem is to decide, given a set of propositional letters P and a set $\mathcal{C}$ of positive clauses over P, each of which has at most three elements, whether there is a traversal of $\mathcal{C}$. It is known that One-In-All-Pos-Three-3SAT is NP-complete [GJ79].

Lemma 5.7 *Given a set P of propositional letters, and a set $\mathcal{C}$ of positive clauses over P, each of which has at most three letters, one can construct in polynomial time a database $D_\mathcal{C}$ and a tree query $Q_\mathcal{C}$, where the filter constraints of $Q_\mathcal{C}$ contain only equality constraints, such that the following are equivalent:*

- *$Q_\mathcal{C}$ has a solution over $D_\mathcal{C}$;*
- *$\mathcal{C}$ has a traversal.*

The equality constraints can either all be value equalities or all be object equalities.

Proof. We first give a reduction with value equalities and then indicate how to modify it in order to yield a reduction with object equalities.

Let $P = \{p_1, \ldots, p_n\}$ be a set of propositional atoms and $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of positive clauses over P. We construct a database $D_\mathcal{C}$ and a query $Q_\mathcal{C}$ such that $D_\mathcal{C}$ encodes the structure of $\mathcal{C}$ and the possible ways to choose elements from each C_i for a traversal. When evaluating the query $Q_\mathcal{C}$, a particular choice has to be made for each clause. The filter constraints of $Q_\mathcal{C}$ will ensure that the local choices from the C_i fit together to yield a traversal. As an example, Figure 9 shows the database and the query constructed for $\mathcal{C} = \{\{p_1, p_2, p_3\}, \{p_2, p_4\}\}$.

Let $r_{D_\mathcal{C}}$ be the root of $D_\mathcal{C}$. For each clause C_i in $\mathcal{C}$ we draw an edge labeled $\texttt{clause}_i$ from $r_{D_\mathcal{C}}$ to a new object ocl_i, which represents the i-th clause C_i. Let $p_{j_{i1}}, p_{j_{i2}}, p_{j_{i3}}$ be the letters in C_i. (Without loss of generality we assume that C_i has exactly three letters. In a case where there are less, the construction has to be modified in an obvious way.) In our example in Figure 7, since $C_1 = \{p_1, p_2, p_3\}$ and $C_2 = \{p_2, p_4\}$, we have $j_{11} = 1$, $j_{12} = 2$, $j_{13} = 3$, and $j_{21} = 2$, $j_{22} = 4$.

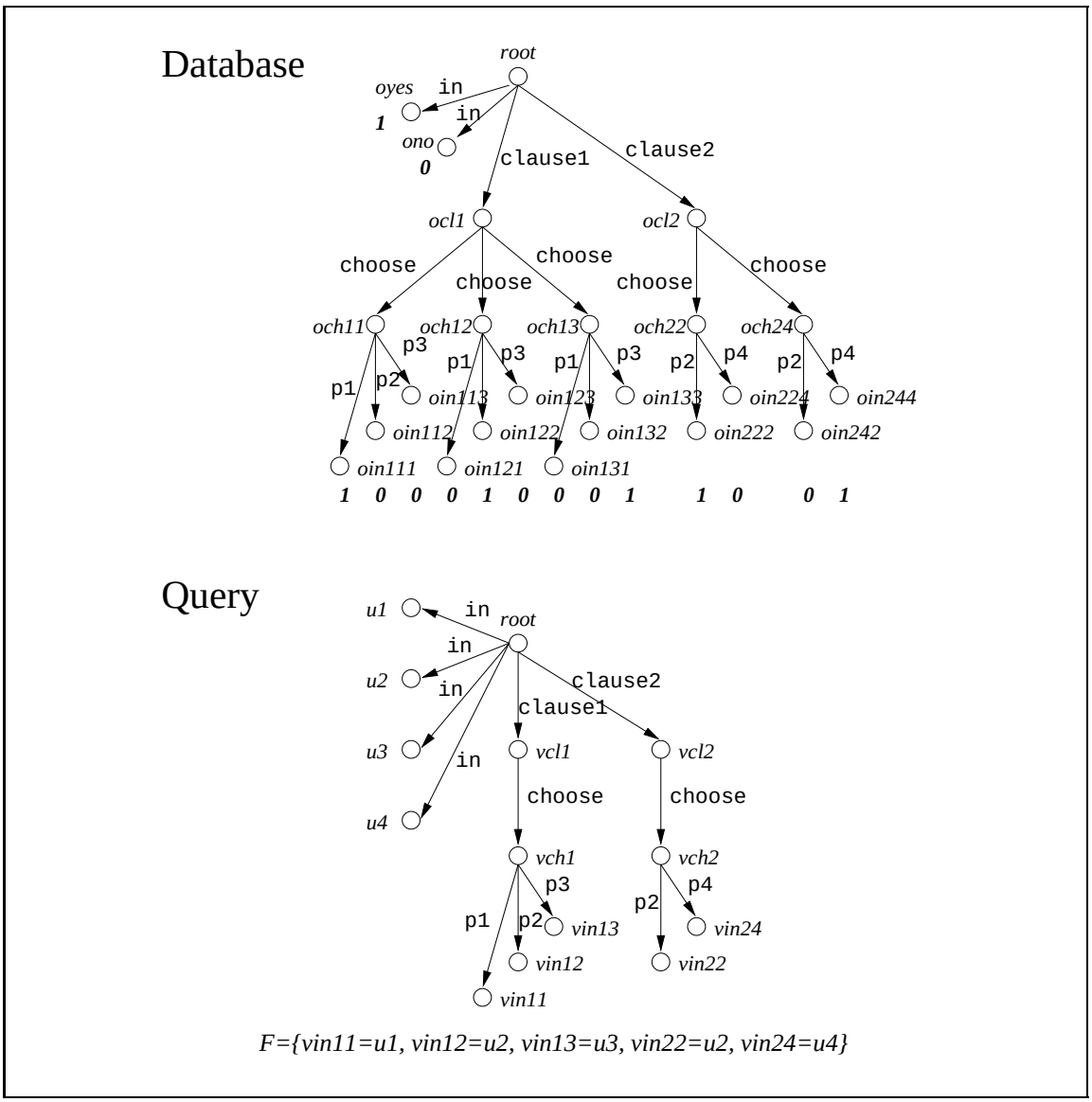

Figure 9: The construction of Lemma 5.7 for $\mathcal{C} = \{C_1, C_2\}$, where $C_1 = \{p_1, p_2, p_3\}$ and $C_1 = \{p_2, p_4\}$.

For each $p_{j_{ik}}$, where $k = 1, 2, 3$, there is an edge labeled `choose` from ocl_i to a new object $och_{i,j_{ik}}$. This object represents the choice of the k-th letter $p_{j_{ik}}$ from C_i. Therefore, in our example, we have objects $och_{1,1}$, $och_{1,2}$, $och_{1,3}$ to represent choosing a variable from C_1 and objects $och_{2,2}$, $och_{2,4}$ to represent choosing a variable from C_2.

Next, we add objects that record which elements of C_i, as a consequence of this choice, are in the traversal and which are not. From $och_{i,j_{ik}}$ we draw three (this depends on the number of elements of C_i) outgoing edges labeled $\mathtt{p}_{j_{il}}$, one for each proposition $p_{j_{il}}$ in C_i, to leaf objects $oin_{i,j_{ik},j_{il}}$. Recall that choosing the object $och_{i,j_{ik}}$ represents the choice of the k-th letter in C_i. Thus, we define this object to have the

value $\alpha(oin_{i,j_{ik},j_{il}}) := \mathbf{1}$ if $k = l$, and the value $\alpha(oin_{i,j_{ik},j_{il}}) := \mathbf{0}$ if $k \neq l$. The value $\mathbf{1}$ symbolizes that the element $p_{j_{ik}}$ is present in the traversal if it is chosen from C_i, and the value $\mathbf{0}$ symbolizes that an element $p_{j_{il}}$, where $k \neq l$, is not present.

In the example, for instance, choosing from C_1 the atom p_2 has consequences as to whether p_1, p_2, p_3 are in the traversal or not. They are represented by three edges, labeled $\mathtt{p}_1$, $\mathtt{p}_2$, $\mathtt{p}_3$, issuing from $och_{1,2}$ to the three objects $oin_{1,2,1}$, $oin_{1,2,2}$, and $oin_{1,2,3}$, and by the values $\mathbf{0}$, $\mathbf{1}$, $\mathbf{0}$ attached to these objects. The values say that p_2 is in the traversal, while p_1, p_3 are not.

There are two more edges emanating from the root. They are labeled `in` and connect the root to two objects *oyes* and *ono*. We also have $\alpha(oyes) = \mathbf{1}$ and $\alpha(ono) = \mathbf{0}$. The role of these objects is to synchronize the choices for the single clauses.

Next, we construct the query $Q_{\mathcal{C}}$. We first describe the query graph G. There are n variables $u_1, \ldots, u_n$, which correspond to the propositions $p_1, \ldots, p_n$, and there is an edge labeled `in` from the root $r_{Q_{\mathcal{C}}}$ to each u_j. Thus, each u_j has to be bound either to *oyes* or to *ono*. This will stand for p_j being in the traversal or not.

For each clause C_i in $\mathcal{C}$ there is an edge labeled $\mathtt{clause}_i$ from $r_{Q_{\mathcal{C}}}$ to a variable vcl_i. Moreover, there is an edge labeled `choose` from vcl_i to a variable vch_i. Thus, vcl_i will be bound to ocl_i, and vch_i will be bound to one of the objects $och_{i,j_{ik}}$, which means that one of the letters $p_{j_{ik}} \in C_i$ has to be chosen for the traversal. (We still assume that $C_i = \{p_{j_{i1}}, p_{j_{i2}}, p_{j_{i3}}\}$.) For each letter $p_{j_{il}} \in C_i$, there is an edge labeled $\mathtt{p}_{j_{il}}$ from vch_i to a variable $vin_{i,j_{il}}$. Thus, $vin_{i,j_{il}}$ has to be bound to $oin_{i,j_{ik},j_{il}}$. Note, that the graph we have constructed here is a tree.

The set F of filter constraints consists of the value equalities $vin_{i,j_{il}} \doteq_v u_{j_{il}}$ for each variable $vin_{i,j_{il}}$, which records whether $p_{j_{il}}$ is chosen from C_i, and the corresponding $u_{j_{il}}$, which records whether $p_{j_{il}}$ is chosen for the traversal.

Summarizing, the query and the database are constructed such that matching the query to the database amounts to two things: first, by matching the variables $u_1, \ldots, u_n$ to *oyes* and *ono*, a subset of P is chosen; second, a maximal matching for the query graph G corresponds to choosing one element of each C_i. Finally, a maximal matching satisfies the filter constraints only if the subset of P chosen is a traversal. Thus, there is a one-to-one correspondence between traversals of $\mathcal{C}$ and solutions of $Q_{\mathcal{C}}$ over $D_{\mathcal{C}}$. This proves the claim for queries with value equalities.

To prove the claim for object equalities, we first modify the database. We replace the object $oin_{i,j_{ik},j_{il}}$ by *oyes* if $\alpha(oin_{i,j_{ik},j_{il}}) = \mathbf{1}$, and by *ono* otherwise. In the query, we replace each value identity by an analogous object identity. Now, we can argue in a similar fashion as before that there is a one-to-one-correspondence between traversals and solutions to the query over the database. □

The preceding lemma implies immediately the desired complexity result. Note that the result does not depend on the semantics under which the search constraints are evaluated, since the query we have constructed is a tree.

Theorem 5.8 (Weak Equalities) *The evaluation problem for tree queries with*

weak equality constraints is NP*-complete. The* NP*-hardness holds both for value and object equalities.*

5.2.2 Adding Weak Equality Constraints

For inequality constraints we prove NP-hardness by a reduction of the graph 3-coloring problem: given a non directed graph, can one assign to each vertex a color such that only three colors are used and distinct colors are assigned to adjacent vertices. It is known that 3-coloring is an NP-complete problem.

Lemma 5.9 *For a finite graph H, one can construct in polynomial time a database D and a tree query Q_H, such that the set of filter constraints in Q_H contains only inequality constraints, and such that there is a maximal solution for Q_H over D iff there is a 3-coloring of H.*

Proof. Figure 10 shows the reduction for an example graph H. The database D is independent of the particular graph. It consists of the root and three terminal objects with the atomic values **red**, **blue**, and **green**. The graph G_H of the query Q_H consists of the root and, for every node n_i of H, a variable v_i, which is a terminal node, as shown in the example. Whenever two nodes n_i, n_j are connected in H, we add a weak value inequality $v_i \not=_v v_j$ to the filter constraints F_H.

Now, it is obvious that every 3-coloring of H gives rise to a solution of Q_H over D. Conversely, note that every maximal matching for Q_H over D is a total assignment that assigns to each variable v_i one of the objects from o_1, o_2, or o_3. Since a solution is a maximal matching that in addition satisfies the filter constraints, it gives rise to a 3-coloring.

Clearly, the reduction also works if we replace the value inequalities by object inequalities. □

Noting that solutions can be guessed and verified in polynomial time, we conclude from the preceding lemma an NP-completeness result.

Theorem 5.10 (Weak Inequalities) *The evaluation problem for tree queries with weak inequality constraints is* NP*-complete. The* NP*-hardness holds both for value and object inequalities.*

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.

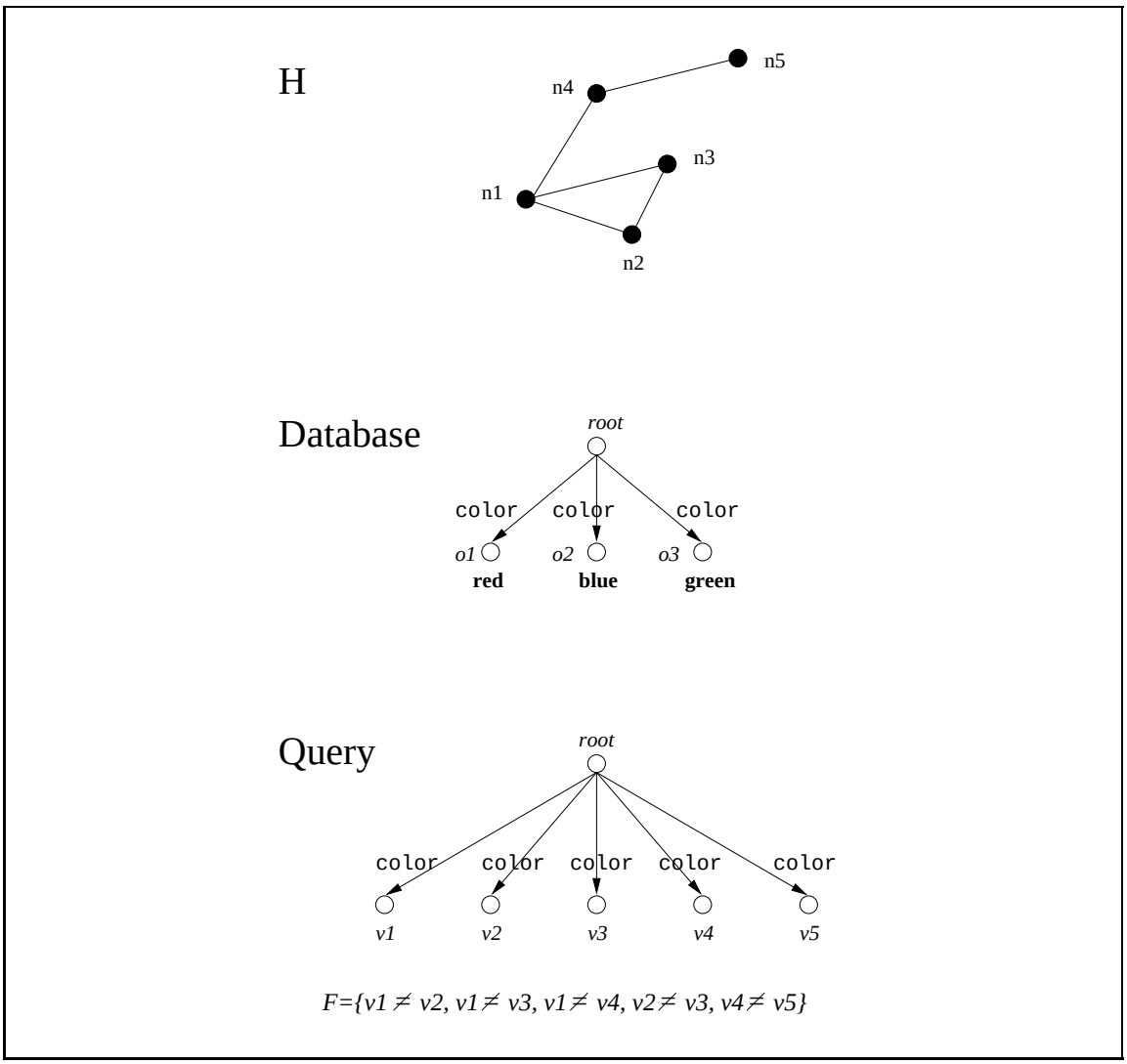

Figure 10: The construction of Lemma 5.9 for an example graph H

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 $\bar{a}'$. For each semantics $\sigma \in \{\wedge, w, \vee\}$, we say that an answer in the set $Ans^{\sigma}_{D}(Q)$ is *maximal* if it is a maximal element of $Ans^{\sigma}_{D}(Q)$ with respect to 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 with respect to σ. We say that Q is *contained* in Q' under σ if over every database D, and for every $\bar{a} \in Ans^{\sigma}_{D}(Q)$ there is an $\bar{a}' \in Ans^{\sigma}_{D}(Q')$ such that $a \sqsubseteq a'$. Obviously, two queries

are equivalent under σ if and only if they subsume each other with respect to σ. We write $Q \sqsubseteq_\sigma Q'$, where $\sigma \in \{s, \wedge, w, \vee\}$, if Q is contained in Q' under strong, AND, weak, or OR-semantics, respectively. Since for tree queries all semantics that allow for partial answers coincide, we may omit the subscript to "$\sqsubseteq$" when we refer to containment of tree queries.

6.1 Pruned Queries

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 $Q' = (Pr_{\bar{x}}(G), \bar{x})$ the *pruned version* of Q, and we say that a query is pruned if its query graph is pruned.

Note that if a variable is retained when pruning a query, then also all paths from the root of the query to such a variable are retained.

Proposition 6.1 *Suppose that $Q = (G, \bar{x})$ is a query. Let v be a node in $Pr_{\bar{x}}(G)$ and π be a path from r_G, to v in G. Then π is also a path from r_G to v in $Pr_{\bar{x}}(G)$.*

Under all semantics that allow for partial answers, a query is equivalent to its pruned version.

Proposition 6.2 *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 $\sigma \in \{\wedge, \vee, w\}$ we have*

$$Ans^\sigma_D(Q) = Ans^\sigma_D(Q').$$

Proof. Let $\bar{a} \in Ans^\sigma_D(Q)$ be an answer of Q over D. We show that $\bar{a}$ is also an answer of Q'.

There is a matching $\mu \in Mat^\sigma_D(Q)$ such that $\bar{a} = \mu(\bar{x})$. We define the assignment μ' as the restriction of μ to the nodes in $Pr_{\bar{x}}(G)$. Then $\bar{a} = \mu'(\bar{x})$. It is also easy to see that μ' is a σ-matching of Q' over D for each $\sigma \in \{\wedge, \vee, w\}$. This is the case because, as a consequence of Proposition 6.1, all edge constraints in $Pr_{\bar{x}}(G)$ are already present in G, and, since they are satisfied by μ, they are also satisfied by μ'. Since our queries have no filter constraints, μ' is a σ-solution of Q' over D. This proves the inclusion $Ans^\sigma_D(Q) \subseteq Ans^\sigma_D(Q')$.

To show the converse, let $\bar{a}$ be a σ-answer of Q' over D. Then $\bar{a} = \mu'(\bar{x})$ for some σ-solution μ' of Q'.

Let μ be the assignment to the variables of Q that is defined for a variable if and only if μ' is defined, and that agrees with μ' whenever it is defined. Then μ is a prematching because μ' is a prematching.

Suppose that μ' is a σ-matching of $Pr_{\bar{x}}(G)$. If v is a node in G with $\mu(v) \neq \perp$, then all incoming edge constraints of v in G are already present in $Pr_{\bar{x}}(G)$ because of Proposition 6.1. Thus, μ is a σ-matching of G. Let $\tilde{\mu}$ be a maximal σ-matching that extends μ. Then $\tilde{\mu}(\bar{x}) = \mu(\bar{x}) = \mu'(\bar{x}) = \bar{a}$, which implies that $\bar{a}$ is a σ-answer of Q over D. This proves the inclusion $Ans^{\sigma}_{D}(Q') \subseteq Ans^{\sigma}_{D}(Q)$. □

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

6.2 Containment under AND-semantics

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.

A homomorphism is an *isomorphism* if it is bijective and its inverse is also a homomorphism. Intuitively, the existence of an isomorphism between two queries means that the queries cannot be distinguished with respect to their structure.

We start by examining containment for the case of two queries that are evaluated under AND-semantics. In that case, we have a characterization which is similar to the well-known one for containment of conjunctive queries (cf. [CM77]).

Theorem 6.3 (Containment under AND-semantics) *Let Q_1 and Q_2 be pruned queries. Then $Q_1 \sqsubseteq_{\wedge} Q_2$, 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.*

Proof. The proof is similar to the one for the classical characterization in [CM77]. Let $Q_1 = (G_1, \bar{x})$ and $Q_2 = (G_2, \bar{x})$.

"⇒" Suppose, φ is a homomorphism from Q_2 to Q_1. If D is a database and μ_1 is a maximal AND-matching of Q_1 over D that produces the AND-answer $\bar{a}$, then the composition $\mu_2 = \mu_1 \circ \varphi$ is an AND-matching of Q_2 that also produces the answer $\bar{a}$. Any AND-matching that extends μ_2 produces the same answer. Thus, $Q_1 \sqsubseteq_{\wedge} Q_2$.

"⇐" Suppose $Q_1 \sqsubseteq_{\wedge} Q_2$. We show that there is a homomorphism from Q_2 to Q_1. To this end we consider G_1 as a database. Evaluating Q_1 over G_1, viewed as

a database, produces the AND-answer $\bar{x}$. Since Q_1 is contained in Q_2, also Q_2 returns this answer over that database. One can then show that the AND-matching of Q_2 that produces the answer $\bar{x}$ over the database G_1 is in fact a homomorphism from Q_2 to Q_1. □

Corollary 6.4 (Complexity of AND-containment) *Containment of queries under* AND*-semantics is* NP*-complete.*

Proof. To decide the existence of a graph homomorphism is NP-complete. □

Corollary 6.5 (Containment among Trees) *Existence of a homomorphism is a sufficient and a necessary condition for containment among pruned tree queries under* AND*, weak, and* OR*-semantics.*

Proof. For tree queries, AND, weak, and OR-semantics coincide. □

6.3 Containment under OR and Weak 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 6.6 *Let $Q = (G, \bar{x})$ be a query and $\mathcal{T}_Q$ its tree expansion. Then we have for any database D that*

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

Proof. Obviously, every answer returned by a query in the tree expansion of Q is also returned by Q. It remains to show the converse.

Suppose $\bar{a} \in Ans^{\vee}_D(Q)$. Then $\bar{a} = \mu(\bar{x})$ for some OR-matching μ of Q. For each variable $x_j \in \bar{x}$ to which μ assigns a value distinct from $\perp$, there is a path from the root of Q to x_i such that every variable on the path is bound to a value distinct from $\perp$, and all constraints on the path are satisfied by μ. Let G' be the subgraph of G that is the union of these paths, and let T' be a spanning tree of G'. We can extend

T' to a subgraph T'' of G such that T'' is a tree and contains the variables in $\bar{x}$. Let T be the pruned version of T''. Then the restriction $\mu' := \mu_{|T}$ of μ to T is a matching of T, and $\mu'(\bar{x}) = \bar{a}$. Hence, for $Q' := (T, \bar{x})$ we have $\bar{a} \in Ans^{\vee}_D(Q')$. □

Theorem 6.7 (Containment under OR-semantics) *For two queries $Q_1 = (G_1, \bar{x})$ and $Q_2 = (G_2, \bar{x})$ the following are equivalent:*

- *$Q_1 \sqsubseteq_{\vee} Q_2$, i.e., Q_1 is contained in Q_2 under* OR*-semantics;*
- *for every query $Q_1' \in \mathcal{T}_{Q_1}$ there is a query $Q_2' \in \mathcal{T}_{Q_2}$ such that $Q_1' \sqsubseteq Q_2'$.*

Proof. "⇒" We show that if $Q_1 \sqsubseteq_{\vee} Q_2$, then for every query $Q_1' = (T_1, \bar{x}) \in \mathcal{T}_{Q_1}$ in the tree expansion of Q_1, there is a query $Q_2' \in \mathcal{T}_{Q_2}$ such that $Q_1' \subseteq Q_2'$.

We give a proof sketch. As in the classical proof in [CM77], we view T_1 as a database D. If we evaluate Q_1' over D, the answer $\bar{x}$ is returned. Since $Q_1 \sqsubseteq_{\vee} Q_2$, and because of Proposition 6.6, there is also a query $Q_2' = (T_2, \bar{x})$ in $\mathcal{T}_{Q_2}$ that returns the answer $\bar{x}$. This answer is produced by a matching μ of Q_2' over D, that is, $\bar{x} = \mu(\bar{x})$. Viewing D again as the query graph T_1, we see that μ is a homomorphism from Q_2' to Q_1'. Now, by Corollary 6.5, the existence of a homomorphism between tree queries implies containment, which yields that $Q_1' \sqsubseteq Q_2'$.

"⇐" Suppose that for every query $Q_1' \in \mathcal{T}_{Q_1}$ there is a $Q_2' \in \mathcal{T}_{Q_2}$ such that $Q_1' \sqsubseteq Q_2'$. Then we have for every database D that

$$\bigcup_{Q_1' \in \mathcal{T}_{Q_1}} Ans^{\vee}_D(Q_1') \subseteq \bigcup_{Q_2' \in \mathcal{T}_{Q_2}} Ans^{\vee}_D(Q_2').$$

By Proposition 6.6 this implies $Ans^{\vee}_D(Q_1) \subseteq Ans^{\vee}_D(Q_2)$. Hence, $Q_1 \sqsubseteq_{\vee} Q_2$. □

Theorem 6.8 (Complexity of OR-containment)

1. *Containment of queries under* OR*-semantics is in Π^{P}_2.*
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^{P}_2, is clear from the characterization in Theorem 6.7. 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 a variant of the Hamiltonian path problem. We are given a directed graph $G = (N, \gamma)$, where N is the set of nodes and $\gamma: N \to 2^N$ associates to every node its set of successors. In addition, we are given two nodes n_s, $n_e \in N$. The question is, whether there exists a path $n_s = n_1, \ldots, n_k = n_e$ such that each node of N occurs exactly once among the $n_1, \ldots, n_k$, and each n_i is a successor of n_{i-1} for $i \in 2..k$. We reduce this problem to

a containment problem. Without loss of generality, we can assume that all nodes in the graph G are reachable from the start node n_s.

We obtain from G a rooted ldg G_2 that has only edges with the label `succ` by considering the nodes of G as variables, drawing an edge between two variables if they are linked by γ, and making n_s the root of G_2. The query $Q_2 = (G_2, n_e)$ has G_2 as its query graph and the end node n_e as its distinguished variable.

We define the graph G_1 as having the same nodes as G_2. There is a `succ`-edge from n_{i-1} to n_i for $i \in 2..k$, and there are no other edges. Again, the root of G_1 is n_s. We turn G_1 into the query $Q_1 = (G_1, n_e)$, which has n_e as its distinguished variable. A homomorphism from Q_2 to Q_1 maps n_s to n_s and n_e to n_e. Since G_1 is a thread from n_s to n_e, and all nodes in G_2 are reachable from the root, every homomorphism establishes a linear order on the nodes of G_2, with n_s as the first and n_e as the last element. This order defines a Hamiltonian path from n_s to n_e. Conversely, such a path gives rise to a linear order and thus to a homomorphism.

This shows that containment in the second case is NP-hard. It is also in NP, because the tree expansion of the containee has only one element. □

For weak semantics we have a characterization of containment resembling the one for OR-semantics that is given in Theorem 6.7. 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 6.8 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 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 access to the World-Wide Web. As described abstractly in this paper, queries are based on query graphs, which have to be matched against a 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 expressions.

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, *Proc. 6th International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18, Delphi (Greece), January 1997. Springer-Verlag.

[ABS00] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann Publishers, 2000.

[AM98] G.O. Arocena and A.O. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proc. 14th International Conference on Data Engi-*

neering, 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, *Proc. 6th 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, *Proc. 6th 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 J. Paredaens, editor, *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.

[CGd99] D. Calvanese and M. Lenzerini G. de Giacomo. Queries and constraints on semi-structured data. In 1626, editor, *Advanced Information Systems Engineering, Proc. 11th International Conference*, Lecture Notes in Computer Science, pages 434–438, Heidelberg (Germany), July 1999. Springer-Verlag.

[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 J. Paredaens, editor, *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.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman and Company, San Francisco, 1979.

[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. 3nd 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.

[MW95] A. Mendelzon and P. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6), 1995.

[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.

[Ull89] J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. II: The New Technologies.* Computer Science Press, New York (New York, USA), 1989.