December 1996

Adding Structure to Unstructured Data

Peter Buneman
*University of Pennsylvania*, peter@cis.upenn.edu

Susan B. Davidson
*University of Pennsylvania*, susan@cis.upenn.edu

Mary Fernandez
*University of Pennsylvania*

Dan Suciu
*University of Pennsylvania*

Follow this and additional works at: [http://repository.upenn.edu/cis_reports](http://repository.upenn.edu/cis_reports)

Recommended Citation

Peter Buneman, Susan B. Davidson, Mary Fernandez, and Dan Suciu, "Adding Structure to Unstructured Data", December 1996.


This paper is posted at ScholarlyCommons. [http://repository.upenn.edu/cis_reports/192](http://repository.upenn.edu/cis_reports/192)

For more information, please contact libraryrepository@pobox.upenn.edu.
Adding Structure to Unstructured Data

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.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-
CIS-96-21.

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/192
Adding Structure to Unstructured Data

MS-CIS-96-21

Peter Buneman, Susan Davidson, Mary Fernandez, Dan Suciu

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1996
Adding Structure to Unstructured Data

Peter Buneman    Susan Davidson    Mary Fernandez    Dan Suciu

December 19, 1996

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 sub-graphs. 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 $p(a)$ holds. This notion of conformance
Figure 1: A fragment of http://www.ucsd.edu.

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 (\( \mathcal{U} = \text{Int} \cup \text{String} \cup \text{Bool} \cup \ldots \)). 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, \( E \subseteq V \times \mathcal{U} \times V \), and \( v_0 \in V \) is a distinguished root. Fig. 1 is an example of a graph database. We denote an edge with \( v \rightarrow u \), instead of the official \( (u, a, v) \in E \). This model is powerful enough to encode relational databases, as illustrated in Fig. 2(a), which encodes a relation \( R(A : \text{Int}, B : \text{Int}, C : \text{String}) \), but flexible enough to represent unstructured data, like Fig. 2(b) and (c). There is no distinction in this model between set, record, and variant nodes. 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 \( u \rightarrow v \) in \( DB \), there exists \( u' \rightarrow v' \) in \( DB' \) such that \( v \approx v' \), and for every \( u' \rightarrow v' \) in \( DB' \), there exists \( u \rightarrow v \) 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 \( \{\text{tup} \rightarrow \{A \rightarrow \{3\}\}\} \). Also, we defined a union operation on two graph databases \( DB_1 \cup DB_2 \) 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 \( DB_1 \cup DB_2 = \{a \Rightarrow \{b\}, c, a \Rightarrow \{d\}\} \).

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

**Definition 2.1** A graph schema is a rooted, labeled graph, in which the edges are labeled with
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 $\text{Int}(x), \text{String}(x), \text{Nat}(x), \text{Bool}(x)$, which denote $x \in \text{Int}, x \in \text{String}, x \in \text{Nat}, x \in \text{Bool}$, and user-defined unary predicates, $P(x)$. The theory has an equality operator, so we have predicates such as $x = 5$ and $x = \text{"abc"}$. Such a theory is decidable [DG79].

Fig. 1(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 $A, \text{Int}$, and $\text{Int} \lor \text{String}$ instead of $x = A, \text{Int}(x)$, and $\text{Int}(x) \lor \text{String}(x)$ respectively. 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 $\text{tup}$-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, $\text{isDept}(x)$ and $\text{isPaper}(x)$ 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 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. We will comment later about schemas (c) and (f).

Schema (b) in Figure 4 is of special interest: it says that the database may have any labels whatsoever (they have to satisfy $\text{true}$), thus it does not impose any structure on the database. We denote it with $S_T$.

**Definition 2.2** A database $DB$ conforms to a graph schema $S$, in notation $DB \preceq S$, 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:
Figure 4: Six examples of graph schema.

(A) "tup" to "Int", "Int", "String"
(B) "true"
(C) "not(isDept)" "isDept" "not(isDept)"
(D) "not(isDept)"
(E) "not(isPaper)"
(F) "not(isDept)"

Figure 5: A graph schema $S$ and its infinite expansion $S^\infty$.

\begin{align*}
\text{Nat} & \rightarrow \text{A} \rightarrow \text{Nat} \\
0 & \rightarrow 1 \rightarrow 2 \rightarrow 3 \\
\vdots & \\
\vdots & \\
\end{align*}

(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 $u' \xrightarrow{a} v'$ 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 multiple occurrences of 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 $x = a$. In particular this 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: e.g. take $DB = \{a \Rightarrow \{b, c\}, a \Rightarrow \{b\}\}$ and $DB' = \{a \Rightarrow \{b, c\}\}$.

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 $S^\infty$, 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 $S^\infty$.

One can easily check that for any database $DB$ and graph schema $S$, $DB \preceq S$ iff $DB \preceq S^\infty$,
where 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. Their algorithm can easily be adapted to test 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 $x \rightarrow y$ into two unlabeled edges $x \rightarrow z \rightarrow y$, 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 3.1** 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 $O(m^2t)$. Here $m$ is the total number of edges in $DB$ and $S$, which are each assumed to be connected.

**Proof:** Consider the algorithm EfficientSimilarity of [HHK95]. The only place where it checks equality of node labels is in the initialization phase. Given $DB$ and $S$, we replace this initialization step with one which compares whether a label $a$ from $DB$ and a predicate $p$ from $S$ satisfy $U \models p(a)$; this can be done in time $t$. The rest of the algorithm remains unchanged. \hfill $\square$

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 $S_T$ (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), in that any database conforming to the schema in (f) will also conform
to the schemas in (d) and (e). Formally, we define $[S] = \{ DB \mid DB \preceq S \}$. Given two schemas $S, S'$, 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 $DB \not\preceq S$ and $DB \not\preceq S'$. We show that both $DB \npreceq S$ and $DB \npreceq S'$ be checked in polynomial time.

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 has~d 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 when applied to some $DB$ which fails to conform to $S$. 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 $DB \not\preceq S$, then $DB_0$ is $DB$. We can now state what we expect from an optimizer. Given a query $Q$ on schema $S$, we expect a correct optimizer to produce an optimized query $Q_{opt}$ such that for any database $DB$,

$$Q_{opt}(DB) = Q(DB_0)$$

This implies that $Q_{opt}(DB) = Q(DB)$ whenever $DB \preceq S$.

We address these three issues in the sequel.

### 4.1 Subsumption of graph schemas

We define schema subsumption and equivalence as follows.

**Definition 4.1** 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 $a \in U$ s.t. $U \models p(a)$, there exists an edge $u' \xrightarrow{p'} v'$ in $S'$ s.t. $U \models p'(a)$ and $v \preceq v'$. $S$ and $S'$ are equivalent if $S \preceq S'$ and $S' \preceq S$.

When $S, S'$ are databases (i.e. each predicate is equality with a constant), then the subsumption relation coincides with the simulation relation between databases.

Recall that a graph schema $S$ represents its possibly infinite expansion, $S^\infty$, i.e., an edge $x \xrightarrow{p} y$ represents infinitely many edges, one for each $a$ for which $U \models p(a)$. Each such edge may be simulated in $S'$ by some unary formula. First, we choose $a \in U$, then decide which edge $x' \xrightarrow{p'} y'$ in $S'$ will “mimic” the edge $x \xrightarrow{p} y$ in $S$. To justify our choice for condition (2), consider the example in Figure 6, where $S = \{ Nat \vee String \Rightarrow \{ 5 \} \}, S' = \{ Nat \Rightarrow \{ Nat \}, String \Rightarrow \{ Nat \} \}$. Clearly $S^\infty \preceq S'^\infty$, so we expect $S \preceq S'$, and indeed condition (2) is satisfied when $u, u'$ the roots: for any $a \in U$ s.t. $Nat(a) \vee String(a)$, we can have either $Nat(a)$ or $String(a)$. In the first case we “move” to the left in $S'$ (i.e. pick $v'$ to be the left node) else we “move” to the right. Now consider a more rigid choice for condition (2) above, requiring: $\forall u \leq u'$, and for every labeled edge $u \xrightarrow{p} v$ in $S$, there exists an edge $u' \xrightarrow{p'} v'$ in $S'$ s.t. $U \models p \Rightarrow p'$. Then we would not have had $S \preceq S'$, because neither $Nat \vee String \Rightarrow Nat$ nor $Nat \vee String \Rightarrow String$ holds.

**Proposition 4.2** $S \preceq S'$ iff $S^\infty \preceq S'^\infty$. The latter is the simulation relation between (possibly infinite) databases.
Let $R \leftarrow \{(u, u') \mid u \in \text{nodes}(S), u' \in \text{nodes}(S')\}$

while any change do

find $(u, u') \in R$ and edge $u \xrightarrow{p} v$ in $S$

such that $U \models \exists a. (p(a) \land (\bigwedge_{i=1,k} \neg p_i'(a)))$

where $u_i' \xrightarrow{p_i'} v_i'$, $i = 1, k$ are all edges from $u'$ in $S'$

$R \leftarrow R - \{(u, u')\}$

return $((v_0, v_0') \in R)$

Figure 6: Simulation between two graph schemas $S$ and $S'$.

Proof: Suppose $S \preceq S'$ and let $\preceq$ be a subsumption relation. $S^\infty$ and $S'^\infty$ have the same nodes as $S, S'$ respectively: we prove that $\preceq$ satisfies conditions (1) and (2) in Definition 2.2. (1) is immediate. For (2) let $u \preceq u'$ and let $u \xrightarrow{a} v$ in $S^\infty$. That is, there exists $u \xrightarrow{a} v$ in $S$ s.t. $U \models p(a)$.

By (2) of Definition 4.1 there exists $u' \xrightarrow{p'} v'$ in $S'$ s.t. $U \models p'(a)$ and $v \preceq v'$. But this implies that there exists an edge $u' \xrightarrow{a} v'$ in $S'^\infty$.

Conversely, let $S^\infty \preceq S'^\infty$, and let $\preceq$ be a simulation relation. We show that $\preceq$ satisfies condition (2) of Definition 4.1 (condition (1) is immediate). So let $u \preceq u'$, $u \xrightarrow{a} v$ in $S$ and $U \models p(a)$. Then $a \xrightarrow{u} v$ is an edge in $S^\infty$ hence, by Definition 2.2, there exists in $S'^\infty$ an edge also labeled $a$, $u' \xrightarrow{a} v'$ s.t. $v \preceq v'$. It follows immediately that $S'$ must have some edge $u' \xrightarrow{p'} v'$ s.t. $U \models p'(a)$. \[\square\]

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 consider the problem of determining whether $S \preceq S'$. From [HHK95], this problem is decidable. Moreover, our algorithm in Fig. 7 checks whether $S \preceq S'$ in polynomial time.

Proposition 4.3 The algorithm in Fig. 7 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 4.5 below, 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 4.2), and this allows us to define the
least upper bound (lub) of a set of graph schemas, as in any preordered set. We review here the definition of a lub, 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) whenever another graph schema \( S' \) has this property, it follows that \( S \preceq S' \). We use \( \bigcup \mathcal{D} \) for the set of least upper bounds of \( \mathcal{D} \). Since \( \preceq \) is a preorder rather than an order relation, \( \bigcup \mathcal{D} \) may have more than one element, but all are equivalent, i.e. \( S, S' \in \bigcup \mathcal{D} \Rightarrow S \preceq S' \) and \( S' \preceq S \). This justifies abbreviations like \( \bigcup \mathcal{D} \preceq \bigcup \mathcal{D}' \) for \( \exists S \in \bigcup \mathcal{D}, \exists S' \in \bigcup \mathcal{D'}, S \preceq S' \).

The following theorem relates the order relation \( \preceq \) to the meaning of a graph schema, \([S]\):

**Theorem 4.4** If \( \mathcal{D} = [S] \) then \( S \in \bigcup \mathcal{D} \).

Before proving this result, we prove a corollary:

**Corollary 4.5** \( S \preceq S' \) iff \( [S] \subseteq [S'] \). Hence \( S, S' \) are equivalent iff \( [S] = [S'] \).

**Proof:** Obviously, \( S \preceq S' \Rightarrow [S] \subseteq [S'] \). The converse follows from Theorem 4.4, because \( [S] \subseteq [S'] \) implies \( \bigcup[S] \subseteq \bigcup[S'] \), hence \( S \preceq S' \).

Together, Corollary 4.5 and Proposition 4.3 imply that \( [S] \subseteq [S'] \) and \( [S] = [S'] \) are decidable in polynomial time. The rest of this subsection contains the proof of Theorem 4.4. The idea is to 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 \( \text{appr} \ (DB) = \{ \text{TDB} \mid \text{TDB} a \text{ TDB} \preceq DB \} \). When \( DB \) is cycle-free, then \( \text{appr} \ (DB) \) is a finite set; when \( DB \) is a tree database itself, then \( DB \in \text{appr} \ (DB) \). When \( DB \) has cycles, \( \text{appr} \ (DB) \) is infinite, and can be thought of as the set of all finite unfoldings of \( DB \). Approximations allow us to infer simulations:

**Proposition 4.6** \( \text{appr} \ (DB) \subseteq \text{appr} \ (DB') \) iff \( DB \preceq DB' \).

**Proof:** Obviously, \( DB \preceq DB' \) implies \( \text{appr} \ (DB) \subseteq \text{appr} \ (DB') \). 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 \( \text{appr} \ (DB_u) \subseteq \text{appr} \ (DB'_u) \). 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 \( \text{appr} \ (DB_u) \), hence it is in \( \text{appr} \ (DB'_u) \), 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. \( \text{appr} \ (DB_v) \subseteq \text{appr} \ (DB'_v) \), implying \( v \preceq v'_i \).

Suppose by contradiction that this is not true: then for each \( i = 1, k \) there exists some tree database \( TDB_i \in \text{appr} \ (DB_v) \) s.t. \( TDB_i \notin \text{appr} \ (DB'_v) \). Consider the tree \( TDB = \{a \rightarrow (TDB_1 \cup \ldots \cup TDB_k)\} \). We have \( TDB \in \text{appr} \ (DB_u) \), but \( TDB \notin \text{appr} \ (DB'_u) \) – 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. In fact, we have proven a stronger version of Proposition 4.6:

**Proposition 4.7** Let \( \text{appr} \ (DB) \subseteq \text{appr} \ (DB') \), with \( DB, DB' \) possibly infinite databases, but with \( DB' \) label-finite. Then \( DB \preceq DB' \).
Without the label-finiteness condition, Proposition 4.7 fails, as illustrated in the following example.

**Example 4.8** Let \( DB = \{ a \Rightarrow \{ 0,1,2,\ldots \} \} \) and \( DB' = \{ a \Rightarrow t_0, a \Rightarrow t_1, a \Rightarrow t_2,\ldots \} \), where \( t_k = \{ 0,1,\ldots, k-1, k+1, k+2,\ldots \} \), see Figure 8. Then \( \text{appr} (DB) \nsucceq \text{appr} (DB') \) but \( DB \not\subseteq DB' \), proving that Proposition 4.7 fails when \( DB' \) is not label finite.

We now prove Theorem 4.4 using Proposition 4.7. We extend the notation \( \text{appr} \) to graph schemas, i.e. \( \text{appr} (S) = \{ TDB \mid TDB \preceq S, TDB \text{ is a tree} \} \). Suppose \( S' \) satisfies \( \forall DB \in \mathcal{D}, DB \preceq S' \). We have to prove \( S \preceq S' \). First we show \( \text{appr} (S) \subseteq \text{appr} (S') \): \( TDB \preceq S \implies TDB \in \mathcal{D} \implies TDB \preceq S' \implies TDB \in \text{appr} (S') \). Now we observe that \( S'^\infty \) is label-finite, hence Proposition 4.7 implies \( S'^\infty \preceq S'^{\infty} \). Finally Proposition 4.2 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 \( (u_1, u_2) \xrightarrow{p_1 \land p_2} (v_1, v_2) \), 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 \( S_1 \cap S_2 \).

For \( S_1 \), \( S_2 \) graph schemas representing \( D_1 \) and \( D_2 \) respectively, we define \( D \) to be their union (Section 2). It follows that \([S] = \text{idl} ([S_1] \cup [S_2]) \) and that \( S \) is the least upper bound of \( S_1, S_2 \), in notation \( S_1 \cup S_2 \).
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 $DB_0 \overset{\text{def}}{=} DB \cap S$ we can prove:

**Proposition 4.9** 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. For example, consider the database $DB$ in Figure 1, which conforms to the schema $S$ in Figure 4(d). Let us denote with $v_0, v_1$ the two nodes of schema $S$. Then $S$ classifies $DB$'s nodes into two categories: those "conforming" to $v_0$, and those "conforming" to $v_1$. Informally, the first category consists of all nodes before a Department edge, and the second category of nodes after some Department edge. But not for any schema does the classification work so nicely. For example, consider the schema in Fig. 1(e), with two nodes $u_0, u_1$, and suppose the database contains some Department without papers. Then the nodes in $DB$ following that Department link can be either classified as $u_0$ or as $u_1$. What distinguishes between the two schemas is that the first one 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 $tup1, tup2, tup3$. 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. Since we argued that deterministic schemas are more suitable than nondeterministic ones, one wonders what we may lose by restricting to deterministic schemas. We show that for any non-deterministic schema $S$, there exists a canonical $S_d$ which best "approximates" $S$. 

10
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\(^1\). A database $DB$ is deterministic if there exists some deterministic graph bisimilar to it. Similarly, we call a graph schema $S$ deterministic iff $S^\infty$ is deterministic. We will show below that testing whether a schema $S$ is deterministic is decidable. The following is a sufficient condition for checking if a graph schema $S$ is deterministic:

**Proposition 5.1** 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 $U \models \neg(\exists x. p(x) \land p'(x))$.

Deterministic graph schemas are important because of the following:

**Proposition 5.2** 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 $\varphi$ 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, $a_1 \ldots a_n$. Such a sequence is mapped uniquely into some node in $S$, because $S$ is deterministic, and this defines the function $\varphi$. Moreover, $\varphi(u)$ 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 $S_d = \{a \Rightarrow \{b, c\}\}$. 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 N DFA:

**Proposition 5.3** 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 $S_d \preceq S'$.

**Proof:** We use a powerset construct to get $S_d$. Let $S = (V, E, v_0)$. Then $S_d = (V_d, E_d, v_{0d})$, where: the nodes are nonempty sets of nodes of $S$, $V_d = \mathcal{P}(V)$, the root $v_{0d}$ is $\{v_0\}$, and the edges are defined as follows. Let $U = \{u_1, \ldots, u_n\}$, $n \geq 1$, be a node in $S_d$, and let $u_i \xrightarrow{p_i} v_1, \ldots, u_m \xrightarrow{p_m} v_m$, $m \geq 0$, be all edges in $S$ whose source is in this set. For any nonempty subset $J \subseteq \{1, 2, \ldots, m\}$, let $V = \{v_j \mid j \in J\}$. We introduce an edge $U \rightarrow V$ in $S_d$ labeled with the unary formula $\land_{j=1, m} q_j$, where $q_j = p_j$ when $j \in J$, and $q_j = \neg p_j$ otherwise. Note that at least one unary formula $p_j$ is not negated. Note also that $S_d$ is deterministic, because for any two outgoing edges $U \rightarrow V$ and $U \rightarrow V'$ there exists at least some unary formula $p_j$ which is negated on the first edge but unnegated in the second one, or vice versa. To show $S \preceq S_d$ we prove that the relation $u \preceq U \iff u \in U$ is a simulation. For $u \preceq U$, $u \xrightarrow{p} v$ an edge in $S$ and $a \in U$ s.t. $U \models p(a)$, using the notations above we define $J = \{j \mid U \models p_j(a)\}$ and $V = \{v_j \mid j \in J\}$. Finally we have to prove item (2). Let $\preceq$ be a simulation from $S$ to $S'$, with $S'$ deterministic. Define $\preceq'$ to be the following relation from the nodes of $S_d$ to those of $S'$: $U \preceq' u'$ iff $\forall u \in U, u \preceq u'$. For $U \preceq' u'$, edge $U \rightarrow V$ in $S_d$ labeled $\lor_j q_j$

\(^1\)The tree $\{a\}$ is deterministic and bisimilar to the tree $\{a, a\}$; but the latter is not deterministic.

\(^2\)The $v_1, \ldots, v_m$ are not necessarily distinct.
(notations as above), and \( a \in U \) s.t. \( U \models \exists j q_j(a) \), we use the fact that \( \leq \) is a simulation, so for each edge \( u_i \xrightarrow{p_i} v_j \), with \( j \in J \) there has to be some transition \( u' \xrightarrow{p'} v' \) with \( U \models p'(a) \), and with \( v_j \leq v' \). Here we use the fact that \( S' \) is deterministic, to conclude that \( p' \) and \( v' \) are the same, for all \( j \in J \). It follows that \( V \leq v' \).

In particular, the construction of \( S_d \) gives us a procedure for checking whether \( S \) is deterministic:

**Corollary 5.4** Given a schema \( S \), it is decidable whether \( S \) is deterministic or not.

**Proof:** \( S \) is deterministic iff \( S_d \leq S \): the latter relation is decidable. \( \square \)

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 \( \text{Int}, \text{String}, \text{Bool}, \text{Nat}, \text{isDept} \).

The graph schemas in Figure 4 have this property. Then we can prove:

**Proposition 5.5** 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. \( U \models \neg \exists x.(p(x) \land p'(x)) \)). 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 \texttt{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 \( [S] = \{ DB \mid DB \leq S \} \). We consider whether, given a schema \( S \) and an UnQL query \( Q \), we can describe the set \( \{ Q(DB) \mid DB \leq S \} \) by a schema \( S' \). This question is important for two reasons. First, we plan to use graph schemas in query optimization of UnQL. Since UnQL is compositional, when we optimize a composed query \( Q(DB) \overset{\text{def}}{=} Q_2(Q_1(DB)) \) whose input conforms to some graph schema, \( DB \leq S \), we first optimize \( Q_1 \) according to graph schema \( S \), then optimize \( Q_2 \) according to the graph schema of the set \( \{ Q_1(DB) \mid DB \leq S \} \), hence the need to compute the latter. Second, UnQL queries can be used to define views, like \( V \overset{\text{def}}{=} Q(DB) \). Given

\begin{align*}
Q_1(DB) & = \text{select} \ \{ x \Rightarrow \{ x \} \} \ \text{where} \ \backslash x \leftarrow DB \\
Q_2(DB) & = \text{select} \ \{ x \Rightarrow \{ x \} \} \ \text{where} \ \backslash x \leftarrow DB
\end{align*}

\( Q_1 \) 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., \( Q_1 \) 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 \( [S] = \{ DB \mid DB \leq S \} \). We consider whether, given a schema \( S \) and an UnQL query \( Q \), we can describe the set \( \{ Q(DB) \mid DB \leq S \} \) by a schema \( S' \). This question is important for two reasons. First, we plan to use graph schemas in query optimization of UnQL. Since UnQL is compositional, when we optimize a composed query \( Q(DB) \overset{\text{def}}{=} Q_2(Q_1(DB)) \) whose input conforms to some graph schema, \( DB \leq S \), we first optimize \( Q_1 \) according to graph schema \( S \), then optimize \( Q_2 \) according to the graph schema of the set \( \{ Q_1(DB) \mid DB \leq S \} \), hence the need to compute the latter. Second, UnQL queries can be used to define views, like \( V \overset{\text{def}}{=} Q(DB) \). 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 Equation 1 holds, but \( Q(S) \) may not necessarily get the tightest description of the set \( \{Q(DB) \mid DB \preceq S\} \). It follows however that \( Q(S) \) can be computed in PTIME, and that it satisfies Equation 1. But this can be trivially satisfied by taking \( Q(S) = S_\uparrow \) (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 \( D \triangleq \{Q(DB) \mid DB \preceq S\} \). Unfortunately, this does not hold. Worse, there are examples of simple queries \( Q \) and graph schema \( S \) for which \( \bigsqcup D \) does not exist, as illustrated in the following example.

**Example 6.1** 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 \( DB = \{2,4,5\} \) returns \( \{2 \Rightarrow 2\}, 4 \Rightarrow \{4\}, 5 \Rightarrow \{5\}\). When we apply our method to the schema \( S \) and query \( Q \), we obtain the graph schema \( S' = Q(S) = \{Nat \Rightarrow \{Nat\}\} \). But this is not \( \bigsqcup D \), and in fact one can show that \( \bigsqcup D \) does not exists. Instead of giving the formal proof, we explain what is going on. Consider the sequence of graph schemas \( S_0, S_1, S_2, \ldots \) where \( S_n = \{0 \Rightarrow \{0\}, 1 \Rightarrow \{1\}, \ldots, n-1 \Rightarrow \{n-1\}, p_n \Rightarrow \{Nat\}\} \), with \( p_n(x) = (x \neq 0 \land \ldots \land x \neq n-1 \land Nat(x)) \). In particular \( S' = S_0 \). Each of them is an upper bound of \( D \), and they form an infinite, strictly descending chain of graph schemas. Hence none of them is a least upper bound for \( D \).

The reason why graph schemas cannot describe all sets of the form \( \{Q(DB) \mid DB \preceq S\} \) is 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 \( n \geq 0 \) 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 \( n \) 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 \( u \rightarrow v \) 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 the following four conditions on extended graph schemas:

- For every edge \( u \rightarrow v \) entering some graph \( G_z \) (i.e. \( u \notin V_z \) and \( v \in V_z \)), \( v \) is one of the inputs of \( G_z \).
- Similarly, every edge \( u \rightarrow v \) leaving some graph \( G_z \) exits from an output node, \( u \in O_z \).
- Each formula labeling some edge in the scope of \( k \) variables \( z_i_1, \ldots, z_i_k \) may have \( k + 1 \) free variables: \( z_i_1, \ldots, z_i_k \) and a distinguished variable \( x \) (as before).
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 $S^\infty$, which we describe formally below. Intuitively, each graph $G_z$ is replicated once for each value $z \in \mathcal{U}$.

**Example 6.2** Figure 9 contains two examples of extended graph schemas, both with a single variable, $z$. In (a), $G_z = G$, $I_z$ consists of a single node (the root), and $O_z = \emptyset$. In its expansion, we replace $z$ with every value in the universe, $\mathcal{U}$, but for values outside $\text{Nat}$ we obtain predicates equivalent to false. This infinite expansion describes precisely the set $\{Q(DB) \mid DB \subseteq S\}$, where $Q$ and $S$ are from Example 6.1. In (b), $G_z$ is a proper subgraph of $G$, with each of $I_z$ and $O_z$ consisting of one node. The "expansion" depicted in (b) is incomplete: $\text{not}(0)$ should be further expanded with all atoms $a \in \mathcal{U}$, $a \neq 0$, and similarly for $\text{not}(1), \text{not}(2), \ldots$.

Formally, for a given extended schema $S = (V, E, v_0, G_{z_1}, \ldots, G_{z_n})$ over variables $z_1, \ldots, z_n$, we define $S^\infty$ to be the following infinite database. Its nodes are tuples $(u, a_1, \ldots, a_k)$, where $u \in V$, $k$ is the number of variables $z_1, \ldots, z_n$ in whose scope $u$ is, and $a_1, \ldots, a_k$ are elements of the universe $\mathcal{U}$. We call nodes of the form $(u, a_1, \ldots, a_k)$ copies of $u$. The root of $S^\infty$ is a fresh node $v_0$, and there are $\varepsilon$-edges from $v_0$ to all "copies" of the old root $v_0$ (see [BDHS96a] for a definition of $\varepsilon$ edges). For each edge $u \xrightarrow{a} v$ in $S$ we construct copy edges in $S^\infty$ as follows. Let $u$ be in the scope of $z_{i_1}, \ldots, z_{i_k}$. Due to the scoping rules for the graphs $G_z$, only one of the following three cases can happen:

- **Node $v$ is in the scope of the same variables.** Then we add and edge labeled $a$ between any two similar copies of $u$ and $v$, for which $a \in \mathcal{U}$ "satisfies" $p$: more precisely, we add an edge $(u, a_{i_1}, \ldots, a_{i_k}) \xrightarrow{a} (v, a_{i_1}, \ldots, a_{i_k})$ whenever $p(a_{i_1}, \ldots, a_{i_k}, a)$ is true.

- **Node $v$ is in the scope of strictly more variables, i.e. of $z_{i_1}, \ldots, z_{i_l}$, with $k < l$.** Then from each copy $(u, a_{i_1}, \ldots, a_{i_k})$ of $u$ we add "fan-out" edges to copies of $v$ extending that of $u$: more precisely, we add an edge $(u, a_{i_1}, \ldots, a_{i_k}) \xrightarrow{a} (v, a_{i_1}, \ldots, a_{i_l})$ whenever $p(a_{i_1}, \ldots, a_{i_k}, a)$ is true.

- **Node $v$ is in the scope of strictly less variables, i.e. of $z_{i_1}, \ldots, z_{i_l}$, with $k > l$.** Then into each copy $(v, a_{i_1}, \ldots, a_{i_k})$ of $v$ we add "fan-in" edges from copies of $u$ extending that of $v$: more precisely, we add an edge $(u, a_{i_1}, \ldots, a_{i_k}) \xrightarrow{a} (v, a_{i_1}, \ldots, a_{i_k})$ whenever $p(a_{i_1}, \ldots, a_{i_k}, a)$ is true.
Note that unlike graph schemas, in the case of extended graph schemas $S^\infty$ has infinitely many nodes.

Since extended graph schemas are nothing more than an elaborate way of specifying the infinite graph $S^\infty$, 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.

By adding extended graph schemas, we have enriched our set of schemas. As a consequence, if before $S$ was the least upper bound of some set of databases $\mathcal{D}$, $S = \bigsqcup \mathcal{D}$, this may no longer be the case once we introduce extended schemas. So we have to reconsider Theorem 4.4. In fact, now it fails: even when $S$ is a non-extended schema, it need not be the least upper bound of $[S]$, as illustrated in the following example.

**Example 6.3** Let $S = \{a \Rightarrow \{\text{Nat}\}\}$. Then $S^\infty = DB$ with $DB$ from Example 4.8, and $[S] = \text{appr} (DB)$. We show that $S \not\preceq [S]$. Take $S'$ to be the extended schema given by the graph $G = G_2 = \{u \xrightarrow{a} v \xrightarrow{\text{Nat}(z)} w\}$ with $I_2 = \{u\}$ and $O_2 = \emptyset$. Then $S'^\infty = DB'$ of Example 4.8, and $S'$ is an upperbound of $[S]$ but $S \not\preceq S'$. Intuitively, $S'$ is better than $S = \{a \Rightarrow \{\text{Nat}\}\}$ because it says that after each $a$-edge, at least one natural number is missing. This proves that $S \not\preceq [S]$. One wonders whether $S' = [S]$: this is also false, because by using two variables $z_1, z_2$ we can write an extended schema $S''$ saying say that at least two natural numbers are missing on each branch etc. In fact the set $[S]$ does not have a least upper bound in the preordered set of extended schemas.

The reason why Theorem 4.4 fails in the set of all extended graph schemas is because, for an extended schema $S$, $S^\infty$ is not generally label-finite. It still holds in the following weaker form: if $\mathcal{D} = [S]$ and $S'$ is an upper bound of $\mathcal{D}$, then if $S'^\infty$ is label finite, it follows that $S \preceq S'$. In particular, this holds for deterministic $S'$. Generalizing this, we 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 isempty (isempty is the only non-monotone operator in UnCAL, see [Sue96] for a more detailed discussion).

**Theorem 6.4** 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 $Q(S) \preceq S'$.

**Proof:** Following [BDHS96b], we define the *positive* fragment of UnCAL, denoted UnCAL+, to be the language described by the rules in Figure 10. We refer the reader to [BDHS96b] for a semantics of the language. It is “positive” in that the operator isempty is missing. It suffices to prove the statement of the theorem for queries in UnCAL+.

UnCAL+ manipulates graph data whose structure is more general than the rooted, labeled graphs defined in Section 2: namely they are graphs with $m$ distinguished inputs, $n$ distinguished outputs, and with $\varepsilon$-edges. As in [BDHS96b] we “distinguish” nodes by labeling them with markers: $\varepsilon$-edges are simply silent transitions. To describe such graphs in terms of schemas, we generalize our (extended) schemas to graphs with $m$ inputs, $n$ outputs and $\varepsilon$-edges too. All definitions and properties of graph schemas generalize mutatis mutandis to the new framework.

We will show how to construct $S' = Q(S)$ by induction on the structure of the query $Q$. More precisely, let $y_1, \ldots, y_m$ be the free label variables, and $t_1, \ldots, t_n$ be the free tree variables of $Q,$
A label variable \( y \) and a tree variable of type \( \text{Tree}_X \)

\[
\begin{align*}
& a \in U \\ & a : \text{Label} \\
& y \text{ a label variable} \\ & y : \text{Label} \\
& t \text{ a tree variable of type } \text{Tree}_X \\ & t : \text{Tree}_X \\
& {} : \text{Tree}_X \\
& X : \text{Tree}_X \\
& Y \in X \\
& l : \text{Label} \\& Q : \text{Tree}_X \\
& X = 0 \\
& l_1 : \text{Label} \\& l_2 : \text{Label} \\
& l_1 = l_2 : \text{Bool} \\
& l_1 : \text{Label} \\& \ldots \\& l_n : \text{Label} \\
& p(a_1, \ldots, a_n) : \text{Bool} \\
& b : \text{Bool} \\& Q_1 : \text{Tree}_X \\& Q_2 : \text{Tree}_X \\
& \text{if } b \text{ then } Q_1 \text{ else } Q_2 : \text{Tree}_X \\
& Q_1 : \text{Tree}_Y \\& \ldots \\& Q_m : \text{Tree}_Y \\
& (X_1 := Q_1, \ldots, X_m := Q_m) : \text{Tree}_Y^{(X_1, \ldots, X_m)} \\
& Q_1 : \text{Tree}_X \\& Q_2 : \text{Tree}_X \\
& Q_1 \cup Q_2 : \text{Tree}_X \\
& Q_1 \otimes Q_2 : \text{Tree}_Y \\
& y \text{ label variable} \\& t \text{ tree variable of type } \text{Tree}_Y \\
& Q_1 : \text{Tree}_X \\& Q_2 : \text{Tree}_X \\
& g_{ext} \lambda \text{Tree}_X(Y, t, Q_1)(Q_2) : \text{Tree}_Y \\
\end{align*}
\]

\( X \) denotes a finite set of markers. \( \text{Tree}_X \) denotes rooted graphs whose output markers are in the set \( X \). \( \text{Tree}_Y \) denotes graphs whose input markers are \( X \) and whose output markers are in \( Y \).

Figure 10: The rules for the positive fragment of UnCAL, denoted UnCAL*.

In notation \( Q(y_1, \ldots, y_m, t_1, \ldots, t_n) \), and let \( S_1, \ldots, S_n \) be \( n \) given extended schemas over variables \( z_1, \ldots, z_p \). Then we show how to construct a new extended schema \( S' \triangleq Q(y_1, \ldots, y_m, S_1, \ldots, S_n) \) over variables \( y_1, \ldots, y_m, z_1, \ldots, z_p \) such that for any labels \( a_1, \ldots, a_m \), then the following two conditions hold:

1. For any databases \( DB_1, \ldots, DB_n \) conforming to \( S_1, \ldots, S_n \) we have:

\[
Q(a_1, \ldots, a_m, DB_1, \ldots, DB_n) \preceq S'[a/y] 
\]

2. For any tree database \( TDB' \in \text{appr}(S'[a/y]) \) there exists \( n \) tree databases \( TDB_1, \ldots, TDB_n \) conforming to \( S_1, \ldots, S_n \) respectively, such that:

\[
TDB' \preceq Q[a/y, TDB'/t] 
\]

When \( Q \) is a query with a single free variable (the input database), then condition 2 implies that \( S' \) is an upper bound of \( \{Q(DB) \mid DB \preceq S\} \), while condition 3 implies (by Proposition 4.7) that \( S' \) will be below any other label-finite upper bound, in particular below any deterministic upper bound.

We now illustrate the more interesting cases for \( Q \).

**Tree variable** \( Q = t_i \). Take \( S' \triangleq S_i \).
Singleton construct $Q = \{ l \Rightarrow Q_1 \}$ First apply induction hypothesis to $Q_1$, and obtain schema $S'_1$. The label expression $l$ is either a constant $a$, or a variable $y$. In the first case return the schema $\{(x = a) \Rightarrow S'_1\}$. In the second case return $\{(x = y) \Rightarrow S'_1\}$.

Union $Q = Q_1 \cup Q_2$. Obtain $S'_1$ and $S'_2$ first. Then take $S' \overset{\text{def}}{=} S'_1 \cup S'_2$.

Conditional $Q = \text{if } b \text{ then } Q_1 \text{ else } Q_2$. Construct $S'_1, S'_2$ first. Then define $S'$ to have a new root and two $\varepsilon$-edges into $S'_1$ and $S'_2$ respectively, labeled with $b$ and $\neg b$ respectively. That is, $b$ and $\neg b$ are predicates with free variables $y_1, \ldots, y_m$ (not the distinguished variable $x$). “Labeling” an $\varepsilon$ edge with a predicate means that we enable it when the predicate is true, and disable it when it is false. In the case of $S'$, exactly one of the two $\varepsilon$ edges will be enabled, for each $m$-tuple $a_1, \ldots, a_m$. For example, to check Equation 3 above, let $a_1, \ldots, a_m$ be given, and $TDB'$ be some database in $\text{app} (S'[a/y])$. Then exactly one of $b[a/y]$ or $\neg b[a/y]$ must be true. In the first case we reason that $TDB'$ is also in $\text{app} (S'[a/y])$, hence there exists tree databases $TDB_1 \preceq S_1, \ldots, TDB_n \preceq S_n$ such that $TDB' \preceq Q_1(a_1, \ldots, a_m, TDB_1, \ldots, TDB_n)$, which also implies that $TDB' \preceq Q(a_1, \ldots, a_m, TDB_1, \ldots, TDB_n)$. The second case is similar.

Gext Assume, for sake of illustration, that the gext construct has a single marker, i.e. $Q = \text{gext}_{\{X\}} (\lambda(y,t).(X := Q_1))(Q_2)$. First we apply induction hypothesis to $Q_2$ and obtain some extended schema $S'_2$ with variables $y_1, \ldots, y_m, z_1, \ldots, z_p$. Our schema $S'$ will consists of the nodes of $S'_2$, with some subschemas replacing the edges in $S'_2$. The subschemas are obtained from the edges of $S'_2$ as follows. For each edge $u \overset{p}{\rightarrow} v$ in $S'_2$, we apply induction hypothesis to $Q_1$, in which we “bind” the variable $t$ to the schema $S'_{2v}$, that is having the same nodes and edges as $S'_2$, but with $v$ as its root. We obtain an extended schema $S'_1(u, p, v) = Q_1(y_1, \ldots, y_m, S_1, \ldots, S_n, S'_{2v})$, over the variables $y_1, \ldots, y_m, y, z'_1, \ldots, z'_q$. We assume $z_1, \ldots, z_p$ to be distinct from $z'_1, \ldots, z'_q$ (else we rename some of them). Then we join $u$ with the root $r$ of $S'_1(u, p, v)$ by a chain of two $\varepsilon$-edges: $u \overset{\varepsilon}{\rightarrow} w_{u,p,v} \overset{\varepsilon}{\rightarrow} r$ (here $w_{u,p,v}$ is a fresh node). We label the first edge with true, and the second with $p[y/x]$ (which is a predicate with variables $y_1, \ldots, y_m, y, z_1, \ldots, z_p$). We denote with $w_{u,p,v} \overset{\varepsilon}{\rightarrow} S'_1(u, p, v)$ the extended sub-schema consisting of the root $w_{u,p,v}$ and the $\varepsilon$-edge labeled $p[y/x]$ into the root of $S'_1(u, p, v)$: its variables are $y_1, \ldots, y_m, y, z_1, \ldots, z_p, z'_1, \ldots, z'_q$ and we define the scope of $y$ to be the entire graph $w_{u,p,v} \overset{\varepsilon}{\rightarrow} S'_1(u, p, v)$. In constructing $S'$ from $S'_2$, we replace the edge $u \overset{b}{\rightarrow} v$ with this subschema. We repeat this construct for every edge $u \overset{b}{\rightarrow} v$, and call $S'$ the resulting schema. Its variables are $y_1, \ldots, y_m, y, z_1, \ldots, z_p, z'_1, \ldots, z'_q$. (Technically, we have to rename all different occurrences of the variables $y$, $z'_1, \ldots, z'_q$ from different subschemas $S'_1(u, p, v)$; we omit some of the details.) We prove now Equation 3. Let $a_1, \ldots, a_m, TDB_1 \preceq S_1, \ldots, TDB_m \preceq S_m$ be given. Let $TDB' \in \text{app} (S'[a/y])$, which means $TDB' \preceq (S'[a/y])$. Since $TDB'$ is a tree, there exists a simulation which is a many-to-one mapping from the nodes of $TDB'$ to those of $(S'[a/y])^\infty$. We will decompose $TDB'$ into another tree $TDB'_1$, and a number of subtrees, which we generically denote $TDB'_2$. The nodes of $TDB'_2$ consists of those nodes in $TDB'$ which are mapped by $\preceq$ into nodes of $(S'[a/y])^\infty$ lying only within the scopes of $z_1, \ldots, z_p$, and not of $y, z'_1, \ldots, z'_q$. Consider such a node $u$ in $TDB'_2$. Any path in $TDB'$ from $u$ to some other node $v$ in $TDB_1$, without going through any other node in $TDB_2$, has, on all its edges, the same values of the variables $y, z_1, \ldots, z_p$. Moreover, all such paths from the same $u$ have the same values of $z_1, \ldots, z_p$, but they may differ on $y$. Then we construct edges in $TDB'_2$ as follows. We add a
new node $u_a$ for each value $y = a$ taken by $y$ on such a path, draw an edge labeled $a$ from $u$ to $u_a$, then draw $\varepsilon$-edges from $u_a$ to all nodes $v$ reachable in $TDB'$ from $u$ through a path on which $y = a$. Obviously $TDB'_2 \leq (S'_2[a/y])^\infty$. Next, we define the trees $TDB'_i$ to be small pieces which correspond to the same $y$. More precisely, for each node $u$ in $TDB'_2$, and for each value $a$ for which there exists some path in $TDB'$ from $u$ to another node $v$ in $TDB'_2$, we define the tree $TDB'_i(u,a)$ to consists of all nodes accessible from $u$ through paths having $y = a$. Each tree $TDB'_i(u,a)$ is mapped by $\varepsilon$-edges from $u_a$, to all nodes $v$ reachable in $TDB'$ from $u$ through a path on which $r_j = a$. Obviously $TDB'_1 \leq (SJ[\alpha/y], SJ[\alpha/y])^\infty$.'

Next, we define the trees $TDB'_2$ to be small pieces which correspond to the same $y$. More precisely, for each node $u$ in $TDB'_2$, and for each value $a$ for which there exists some path in $TDB'$ from $u$ to another node $v$ in $TDB'_2$, we define the tree $TDB'_2(u,a)$ to consists of all nodes accessible from $u$ through paths having $y = a$. Each tree $TDB'_2(u,a)$ is mapped by $\varepsilon$-edges from $u_a$, to all nodes $v$ reachable in $TDB'$ from $u$ through a path on which $r_j = a$. Obviously $TDB'_1 \leq (SJ[\alpha/y], SJ[\alpha/y])^\infty$.'

First, we notice that w.l.o.g. we may assume that $TDB'_1$ is the same for all trees $TDB'_1(u,a)$: else take their l.u.b., which is simply their union. But we may have different trees $TDB'_1(u,a)$ for different $TDB'_2(u,a)$'s, because they have to conform to different schemas $S'_2$. Hence we apply induction hypothesis to $Q_2$, and conclude that there exists $n$ tree databases $TDB'_1 \leq S_1, \ldots, TDB'_n \leq S_n$, $TDB(u,a) \leq S_{2'}$, such that $TDB'_1 \leq Q_1[a/y, a/y, TDB/i, TDB/t]$. First, we notice that w.l.o.g. we may assume that $TDB'_1$ is the same for all trees $TDB'_1(u,a)$: else take their l.u.b., which is simply their union. But we may have different trees $TDB(u,a)$ for different $TDB'_1(u,a)$'s, because they have to conform to different schemas $S'_2$ (the $v'$ differs). Then, let us denote $TDB''_2$ the tree obtained by expanding $TDB'_2$ with all trees $TDB(u,a)$, and by adding an $\varepsilon$ edge from each node $u_a$ to the root of $TDB(u,a)$. This definition of $TDB''_2$ implies that $TDB'_2 \leq gext(\lambda(y,t).Q_1[a/y, TDB/i], TDB''_2)$. Next we observe that $TDB''_2$ still conforms to $(S'_2[a/y])^\infty$, because the newly added trees $TDB$ conform to correct subschemas $S'_2$. Hence we apply induction hypothesis to $Q_2$, and conclude that there exists $n$ tree databases $TDB'_1 \leq S_1, \ldots, TDB'_n \leq S_n$ (we may assume w.l.o.g. that these are the same as before: else, take their g.l.b.) such that $TDB''_2 \leq Q_2[a/y, TDB/i, TDB/t]$. It follows that $TDB' \leq gext(\lambda(y,t).Q_1[a/y, TDB/i], TDB''_2)(Q_2[a/y, TDB/i])$.

\[ \square \]

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 [CAC94, 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:

\[ \text{select } \text{“Papers”.t where .t. “CS-Department”.t. “Papers”.t } \leftarrow \text{ DB} \]

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 $Q_{\text{opt}}$. 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


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


