Adding Structure to Unstructured Data

Peter Buneman[1] and Susan Davidson[1] and Mary Fernandez[2] and Dan Suciu[2]

[1] University of Pennsylvania, USA {peter,susan}@cis.upenn.edu
[2] AT&T Labs — Research, USA {mff,suciu}@research.att.com

Abstract. We develop a new schema for unstructured data. Traditional schemas resemble the type systems of programming languages. For unstructured data, however, the underlying type may be much less constrained and hence an alternative way of expressing constraints on the data is needed. Here, we propose that both data and schema be represented as edge-labeled graphs. We develop notions of conformance between a graph database and a graph schema and show that there is a natural and efficiently computable ordering on graph schemas. We then examine certain subclasses of schemas and show that schemas are closed under query applications. Finally, we discuss how they may be used in query decomposition and optimization.

1 Introduction

The ability to represent and query data with little or no apparent structure arises in several areas: biological databases, database integration, and query systems for the World-Wide Web[PGMW95, TMD92, BDHS96a, MMM96, QRS+95, KS95, CM90]. The general approach is to represent data as a labeled graph. Data values and schema information, such as field and relation names, are kept in one data structure, blurring the distinction between schema and instance.

Although these models merge schema and data, distinguishing between them is important, because schemas are useful for query decomposition and optimization and for describing a database's structure to its users. The biological database system ACeDB [TMD92] allows flexible representation of data, but also has a schema-definition language that limits the type and number of edges stored in a database. The OEM [PGMW95] model supports database integration by providing a structure in which most traditional forms of data (relational, object-oriented, etc.) can be modeled. Even the World-Wide Web, which appears to be completely unstructured, contains structured subgraphs. Fig. 1 depicts a fragment of the web site `http://www.ucsd.edu`, in which pages connecting schools, departments, and people are structured. Queries applied to this graph's link structure can benefit from structural information, for example, by knowing there exists at most one department on any path from the root to a leaf and that every paper is reachable from a department.

We describe a new notion of schema appropriate for an edge-labeled graph model of data. We use this model to formulate, optimize, and decompose queries for unstructured data [BDS95, BDHS96a, Suc96]. Informally, a database is an edge-labeled graph, and a schema is a graph whose edges are labeled with formulas. A database DB conforms to a schema S if there is a correspondence between the edges in DB and S, such that whenever there is an edge labeled a in DB, there is a corresponding edge labeled with predicate p in S such that

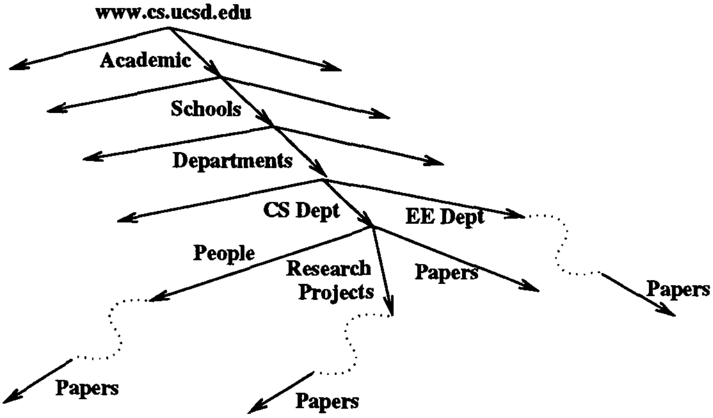


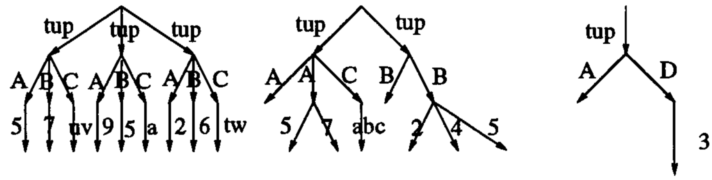
Fig. 1. A fragment of `http://www.ucsd.edu`.

$p(a)$ holds. This notion of conformance is a generalization of *similarity* [HHK95]. We investigate the properties of such schemas, and show that there is a natural subsumption ordering on schemas – a generalization of similarity. We then investigate a "deterministic" subclass of schemas and argue that it is appropriate to have deterministic schemas although data may be "nondeterministic". Finally, we examine queries on a database with a known schema and consider when we can compute a schema for the result of the query. We also discuss how schemas can improve the optimization and decomposition of queries in UnQL [BDHS96a].

2 Basic Definitions

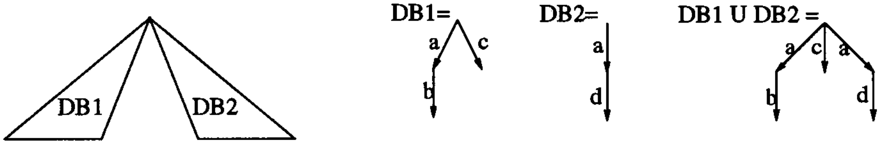
Let $\mathcal{U}$ be the universe of all constants ($\mathcal{U} = Int \cup String \cup Bool \cup \ldots$). We adopt the data model of [BDHS96a], where a *graph database* is a rooted graph with edge labels in $\mathcal{U}$. Formally, $DB = (V, E, v_0)$, where V is a set of nodes, $E \subset V \times \mathcal{U} \times V$, and $v_0 \in V$ is a distinguished *root*. Fig. 1 is an example of a graph database. This model is powerful enough to encode relational databases, as illustrated in Fig. 2(a), which encodes a relation $R(A : Int, B : Int, C : String)$, but flexible enough to represent unstructured data, like Fig. 2(b) and (c). Sets, records, and variant nodes are equivalent in this model. Graphs may have arbitrary cycles and sharing. Two graphs are considered equal if they are *bisimilar* [BDHS96b]. Briefly, DB and DB' are bisimilar if there exists a binary relation $\approx$ from the nodes of DB to those of DB' such that (1) $v_0 \approx v'_0$ where v_0, v'_0 are the two roots, and (2) whenever $u \approx u'$, then for every $u \xrightarrow{a} v$ in DB, there exists $u' \xrightarrow{a} v'$ in DB' such that $v \approx v'$, and for every $u' \xrightarrow{a} v'$ in DB', there exists $u \xrightarrow{a} v$ in DB such that $v \approx v'$.

In earlier work [BDHS96a], we introduced a notation for specifying graphs, e.g., the tree database in Fig. 2(c) is written as $\{tup \Rightarrow \{A, \{D \Rightarrow \{3\}\}\}\}$. Also, we defined a union operation on two graph databases in which their two roots are collapsed (Fig. 3(a)). For example, in Fig. 3(b) $DB_1 = \{a \Rightarrow \{b\}, c\}, DB_2 = \{a \Rightarrow \{d\}\}$, and $DB_1 \cup DB_2 = \{a \Rightarrow \{b\}, c, a \Rightarrow \{d\}\}$.



(a) (b) (c)

Fig. 2. Three examples of graph databases.



(a) (b)

Fig. 3. Union of graph databases.

To define graph schema, consider a set of base predicates over $\mathcal{U}$, $P_1, P_2, \ldots$, such that the first order theory T generated by $\mathcal{U}$ (i.e. the first order sentences true in $\mathcal{U}$) is decidable. A *unary formula* is a formula with at most one free variable.

Definition 1. **A graph schema** is a rooted, labeled graph, in which the edges are labeled with unary formulas.

Although our results apply to every decidable theory, we use theories generated by unary predicates, with equality and with names for all constants in our universe. Typical predicates include $Int(x), String(x), Nat(x)$, and $Bool(x)$, which denote $x \in Int, x \in String, x \in Nat$, and $x \in Bool$, and user-defined unary predicates, $P(x)$. The theory has an equality operator, so we have predicates such as $x = 5$ and $x =$ "*abc*". Such a theory is decidable, because it admits quantifier elimination: e.g. $\exists y.(Int(x) \wedge Int(y) \wedge x \neq y)$ is equivalent to $Int(x)$.

Fig. 4 (a) depicts a graph schema S. By convention, we drop the free variable from unary formulas which are boolean combinations of unary predicates, thus writing A and $Int \vee String$ instead of $x = A$ and $Int(x) \vee String(x)$. Intuitively, a graph schema captures some knowledge about the structure of a graph database. In particular, the graph schema S says that a graph database that conforms to S has only *tup*-edges emerging from the root, possibly followed by A, B, or C edges, and these possibly followed by integers or strings respectively. The graph database encoding a relational database in Fig. 2(a) conforms to this graph schema, but the graph in Fig. 2(c) does not. The database in Fig. 2(b) also conforms to this schema, although it does not encode any relational database.

In schemas (c), (d), (e), (f) in Fig. 4, *isDept*(x) and *isPaper*(x) are user-defined predicates testing whether x is a string denoting a department (e.g., "Computer Science Department" or "Electrical Engineering Department") or a paper. Schema (d) says that there is at most one department on every path

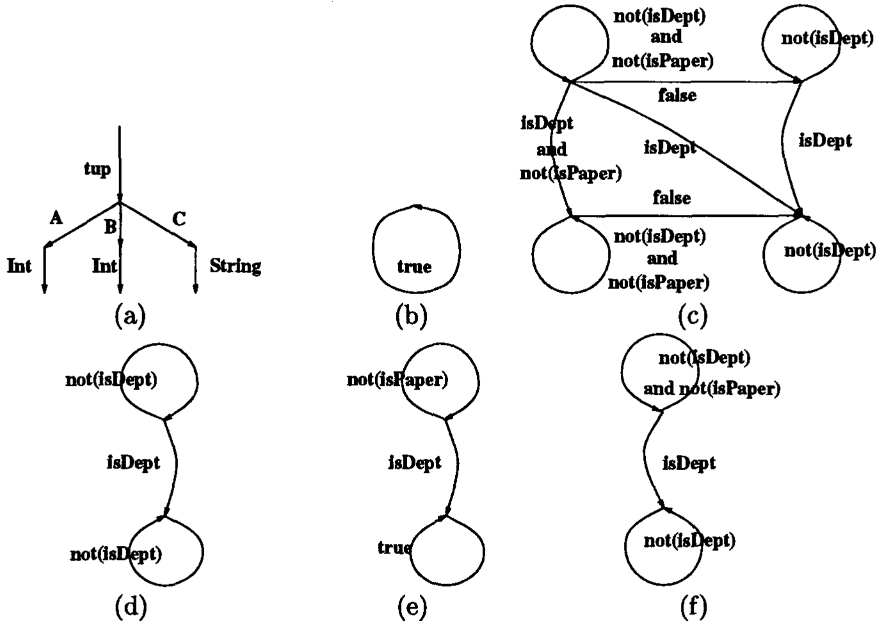


Fig. 4. Six examples of graph schema.

starting at the root, while that in (e) says that no paper edge may occur before a department edge. The database in Fig. 1 conforms to both these schemas.

Definition 2. A database DB *conforms* to a graph schema S, $DB \preceq S$, if there exists a *simulation* from DB to S, i.e. a binary relation $\preceq$ from the nodes of DB to those of S satisfying: (1) the root nodes of DB and S are in the relation $\preceq$, (2) whenever $u \preceq u'$ and $u \xrightarrow{a} v$ is an edge labeled a in DB, then there exists some edge $u' \xrightarrow{p} v'$ in S such that $p(a)$ is true and $v \preceq v'$.

A graph schema cannot enforce the presence of some label. This is consistent with the notion of schema in ACeDB [TMD92]. In particular, the empty database (one node, no edges) conforms to any graph schema S, i.e., $\emptyset \preceq S$. A graph schema cannot model variants, nor can it prevent a node from having several outgoing edges with the same label, as occurs in Fig. 2(b). Finally, any database DB can be viewed as a schema, by replacing every label a with the unary formula $x = a$, which gives us a notion of simulation between databases, $DB \preceq DB'$.

In keeping with our view that two graphs are considered equal if they are bisimilar, we can show that if $DB \preceq S$ and DB and DB' are bisimilar, then $DB' \preceq S$. However, note that $DB \preceq DB'$ and $DB' \preceq DB$ does not necessarily imply that DB, DB' are bisimilar.

Graph schemas can be viewed as infinite databases. For example, we view an edge $u \xrightarrow{Nat} v$ in S, as representing infinitely many edges, $u \xrightarrow{0} v, u \xrightarrow{1} v, u \xrightarrow{2} v, \ldots$. We call the *expansion* of S, denoted S^∞, the (possibly infinite) database obtained from replicating each edge in S once for every constant in the universe

$\mathcal{U}$ satisfying the unary formula on that edge. See Fig. 5 for an example. If any of the schema edges is labeled with the formula *false*, that edge disappears in S^∞.

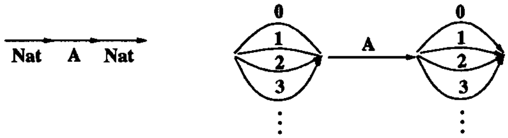


Fig. 5. A graph schema S and its infinite expansion S^∞.

One can easily check that for any database DB and graph schema S, $DB \preceq S$ iff $DB \preceq S^\infty$. The latter relation is a simulation between two databases, one of which may be infinite.

3 Complexities

Paige and Tarjan [PT87] give an $O(m \log n)$ algorithm for the ***relational coarsest partition*** problem, which computes a bisimulation relation on a graph, where n is the number of nodes and m the number of edges. The algorithm tests whether two rooted graphs G_1 and G_2 are bisimilar: take their disjoint union G, compute a bisimulation $\approx$ on G, then test whether the two roots of G_1 and G_2 are in $\approx$. Although bisimulation and simulation are related, they require different algorithms. Henzinger, Henzinger, and Kopke [HHK95] have recently found an $O(mn)$ time algorithm to compute the simulation between two graphs with labeled nodes.

Neither algorithm applies directly to our framework, because they associate labels with nodes, not edges. We can reduce the problem of finding a (bi)simulation of two edge-labeled graphs with a total of n nodes and m edges to that of finding a (bi)simulation between two node-labeled graphs with a total of $m + n$ nodes and $2m$ edges. We split each labeled edge $x \xrightarrow{a} y$ into two unlabeled edges $x \to z \to y$, in which z is a new node labeled a, and we label all other nodes with a new, unique label. Finally, we compute a (bi)simulation for the new graphs, in time $O(2m \log(m + n)) = O(m \log m)$ for bisimulation, or $O((m + n)2m) = O(m^2)$ for simulation. We may assume $m \geq n$, because the graphs G_1, G_2 are connected, but unlike in [HHK95], we no longer necessarily have $m \leq n^2$. This still does not allow us to test $DB \preceq S$, because when we expand S into a database we get an infinite graph. We can, however, adapt the algorithm in [HHK95] to get:

Proposition 3. *Suppose one can test validity of sentences of the theory T in time t. Then there exists an algorithm for checking whether $DB \preceq S$ that runs in time $O(m^2 t)$. Here m is the total number of edges in DB and S, which are each assumed to be connected.*

4 Expressiveness of graph schemas

Graph schemas differ from relational or object-oriented schemas. A relational database has only one schema. A graph database, however, may conform to several graph schemas such as those in Fig. 4 (d) and (e). Moreover, there exists a schema $S_\top$ (Fig. 4 (b)) to which all graph databases conform. Since graph schemas are meant to capture partial information about the structure of data with the purpose of optimizing queries, we could store multiple graph schemas for the same data and offer multiple "hints" to a query optimizer.

The relationship between graph database and graph schemas raises several questions. First, given two graph schemas S and S', how do we know if S says more about some database than S'? How do we know that graph schemas S and S' are "equivalent", i.e. $DB \preceq S$ iff $DB \preceq S'$, for any DB? For example, the graph schema in Fig. 4(f) captures more information about a database than either schema in (d) or (e). Formally, if $[S] \stackrel{\text{def}}{=} \{DB \mid DB \preceq S\}$, then we want to check whether $[S] \subseteq [S']$ and $[S] = [S']$. We show that both $[S] \subseteq [S']$ and $[S] = [S']$ can be checked in polynomial time.

Second, given two graph schemas S and S', which express different constraints on a database, can we describe with a single graph schema S'' their combined constraints ? We want some graph schema S'' such that $DB \preceq S \wedge DB \preceq S'$ iff $DB \preceq S''$. We show that S'' always exists. For example, when S, S' are those in Fig. 4 (d), (e), then S'' is the schema in (f).

Last, when $DB \not\preceq S$, what "fragment" DB_0 of DB does conform to S? This question is important if we wish to use graph schema as *data guides* [Abi97]. Assume we optimize queries based on the assumption that the Web site in Fig. 1 follows schema S in Fig. 4 (d) as a guide. Since the schema does not enforce conformance it is unclear what the optimized query means. We show here that for any database DB and schema S there exists a canonical "fragment" DB_0 of DB that conforms to S. Moreover, whenever $DB \preceq S$, then DB_0 is DB. We can now state what we expect from an optimizer. Given a query Q and schema S, we expect a correct optimizer to produce an optimized query Q_{opt} such that for any database DB, $Q_{\text{opt}}(DB) = Q(DB_0)$. This implies that $Q_{\text{opt}}(DB) = Q(DB)$ whenever $DB \preceq S$.

4.1 Subsumption of graph schemas

We define schema subsumption and equivalence as follows.

Definition 4. Given two graph schemas S, S' we say that S *subsumes* S', in notation $S \preceq S'$, if there exists a binary relation $\preceq$ between the nodes of S and S' such that: (1) $v_0 \preceq v_0'$, where v_0, v_0' are the roots of S, S', (2) whenever $u \preceq u'$, for every labeled edge $u \xrightarrow{p} v$ in S and every $a \in \mathcal{U}$ s.t. $\mathcal{U} \models p(a)$, there exists an edge $u' \xrightarrow{p'} v'$ in S' s.t. $\mathcal{U} \models p'(a)$ and $v \preceq v'$. S and S' are *equivalent* if $S \preceq S'$ and $S' \preceq S$.

The subsumption relation, $S \preceq S'$, naturally extends the simulation relation between databases. Recall that a graph schema S represents its possibly infinite expansion, S^∞, i.e., an edge $x \xrightarrow{p} y$ represents infinitely many edges, one for each a for which $\mathcal{U} \models p(a)$. Each such edge may be simulated in S' by some unary formula. First, we choose $a \in \mathcal{U}$, then decide which edge $x' \xrightarrow{p'} y'$ in S' will "mimic" the edge $x \xrightarrow{p} y$ in S. For example, let $S = \{Int \vee String \Rightarrow \{5\}\}, S' = \{Int \Rightarrow \{Int\}, String \Rightarrow \{Int\}\}$, then $S \preceq S'$, because $\forall a \in \mathcal{U}$ for which $Int(a) \vee String(a)$ there is a corresponding edge in S'.

Proposition 5. *$S \preceq S'$ iff $S^\infty \preceq S'^\infty$. The latter is the simulation relation between (possibly infinite) databases.*

In particular, a database DB conforms to a graph schema S, $DB \preceq S$, iff DB when viewed as a graph schema subsumes S, for which we use the same notation $DB \preceq S$.

We now determine whether $S \preceq S'$. From [HHK95], this problem is decidable. Moreover, our algorithm in Fig. 6 checks whether $S \preceq S'$ in polynomial time.

Let $R \longleftarrow \{(u,u') \mid u \in nodes(S), u' \in nodes(S')\}$
while any change **do**
  find $(u,u') \in R$ and edge $u \xrightarrow{p} v$ in S
    such that $\mathcal{U} \models \exists a.p(a) \wedge (\bigwedge_{i=1,k} \neg p'_i(a))$
      where $u' \xrightarrow{p'_i} v'_i$, $i = 1,k$ are all edges from u' in S'
  $R \longleftarrow R - \{(u,u')\}$
return $((v_0, v'_0) \in R)$

Fig. 6. An algorithm checking whether $S \preceq S'$.

Proposition 6. *The algorithm in Fig. 6 checks in time $m^{O(1)}t$ whether $S \preceq S'$, where t is the time needed to check validity of a sentence in the theory T.*

We want to use this algorithm to check whether $[S] \subseteq [S']$. Corollary 8, which says that $[S] \subseteq [S']$ is equivalent to $S \preceq S'$, allows us to do that. To prove it, we observe that the subsumption relation $\preceq$ on graph schemas is preorder (from Proposition 5), and this allows us to define the least upper bound of a set of graph schemas, as in any preordered set. We review here the definition for completeness. Let $\mathcal{D}$ be a set of graph schemas. S is a *least upper bound* for $\mathcal{D}$ if (1) $\forall S_0 \in \mathcal{D}$, $S_0 \preceq S$, and (2) when another graph schema S' has this property, it follows that $S \preceq S'$. We use $\bigsqcup \mathcal{D}$ for the set of least upper bounds of $\mathcal{D}$. Since $\preceq$ is a preorder rather than an order relation, $\bigsqcup \mathcal{D}$ may have more than one element, but all are equivalent, i.e. $S, S' \in \bigsqcup \mathcal{D} \Longrightarrow S \preceq S'$ and $S' \preceq S$. This justifies abbreviations like $\bigsqcup \mathcal{D} \preceq S'$ for $\exists S \in \bigsqcup \mathcal{D}, S \preceq S'$. The following theorem relates the order relation $\preceq$ to the meaning of a graph schema, $[S]$:

Theorem 7. *If $\mathcal{D} = [S]$ then $S \in \bigsqcup \mathcal{D}$.*

Before proving this result, we prove a corollary:

Corollary 8. *$S \preceq S'$ iff $[S] \subseteq [S']$. Hence S, S' are equivalent iff $[S] = [S']$.*

Proof. Obviously, $S \preceq S' \Longrightarrow [S] \subseteq [S']$. The converse follows from Theorem 7, because $[S] \subseteq [S']$ implies $\bigsqcup[S] \preceq \bigsqcup[S']$, hence $S \preceq S'$.

Together, Corollary 8 and Proposition 6 imply that $[S] \subseteq [S']$ and $[S] = [S']$ are decidable in polynomial time. The rest of this subsection contains the proof of

Theorem 7, in which we approximate graph databases with trees. A *tree database* is a database whose graph is a finite tree. For a database DB, the *approximations* of DB is the set appr $(DB) = \{TDB \mid TDB$ a TDB $\preceq DB\}$. When DB is cycle-free, then appr (DB) is a finite set; when DB is a tree database itself, then $DB \in$appr (DB). When DB has cycles, appr (DB) is infinite, and can be thought of as the set of all finite unfoldings of DB. Approximations allow us to infer simulations:

Proposition 9. appr $(DB) \subseteq$appr (DB') *iff* $DB \preceq DB'$.

Proof. $DB \preceq DB'$ implies appr $(DB) \subseteq$appr (DB'). For the converse, let u be some node in DB, and DB_u be the same graph database DB, but whose root is u. More precisely, when $DB = (V, E, v_0)$ then $DB_u = (V, E, u)$. We define the relation $\preceq$ from the nodes of DB to those of DB' to be $u \preceq u'$ iff appr $(DB_u) \subseteq$appr $(DB'_{u'})$. Obviously, $v_0 \preceq v'_0$, where v_0, v'_0 are the roots of DB, DB' respectively. Now we have to prove that $\preceq$ is a simulation. Assume $u \preceq u'$ and let $u \xrightarrow{a} v$ be an edge in DB. The tree $(\{u, v\}, \{(u, a, v)\}, u)$ (consisting of a single edge $u \xrightarrow{a} v$ with root u) is in appr (DB_u), hence it is in appr $(DB'_{u'})$, so there exists at least one a-labeled edge leaving u'. Let $u' \xrightarrow{a} v'_1, \ldots, u' \xrightarrow{a} v'_k$ be the set of all such edges, $k \geq 1$. We use the fact that this set is finite and show that there exists some i s.t. appr $(DB_v) \subseteq$appr $(DB'_{v'_i})$, implying $v \preceq v'_i$. Suppose by contradiction that this is not true: then for each $i = 1, k$ there exists some tree database $TDB_i \in$appr (DB_v) s.t. $TDB_i \notin$appr $(DB'_{v'_i})$. Consider the tree $TDB = \{a \Rightarrow (TDB_1 \cup \ldots \cup TDB_k)\}$. We have $TDB \in$appr (DB_u), but $TDB \notin$appr $(DB'_{u'})$ – a contradiction.

This proposition also holds for some infinite databases. Let us call some infinite database, DB, *label finite* if for any node u and label a, the set of outgoing edges $u \xrightarrow{a}$ is finite. From the proof of Proposition 9, we derive:

Corollary 10. *Let* appr $(DB) \subseteq$appr (DB'), *with* DB, DB' *possibly infinite databases, but with* DB' *label-finite. Then* $DB \preceq DB'$.

Example 1. Let $DB = \{a \Rightarrow \{0, 1, 2, \ldots\}\}$ and $DB' = \{a \Rightarrow t_0, a \Rightarrow t_1, a \Rightarrow t_2, \ldots\}$, where $t_k = \{0, 1, \ldots, k-1, k+1, k+2, \ldots\}$. Then appr (DB) =appr (DB') but $DB \not\preceq DB'$, proving that Corollary 10 fails when DB' is not label finite.

We now prove Theorem 7 using Proposition 9. We extend the notation appr to graph schemas, i.e. appr $(S) = \{TDB \mid TDB \preceq S, TDB$ is a tree d.b.$\}$ =appr (S^∞). Suppose S' satisfies $\forall DB \in \mathcal{D}$, $DB \preceq S'$: we have to prove $S \preceq S'$. First we show appr $(S) \subseteq$appr (S'): $TDB \preceq S \Longrightarrow TDB \in \mathcal{D} \Longrightarrow TDB \preceq S' \Longrightarrow TDB \in$appr (S'). Now we observe that S'^∞ is label-finite, hence Corollary 10 implies $S^\infty \preceq S'^\infty$. Finally Proposition 5 implies $S \preceq S'$.

4.2 GLB's and LUB's of graph schemas

Next, we show how to construct a schema S that expresses the combined constraints of two graph schemas S_1 and S_2. Given two schemas S_1 and S_2, we show that there exists a schema S s.t. $[\![S]\!] = [\![S_1]\!] \cap [\![S_2]\!]$. Take the nodes of S to be pairs (u_1, u_2), with u_i a node in S_i, $i = 1, 2$, and take edges to be $(u_1, u_2) \xrightarrow{p_1 \wedge p_2} (v_1, v_2)$, for any two edges $u_i \xrightarrow{p_i} v_i$ in S_i, $i = 1, 2$. One can show $[S] = [S_1] \cap [S_2]$. It follows that S is the greatest lower bound of S_1 and S_2, in notation $S_1 \sqcap S_2$. For example, when S_1, S_2 are given by Fig. 4(d) and (e), then $S_1 \sqcap S_2$ is given by the schema in (c) which is equivalent to that of (f), assuming the predicates *isDept* and *isPaper* are disjoint.

A similar fact does not hold for union or complement. Let us say that a set $\mathcal{D}$ of databases is *representable* if it is of the form $\mathcal{D} = [S]$ for some graph schema S. Then it is easy to show that any representable set $\mathcal{D}$ is an *ideal* [Gun92], i.e.: (1) $\mathcal{D}$ is nonempty, (2) $\mathcal{D}$ is downwards closed, i.e. $DB \preceq DB'$ and $DB' \in \mathcal{D}$ implies $DB \in \mathcal{D}$, and (3) $\mathcal{D}$ is directed, i.e. $DB_1, DB_2 \in \mathcal{D}$ implies $\exists DB \in \mathcal{D}$ s.t. $DB_1 \preceq DB$ and $DB_2 \preceq DB$. It follows immediately that, if $\mathcal{D}_1$ and $\mathcal{D}_2$ are representable, then the complement of $\mathcal{D}_1$ and $\mathcal{D}_1 \cup \mathcal{D}_2$ are, in general, not representable. Let $idl(\mathcal{D})$ denote the ideal generated by the set $\mathcal{D}$, i.e. $idl(\mathcal{D}) = \{DB_1 \cup \ldots \cup DB_k \mid \exists DB'_1, \ldots, DB'_k \in \mathcal{D}, \text{ s.t. } DB_i \preceq DB'_i, i = 1, k\}$. Then we can prove that when $\mathcal{D}_1, \mathcal{D}_2$ are representable, so is $idl(\mathcal{D}_1 \cup \mathcal{D}_2)$. For S_1, S_2 graph schemas representing $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively, we define S to be their union(Section 2). It follows that $[S] = idl([S_1] \cup [S_2])$ and that S is the least upper bound of S_1, S_2, in notation $S_1 \sqcup S_2$.

4.3 Fragments of databases

Finally, we address the problem of finding for some database DB and graph schema S, a canonical "fragment" DB_0 of DB such that $DB_0 \preceq S$. This is important if we wish to use graph schemas as data guides [Abi97]. Instead of insisting that a database DB strictly conforms to some schema S, we require that there be a "large fragment" of DB which conforms to S. By "fragment" we mean a database DB_0 s.t. $DB_0 \preceq DB$. The name "fragment" is justified, because whenever $DB_0 \preceq DB$, there exists some graph DB' which is bisimilar to DB (hence, DB and DB' denote the same data) of which DB_0 is a subgraph. E.g. consider the graph schema S in Fig. 4 (a), and let $DB = \{tup \Rightarrow \{A, D \Rightarrow \{3\}\}\}$ be the database in Fig. 2(c). Then $DB_0 = \{tup \Rightarrow \{A\}\}$.

We observe that for any DB, S, the empty database $\emptyset$ (one node, no edges) is a fragment satisfying the requirement above, i.e. $\emptyset \preceq DB$ and $\emptyset \preceq S$. This is not the "canonical" fragment we want, because it is not the largest fragment under the simulation relation $\preceq$. By taking $DB_0 \stackrel{\text{def}}{=} DB \sqcap S$ we can prove:

Proposition 11. *For any graph database DB and graph schema S, there exists some database DB_0 s.t. (1) $DB_0 \preceq DB$ and $DB_0 \preceq S$, and (2) for any other database DB'_0 satisfying this property, $DB'_0 \preceq DB_0$. Moreover DB_0 can be computed in PTIME, and if $DB \preceq S$ then DB_0 is bisimilar to DB. We call DB_0 the* canonical fragment *of DB satisfying S.*

5 Determinism

Nodes in a schema have the potential to classify nodes in a database. This could be useful, for example, in a distributed environment, where we could use a schema to describe how such a database is distributed. For example, suppose that the database DB in Fig. 1 is distributed on two sites, such that all nodes before a department edge are located on site 1, while those after a department edge are on site 2. We could describe this formally using the schema in Fig. 4(d), which has two nodes u', v': database nodes conforming to u' will be on site 1, while those conforming to v' on site 2. However, the schema in Fig. 1(e) does not classify the nodes uniquely, because whenever we encounter an edge $u \xrightarrow{a} v$ in DB such that *isDept*(a), we may either follow the edge $u' \xrightarrow{isDept} v'$ or the edge $u' \xrightarrow{not(isPaper)} u'$ in the schema. We say that the first schema is *deterministic*, while the second one is not.

In object-based graph database models, determinism is natural. For example, the semantics of ACeDB trees imposes that instance databases be deterministic, and in the Tsimmis data model, each node has a unique object identifier making the instance database deterministic. In our graph model, however, a deterministic representation of relational databases requires adding unnecessary object identifiers to sets. For example, in order to make the tree representation of a relational database in Fig. 2(a) deterministic we would use a different object identifier for every *tup* edge, say *tup*1, *tup*2, *tup*3. Determinism for graph schemas in any model, however, is natural. Note that the tree representation of the relational graph schema in Fig. 4 (a) for the database of Fig. 2(a) *is* deterministic.

We show that certain nondeterministic schemas are not equivalent to any deterministic ones. A natural question arises then: given a nondeterministic schema S, how can we best approximate it with a deterministic schema S_d ? We show here that a canonical S_d always exists.

We call an edge-labeled graph G *deterministic* if for every node x and label a, there exists at most one edge labeled a going out of x. This definition is not invariant under bisimulation[3]. A database DB is deterministic if there exists some deterministic graph bisimilar to it. Similarly, we call a graph schema S *deterministic* iff S^∞ is deterministic. The following is a sufficient condition for checking if a graph schema S is deterministic:

Proposition 12. *Let S be a graph schema. S is deterministic if for any node u and any two distinct edges $u \xrightarrow{p} v, u \xrightarrow{p'} v'$, we have $\mathcal{U} \models \neg(\exists x.p(x) \wedge p'(x))$.*

Deterministic graph schemas are important because of the following:

Proposition 13. *Let S be deterministic and TDB a tree database s.t. $TDB \preceq S$. Then TDB conforms to S "in a unique way". More precisely there exists a function φ from the nodes of TDB to those of S s.t. for any simulation $\preceq$ from TDB to S, and for every node u of TDB, $u \preceq \varphi(u)$.*

This follows from the observation that nodes in a tree database are in 1-1 correspondence with sequences of labels, $a_1 \dots a_n$. Such a sequence is mapped uniquely into some node in S, because S is deterministic, and this defines the function φ. $\varphi(u)$ classifies nodes: u and v are in the same class iff $\varphi(u) = \varphi(v)$.

Deterministic schemas are less "expressive" than nondeterministic ones. For example, the nondeterministic graph schema $S = \{a \Rightarrow \{b\}, a \Rightarrow \{c\}\}$ is not equivalent to any deterministic graph schema, i.e. $[S] \neq [S_d]$ for any deterministic graph schema S_d. The "closest" we can get is the deterministic graph schema

[3] The tree $\{a\}$ is deterministic and bisimilar to the tree $\{a, a\}$; but the latter is not deterministic.

$S_d = \{a \Rightarrow \{b, c\}\}$. In general, for any nondeterministic graph schema S, there exists a "closest" deterministic graph schema S_d. The latter is constructed in a way reminiscent of the DFA equivalent to an NDFA:

Proposition 14. *For any graph schema S, there exists some deterministic graph schema S_d with the following properties: (1) $S \preceq S_d$, (2) whenever $S \preceq S'$ and S' is deterministic then $S_d \preceq S'$.*

The proof is based on a standard powerset construction and is given in [BDFS96].

An interesting case is when S is a database (i.e. all unary formulas on its edges are equalities with constants); then S_d is precisely the deterministic automata obtained from S. For the example in which $S = \{a \Rightarrow \{b\}, a \Rightarrow \{c\}\}$, we get $S_d = \{a \Rightarrow \{b, c\}\}$.

In general, the number of nodes in S_d is exponential in that of S. But when S is a tree database, then the number of nodes in S_d is less than or equal to that of S [Per90, pp.7]. When we generalize to unary formulas, then the number of nodes in S_d may be exponential, even when S is a tree. For example, let $S = \{p_1, p_2, \ldots, p_n\}$, then $S_d = \{r_0, r_1, \ldots, r_{2^n-1}\}$, where each $r_i = \bigvee_{j=0,n-1} q_j$, with $q_j = p_j$ or $q_j = \neg p_j$, depending on whether the j's bit in the binary representation of i is 1 or 0. Such arbitrary sets of unary formulas $p_1, p_2, \ldots, p_n$ rarely occur in practice, because the base predicates are either constants, or taken from a list of disjoint predicates, like *Int*, *String*, *Bool*, *Nat*, *isDept*. The graph schemas in Figure 4 have this property. Then we can prove:

Proposition 15. *Let S be a tree schema in which for every two distinct unary formulas $p(x), p'(x)$, either is a constant (i.e. of the form $x = a$), or they are disjoint (i.e. $\mathcal{U} \models \neg\exists x.(p(x) \wedge p'(x))$). Then S_d has at most as many nodes as S, and can be computed in polynomial time.*

6 Graph Schemas and Queries

In [BDHS96a], we propose UnQL, a language for querying and restructuring graph databases. UnQL is compositional, has a simple select ... where ... construct, supports flexible path expressions, and can express complex restructuring of the graph database. Consider the simple UnQL query Q:

$$\text{select } \{x \Rightarrow \{x\}\} \text{ where } \backslash x \leftarrow DB$$

Q takes a graph database of the form $\{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$ and returns the graph database $\{a_1 \Rightarrow \{a_1\}, \ldots, a_n \Rightarrow \{a_n\}\}$, i.e., Q doubles each edge in the first level of edges in DB.

Recall from Section 2 that graph schemas can be thought of as finite descriptions of infinite sets of databases, i.e. S defines the set $[S] = \{DB \mid DB \preceq S\}$. We consider whether, given a schema S and an UnQL query Q, we can describe the set $\{Q(DB) \mid DB \preceq S\}$ by a schema S'. This question is important for two reasons. First, we use graph schemas in query optimization of UnQL. Since UnQL is compositional, when we optimize a composed query $Q(DB) \stackrel{\text{def}}{=} Q_2(Q_1(DB))$ whose input conforms to some graph schema, $DB \preceq S$, we first optimize Q_1 according to graph schema S, then optimize Q_2 according to the graph schema of the set $\{Q_1(DB) \mid DB \preceq S\}$, hence the need to compute the latter. Second, UnQL queries can be used to define views, like $V \stackrel{\text{def}}{=} Q(DB)$. Given that $DB \preceq S$, we want to optimize queries against the view. This requires a graph schema for the set $\{Q(DB) \mid DB \preceq S\}$.

Given a graph schema S and a query Q, there is a natural way to compute a graph schema $Q(S)$, with the property: $(*)\ \forall DB \preceq S,\ Q(DB) \preceq Q(S)$. Since UnQL queries are just graph transformations, we can compute $Q(S)$ much in the same way in which we compute $Q(DB)$. Where the construct is less obvious, we take a conservative action. For example, for a subquery $Q(DB) = \{x \Rightarrow DB\}$, having a free variable x bound in a surrounding context, we define $Q(S)$ to be $\{true \Rightarrow S\}$, or if any predicate $P(x)$ is known about the variable x (e.g. Q occurs in the then branch of an if $P(x)$ then ... else ... construct), then we take $Q(S) = \{P \Rightarrow S\}$. This ensures that $(*)$ holds, but $Q(S)$ may not necessarily get the tightest description of the set $\{Q(DB) \mid DB \preceq S\}$.

We omit the full description of $Q(S)$ from this abstract, but mention that $Q(S)$ can be computed in PTIME, and that it satisfies $(*)$. But $(*)$ can be trivially satisfied by taking $Q(S) = S_{\top}$ (Fig. 4 (b)), which is a maximal element in the partial order $\preceq$. We would like to make the claim $Q(S) = \bigsqcup\{Q(DB) \mid DB \preceq S\}$, thus showing that $Q(S)$ describes precisely the set $\mathcal{D} \stackrel{\text{def}}{=} \{Q(DB) \mid DB \preceq S\}$. Unfortunately, this does not hold. Worse, there are examples of simple queries Q and graph schema S for which $\bigsqcup\mathcal{D}$ does not exist. Consider the graph schema $S = \{Nat\}$ and the UnQL query Q from above. This query doubles every label in the database, e.g. on the database $DB = \{2, 4, 5\}$ Q returns $\{2 \Rightarrow \{2\}, 4 \Rightarrow \{4\}, 5 \Rightarrow \{5\}\}$. Our method computes the graph schema $S' = Q(S)$ to be $\{Nat \Rightarrow \{Nat\}\}$, but this is not $\bigsqcup\mathcal{D}$. The sequence of graph schemas $S_1, S_2, \ldots$ where $S_n = \{0 \Rightarrow \{0\}, 1 \Rightarrow \{1\}, \ldots, n \Rightarrow \{n\}, p_n \Rightarrow \{Nat\}\}$, with $p_n(x) = (x \neq 0 \land \ldots \land x \neq n \land Nat(x))$, forms an infinite, strictly descending chain of graph schemas, each offering a better approximation of $\mathcal{D}$. In fact, we can prove directly that $\mathcal{D}$ has no least upper bound.

Graph schemas cannot describe all sets of the form $\{Q(DB) \mid DB \preceq S\}$, because they cannot impose equality constraints on edges in the database. We can partially fix this by extending the notion of graph schema to allow equality constraints between certain values on edges. Formally, we define an *extended graph schema* with variables $z_1, \ldots, z_n$ to be a rooted graph (V, E, v_0), in which the edges are labeled with formulas as explained below, and with $n \geq 0$ distinguished subgraphs, denoted $G_{z_1}, \ldots, G_{z_n}$. Each subgraph G_z is called the *scope* of the variable z, and is given by (1) a set of nodes $V_z \subseteq V$, (2) a set of edges $E_z \subseteq E$, s.t. for every edge $u \to v$ in E_z, both u and v are in V_z, (3) a set of input nodes $I_z \subseteq V_z$, and (4) a set of output nodes $O_z \subseteq V_z$. We impose several conditions on extended graph schemas: (1) For every edge $u \to v$ entering some graph G_z (i.e. $u \notin V_z$ and $v \in V_z$), v is one of the inputs of G_z. (2) Similarly, every edge $u \to v$ leaving some graph G_z exits from an output node, $u \in O_z$. (3) Each formula labeling some edge in the scope of k variables $z_1, \ldots, z_k$ may have $k+1$ free variables: $z_1, \ldots, z_k$ and a distinguished variable x as before. (4) The scopes of variables follow traditional rules in programming languages: for $z \neq z'$, either $G_z \subseteq G_{z'}$, or $G_{z'} \subseteq G_z$, or G_z and $G_{z'}$ are disjoint.

Graph schemas are particular cases of extended graph schemas with no variables ($n = 0$). As with graph schemas, an extended graph schema S can be modeled by its infinite expansion S^{∞}. Each graph G_z is replicated once for each value $z \in \mathcal{U}$, and their input and output nodes are collapsed. Fig. 7 contains two examples of extended graph schemas with one variable z. I_z has a single node in both (a) and (b); O_z is empty in (a) and has one node in (b). The expansion in (b) is incomplete: *not*(0) should be further expanded with all atoms $a \in \mathcal{U}$, $a \neq 0$, etc. Unlike graph schemas, S^∞ may have infinitely many nodes. Some care is needed when collapsing the input and output nodes. In a formal definition presented elsewhere, we use ε edges to define S^∞ (see [BDHS96a] for a definition of ε edges).

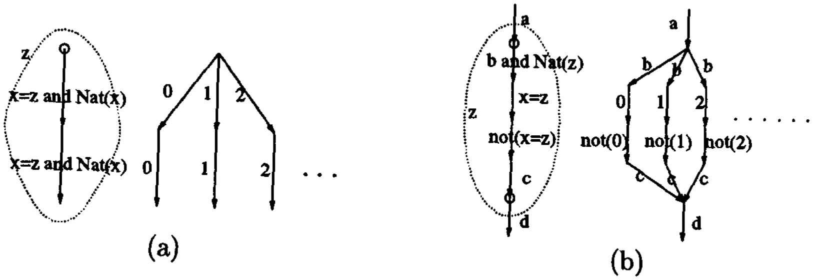


Fig. 7. Two examples of extended graph schemas and their expansions.

Since extended graph schemas are a more sophisticated way of specifying an infinite graph, we can extend previous results for graph schemas. We can define what it means for a database DB to conform to an extended graph schema S, $DB \preceq S$, and for an extended graph schema S to subsume some other extended graph schema S', $S \preceq S'$, etc. From [HHK95], both $DB \preceq S$ and $S \preceq S'$ are decidable. Unfortunately, S^∞ is not generally label-finite, and Theorem 7 fails in general for extended graph schemas. For example, take $S = \{a \Rightarrow \{Nat\}\}$. Then $S^\infty = DB$ with DB from Example 1, and $[S] = \text{appr}\ (DB)$. Take S' to be the graph $G = G_z = \{u \stackrel{x=a \wedge Nat(z)}{\longrightarrow} v \stackrel{Nat(x) \wedge x \neq z}{\longrightarrow} w\}$ with $I_z = \{u\}$ and $O_z = \emptyset$, then $S'^\infty = DB'$ of Example 1, and S' is an upperbound of $[S]$ but $S \not\preceq S'$. Intuitively, S' is better than $S = \{a \Rightarrow \{Nat\}\}$ because it says that after each a-edge, at least one natural number is missing. Using two variables z_1, z_2 we can say that at least two natural numbers are missing, etc. In fact the set $[S]$ does not have a least upper bound in the preordered set of extended schemas. Fortunately, we can address this problem if we are restricted to deterministic extended graph schemas. More precisely, we can prove the following theorem, which is the most complex result of this paper. Here a *positive* UnQL query is a query whose translation into UnCAL does not use the only non-monotone operator in UnCAL, *isempty* (see [Suc96] for a more detailed discussion).

Theorem 16. *Let Q be a positive UnQL query. Then for every (extended) graph schema S there exists an extended graph schema $Q(S)$, computable in PTIME such that: for every deterministic, extended graph schema S', if $\forall DB \preceq S \Rightarrow Q(DB) \preceq S'$, then $Q(S) \preceq S'$.*

The proof appears in [BDFS96]. For the UnQL query Q at the beginning of this section and schema $S = \{Nat\}$, $Q(S)$ is the schema in Fig. 7 (a).

7 Conclusions and Future Work

When querying unstructured data, the ability to use whatever structure is known about the data can have significant impact on performance. Examples abound in optimizations for generalized path expression (see [CACS94, CCM96], among others). We have explored a new notion of a graph schema appropriate for edge-labeled graph databases. Since the known structure of graph databases may be weaker than that of a traditional database, we use unary formulas instead of constants for edge labels. We describe how a graph database conforms to a schema and observe that a graph database may conform to multiple schemas. Since there is a natural ordering on graph schemas, it is possible to take the least upper bound of a set of schemas and combine into a single schema all their constraints. We then describe a "deterministic" subclass of schemas that uniquely classifies nodes of (tree) databases. When optimizing queries for distributed graph databases, node classification allows us to decompose and specialize the query for a target site [Suc96].

In current work, we are using schemas for query optimization and query decomposition. Consider the following UnQL query Q [Suc96], which selects all papers in the Computer Science Department in Fig. 1:

$$\text{select } \textit{“Papers”.t} \text{ where } \_*.\textit{“CS-Department”}.\_*.\textit{“Papers”.t} \leftarrow DB$$

Without any knowledge about the data's structure, one has to search the entire database. We can exploit knowledge about the structure of the data in order to prune the search. For example, if we know that the data conforms to the the schema in Fig. 4(d), we can prune the search after every department edge that is not a Computer Science Department. This can be described by another query, Q_{opt}. An interesting question is what happens if the database DB fails to conform to the schema S, which is likely in unpredictable data sources like the Web. As discussed in Subsection 4.3, one can still describe the precise semantics of $Q_{\text{opt}}(DB)$, namely as $Q(DB_0)$, where DB_0 is the canonical fragment of DB conforming to S (Subsection 4.3). Similarly, we plan to address query decomposition. [Suc96] describes a query decomposition technique that ignores any information about the structure of the data, or how it is distributed. Assuming the database DB is distributed on two sites, the technique in [Suc96] poses three different queries on each site. We plan to use deterministic schemas to describe data in a distributed environment. For example, we could use the schema in Fig. 4(d) to describe how the nodes in the database are located on the two sites and reduce the queries posed at each site from three to one. Maximizing the benefits of these techniques for query decomposition and optimization is an area of future work.

The definition of a graph schema we have given is extremely general. For example, it cannot constrain a graph to be an instance of a relation in the sense that Fig. 2(a) describes a relation, because multiple edges with the same attribute name are allowed in the graph instance. Furthermore, our schemas only place outer bounds on what edges may emanate from a node. In future work, we may consider a dual notion of schema that places inner bounds on edges by requiring certain edges to exist. One could consider further constraints that restrict the number of edges that emanate from a node, as is done in [TMD92] to model variants.

References

[Abi97] Serge Abiteboul. Querying semi-structured data. In *ICDT*, 1997.

[BDFS96] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. Technical Report MS-CIS-96-21, University of Pennsylvania, Computer and Information Science Department, 1996.

[BDHS96a] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD*, 1996.

[BDHS96b] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. Technical Report 96-09, University of Pennsylvania, Computer and Information Science Department, February 1996.

[BDS95] Peter Buneman, Susan Davidson, and Dan Suciu. Programming constructs for unstructured data. In *Proceedings of DBPL'95*, Gubbio, Italy, September 1995.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In Richard Snodgrass and Marianne Winslett, editors, *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 1994.

[CCM96] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating queries with generalized path expressions. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[CM90] M. P. Consens and A. O. Mendelzon. Graphlog: A visual formalism for real life recursion. In *Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys.*, Nashville, TN, April 1990.

[Gun92] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. MIT Press, 1992.

[HHK95] Monika Henzinger, Thomas Henzinger, and Peter Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of 20th Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[KS95] David Konopnicki and Oded Shmueli. Draft of W3QS: a query system for the World-Wide Web. In *Proc. of VLDB*, 1995.

[MMM96] SuA. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. Manuscript, available from `http://www.cs.toronto.edu/ georgem/WebSQL.html`, 1996.

[Per90] D. Perrin. Finite automata. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 1, pages 1–57. Elsevier, Amsterdam, 1990.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, March 1995.

[PT87] Robert Paige and Robert Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16:973–988, 1987.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructure heterogeneous information. In *International Conference on Deductive and Object Oriented Databases*, 1995.

[Suc96] Dan Suciu. Query decomposition for unstructured query languages. In *VLDB*, September 1996.

[TMD92] J. Thierry-Mieg and R. Durbin. Syntactic Definitions for the ACEDB Data Base Manager. Technical Report MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge,CB2 2QH, UK, 1992.