Incomplete Answers for Queries over Semistructured Data

Yaron Kanza* and **Werner Nutt†** and **Yehoshua Sagiv***

Abstract

Semistructured data are inherently incomplete and their structure does not comply with a strict schema. A query language has to take this into account and allow answers to be incomplete. This paper presents a model of queries over semistructured data that allows for incomplete answers, i.e., answers in which not all the variables of the query are bound. We define a simple data model that is similar to OEM [PGMW95]. For this data model we present the concept of a query language based on constraints. The paper describes the different constraints that are used in the query and the different phases needed for evaluating them. We propose the concept of *maximal answers* that are obtained by matching a maximal portion of the database graph with the query constraints. Maximal matchings and constraints that can be evaluated over partial data give us the ability to retrieve partial answers when complete answers do not exist.

1 Introduction

Semistructured data lack a homogeneous structure and are incomplete and irregular. When evaluating queries over semistructured data one has to deal with partial information. When querying the world wide web or another semistructured information source we may want to get answers for a query even when these answers are not complete.

We define a simple data model which 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 one is an attribute filler for the first one. Our queries, too, are defined by graphs, which are to be matched against the database.

In an abstract view, our queries consist of a set of variables and constraints on the variables. We divide the constraints into *search constraints* and *filter constraints.* The queries we suggest have three components, which resemble those of an SQL-query. The first component, the set of search constraints, is analogous to the FROM-clause. The search constraints form a labeled directed graph whose nodes are variables: they are a pattern that has to be matched by a part of the database. We are only interested in *maximal matchings* because they contain maximal information. We define different semantics for the search phase and get different matchings under the different semantics. The second component is the set of filtering constraints, which is analogous to the WHERE-clause. Those matchings which pass the filtering are called *solutions.* The third component is a tuple of *output variables,* which, like the SELECT-clause, specifies *answers* as restrictions of the solutions to some 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 a matching, 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.

For the different semantics, we have developed evaluation algorithms, and we have characterized the containment of queries without filter constraints. We also have complexity results for both types of problems. Due to space constraints, we are unable to present them here.

2 Data Model

Our data model is a simplified version of the Object Exchange Model (OEM) [PGMW95, AQM+97]. Both

*The Hebrew University, Jerusalem 91904, Israel

†German Research Center for Artificial Intelligence (DFKI GmbH), 66123 Saarbrücken, Germany

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. An atomic value has a type. A type includes a set of atoms and a set of decidable relations on the elements of the type, like numbers and comparisons. In every 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 define a *labeled directed graph* or *ldg* over a set of nodes N as a pair $G = (N, \cdot^G)$ where $\cdot^G$ associates to each label l a binary relation $l^G \subseteq N \times N$ between the nodes. We will often view a binary relation l^G as a function $l^G: N \to 2^N$. Note that in a labeled directed graph there can be two nodes u and v such that in the graph there are two different edges from u to v labeled with different labels. If $n' \in l^G(n)$, we say that (n, n') is an edge of G. 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 by a path in G. We denote a rooted ldg G as a triple $G = (N, r_G, \cdot^G)$. A node n in G is a *terminal node* if $l^G(n)$ is empty for each label l, and an *inner node* otherwise. We say that G is *finitely branching* if the number of outgoing edges from every node n is finite.

A *database* consists of a rooted finitely branching ldg $(O, r_D, \cdot^D)$ over a set of objects O and a function α that maps each terminal node to an atom. We denote a database as a 4-tuple $D = (O, r_D, \cdot^D, \alpha)$. The graph in Figure 1 depicts a database containing information about university departments, courses and staff at the Hebrew University. The nodes are the database objects, and edges are annotated with their labels.

3 Overview of the Query Language

In this section we define queries and their semantics. 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 defined later on; and (3) $\bar{x}$ is a tuple of variables occurring in V called the *output variables.*

We also view the search graph G as a set of *search 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. We say that ulv is an *incoming constraint* of the variable v. 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.

3.1 The Search Phase

We start by describing assignments of database objects to the query variables. We denote by $\perp$ a new symbol, called *null.* Let D be a database over O and V be a set of node variables. A *D-assignment over V* is a mapping $\mu: V \to O \cup \{\perp\}$. If $\mu(v) \neq \perp$, we say that the assignment μ is *defined* for the variable v, or that v is *bound.* We say that μ *satisfies* an edge constraint ulv if $\mu(v) \in l^D(\mu(u))$, i.e., the relation l^D of 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.

We define various semantics of the search constraints by stating which search constraints an assignment has to satisfy in order to be a matching. First, all defined variables of a matching must be connected to the root. 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) $\mu(r_G) = r_D$, i.e., μ maps the root of G to the root of 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. The assignment μ is a *strong matching* for G if μ is a prematching that 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. We now define three semantics that allow matchings to be partial assignments. (1) *weak matching*: 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 a constraint only if it is defined for all its variables. (2) AND-*matching*: The prematching μ is an AND-matching if it satisfies *all* incoming constraints of v whenever $\mu(v) \neq \perp$. (3) OR-*matching*: The prematching μ is an OR-matching of Q if it satisfies *some* incoming constraint of v whenever $\mu(v) \neq \perp$. Note that a prematching is also an OR-matching. The sets of all strong, weak, AND, and OR-matchings of Q over D are denoted as

$$Mat^s_D(Q),\ Mat^w_D(Q),\ Mat^\wedge_D(Q),\ Mat^\vee_D(Q),$$

respectively. Obviously, we have

$$Mat^s_D(Q) \subseteq Mat^\wedge_D(Q) \subseteq Mat^w_D(Q) \subseteq Mat^\vee_D(Q).$$

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

Let A be a set of assignments that range over the same set of variables V and let $\mu, \mu' \in A$. We say that μ *subsumes* μ', and write $\mu' \sqsubseteq \mu$, if, whenever $\mu'(v)$ is defined, then $\mu(v)$ is defined and $\mu(v) = \mu'(v)$. An assignment μ

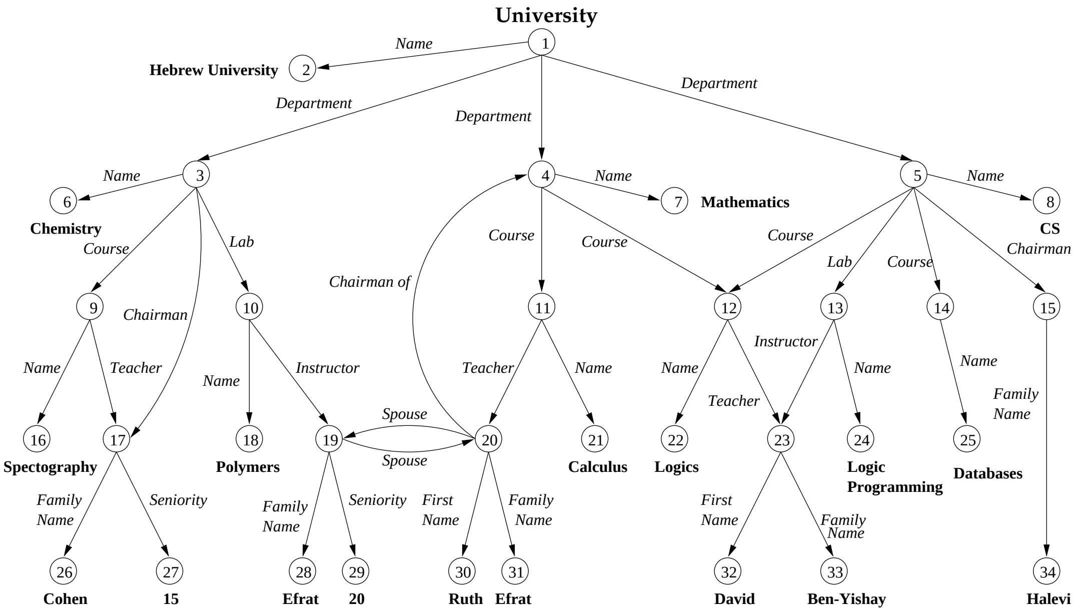

Figure 1: A university database

is a *maximal* element of A if for any element $\mu' \in A$ we have that $\mu \sqsubseteq \mu'$ implies $\mu = \mu'$.

Given a query Q and a database D, we are interested in matchings that have maximal information content. For each $* \in \{s, \wedge, w, \vee\}$ we thus define $MMat^*_D(Q)$ as the set of maximal elements of $Mat^*_D(Q)$. Obviously, $MMat^s_D(Q)$ is the same as $Mat^s_D(Q)$, but for the other types of matchings not all matchings are maximal.

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 $\mathcal{D}$. The elements of $\mathcal{D}$ are the *atomic values* of T. The function α in the database maps the terminal nodes to atoms and each atom is of some type. For simplicity we assume for now that all atomic values are of one 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 $u \doteq_v v$ and $u \not\doteq_v v$ respectively. *Object comparisons* are constraints of the form $u \doteq_o v$ and $u \not\doteq_o v$. The object comparisons are used to compare the identity of database objects. An existence constraint has the form $!v$ and it requires the variable v to be bound.

Filter constraints are applied to maximal matchings, which may be undefined for certain variables. We take this into account by distinguishing between two kinds of constraint satisfaction, 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 an atomic database node; and (2) the tuple $(\alpha(\mu(s_1)), \ldots, \alpha(\mu(s_n)))$ is in the relation r. The assignment μ *strongly satisfies* the object equality $u \doteq_o v$ if $\mu(u)$ and $\mu(v)$ are defined, and $\mu(u) = \mu(v)$. For object inequalities the definition is analogous. Thus, in order for an assignment to strongly satisfy a constraint, it must be defined for all the variables occurring in the constraint, and the values must be in the constraint relation. An assignment *weakly satisfies* a constraint if it is undefined for one of the variables of the constraint, or otherwise, if it strongly satisfies it. If μ strongly (weakly) satisfies a set of constraints C, we write $\mu \models_s C$ ($\mu \models_w C$). We say that μ satisfies the existence constraint $!v$ if $\mu(v) \neq \perp$. Here, we do not distinguish between strong and weak satisfaction. We can turn weak satisfaction into strong satisfaction by adding existence constraints. If c_f is a filter constraint with variables $v_1, \ldots, v_n$ and μ is an assignment, then $\mu \models_s \{c_f\}$ if and only if $\mu \models_w \{c_f, !v_1, \ldots, !v_n\}$.

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

we define the set of $*$-*solutions* of Q over D as

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

3.3 Answers

Up to now we defined the role of the search constraints and filter constraints in the evaluation of a query $Q = (G, F, \bar{x})$ over a database D. The result is the set of solutions $Sol^*_D(Q)$, which depends on the semantics $*$. Sometimes however, we are only interested in the bindings to some of the variables of the query, those in the tuple of output variables $\bar{x}$. We define the set of $*$-*answers* of Q over D as consisting of the tuples $\mu(\bar{x})$, where μ is a $*$-solution of Q, and denote it as $Ans^*_D(Q)$.

4 Examples: Matchings, Solutions and Answers

Figure 2 shows as an example a query graph G_1.

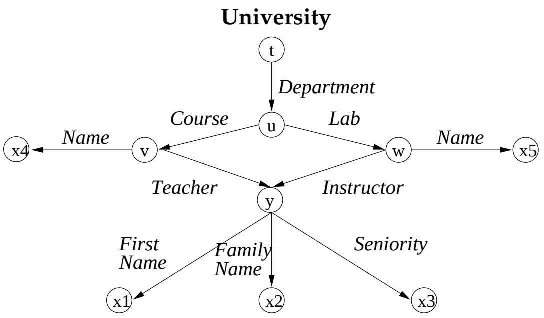

Figure 2: Query graph G_1 asking for course teachers and lab instructors

We evaluate the query graph over the university database. Figure 3 contains a table with the maximal matchings under AND, weak, and OR-semantics.

Under the different semantics, the query in Figure 2 has different meanings. The database objects which are assigned to y under AND-semantics refer to people that are both a course teacher and a lab instructor. Under OR-semantics the objects refer to people that are either a course teacher or a lab instructor. Since we are interested in information about the people, we require the variable y to be bound by adding the existence constraint $!y$. We declare $x1$, $x2$, $x3$ $x4$, $x5$ to be the output variables to get as an answer the information about the person, the course she teaches, and the lab in which she is an instructor.

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 $\bar{x} = (x1, x2, x3, x4, x5)$. Figure 4 shows the answers to Q_1 under AND and weak semantics (Instead of the terminal objects in the answer, the attached values appear in the

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	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	25	$\perp$
Weak	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	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	25	$\perp$
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

Figure 3: Maximal matchings for the search graph G_1 in Figure 2 over the university database

table). We see that under AND-semantics, we receive the one person who both teaches a course and is a lab instructor, with the information about the course and the lab. Under weak semantics we get the information about the persons who teach a course or instruct a lab. We also obtain information about the course the person teaches if there is one and information about the lab where the person is an instructor.

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 $x3$ 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.

5 Conclusion

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 either that data are complete. 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. The query languages so far proposed for semistructured data, like Lorel [AQM+97], UnQL [BDHS96], or Strudel [FFK+98], do not fully take into account the incompleteness of the data. In this paper, we develop foundations of a query language based

Semantics		First Name	Family Name	Seniority	Course Name	Lab Name
And	1	David	Ben-Yishay		Logics	Logic Programming
Weak	1		Cohen	15	Spectography	
	2		Efrat	20		Polymers
	3	Ruth	Efrat		Calculus	
	4	David	Ben-Yishay		Logics	Logic Programming

Figure 4: Answers of Q_1 over the university database under AND and weak semantics

on constraints which returns incomplete answers. We suggest that querying incomplete data proceed in two phases. The first is retrieving the maximal information which match our search pattern and the second is filtering the maximal matchings using weak and strong constraints.

Usually, query languages for semistructured data support 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 [AV97, FLS98]. We have limited our investigation till now to queries without regular path expressions since regular path expression raise many new questions involving higher complexity and new difficulties. Hence we leave that for further research.

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

References

[Abi97] S. Abiteboul. Querying semi-structured data. *International Conference on Database Theory*, volume 1186 of *LNCS*, pages 1–18, Delphi (Greece), January 1997. Springer-Verlag.

[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.

[AV97] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. 16th PODS*, pages 122–133, Tucson (Arizona, USA), May 1997. ACM Press.

[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*, pages 505–516, Montreal (Canada), June 1996.

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

[Bun97] P. Buneman. Semistructured data. In *Proc. 16th PODS*, pages 117–121, Tucson (Arizona, USA), May 1997. ACM Press.

[CDL98] D. Calvanese, G. De Giacomo, and M. Lenzerini. What can knowledge representation do for semistructured data? In *Proc. 15th National Conference on Artificial Intelligence*, pages 205–210, Madison (Wisconsin, USA), July 1998. Morgan Kaufmann Publishers.

[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*, pages 414–425, Seattle (Washington, USA), June 1998. ACM Press.

[FLS98] D. Florescu, A.Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. 17th PODS*, 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 semistructured data. In *6th International Workshop on Database Programming Languages*, Estes Park (Colorado, USA), August 1997. Springer-Verlag.

[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.

[NAM98] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. 1998 ACM SIGMOD*, 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. *Proc. 11th International Conference on Data Engineering*, pages 251–260, Taipei, March 1995. IEEE Computer Society.

[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*, page 549, Montreal (Canada), June 1996.