Path Constraints on Semistructured and Structured Data

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

Wenfei Fan†
University of Pennsylvania
wfan@saul.cis.upenn.edu

Scott Weinstein‡
University of Pennsylvania
weinstein@linc.cis.upenn.edu

Abstract

We present a class of path constraints of interest in connection with both structured and semistructured databases, and investigate their associated implication problems. These path constraints are capable of expressing natural integrity constraints that are not only a fundamental part of the semantics of the data, but are also important in query optimization. We show that in semistructured databases, despite the simple syntax of the constraints, their associated implication problem is r.e. complete and finite implication problem is co-r.e. complete. However, we establish the decidability of the implication problems for several fragments of the path constraint language, and demonstrate that these fragments suffice to express important semantic information such as inverse relationships and local database constraints commonly found in object-oriented databases. We also show that in the presence of types, the analysis of path constraint implication becomes more delicate. We demonstrate some simple decidability results for two practical object-oriented data models.

1 Introduction

Path inclusion constraints have been studied in [3] in the context of semistructured data.

Consider the following object-oriented schema:

```
class student{
      Name:     string;
      Taking:   set(course);
}

class course{
      CName:     string;
      Enrolled:  set(student);
}

Students:  set(student);
Courses:   set(course);
```

*This work was partly supported by the Army Research Office (DAAH04-95-1-0169) and NSF Grant CCR92-16122.
†Supported by an IRCS graduate fellowship.
‡Supported by NSF Grant CCR-9403447.

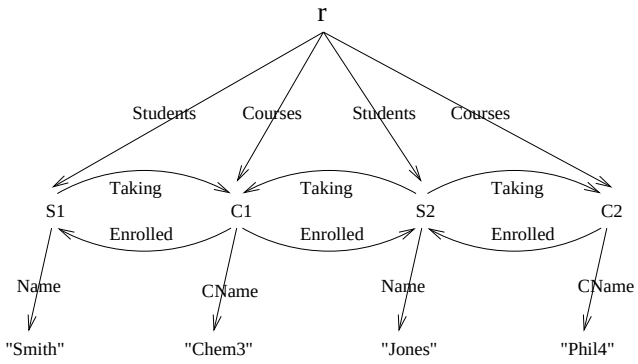

Figure 1: Representation of a school database

in which we assume that the declarations `Students` and `Courses` define (persistent) entry points into the database. As it stands, this declaration does not provide full information about the intended structure. Given such a database one would often expect the following informally stated constraints to hold:

(a) $\forall s \in Students \ \forall c \in s.Taking \ (c \in Courses)$
(b) $\forall c \in Courses \ \forall s \in c.Enrolled \ (s \in Students)$

That is, any course taken by a student must be a course that occurs in the database extent of courses, and any student enrolled in a course must be a student that similarly occurs in the database. We shall call such constraints *extent* constraints. It should be noted that there is a natural analogy between extent constraints and (unary) inclusion dependencies developed for relational databases.

We might also expect an *inverse relationship* to hold between `Taking` and `Enrolled`. Object-oriented databases differ in the ways they enable one to state and enforce extent constraints and inverse relationships. Compare, for example, O_2 [4] and ObjectStore [13]. The presence of such constraints is important both for database and for query optimization.

Let us develop a more formal notation for describing such constraints. To do this we borrow an idea that has been exploited in semistructured data models (e.g., [2, 7]) of regarding semistructured data as an edge-labeled graph. The database consists of two sets, and we express this by a root node r with edges emanating from it that are labeled either `Students` or `Courses`. These connect to nodes that respectively represent students and courses which have edges emanating from them that respectively describe the structure of students and courses. For example a student has a single `Name` edge connected to a string node, and multiple `Taking` edges connected to course nodes. See Figure 1 for an example of such a graph.

Using this representation of data we can examine certain kinds of constraints.

Extent Constraints. By taking edge labels as binary predicates, constraints of the form (a) and (b) above can be stated as:

$$\forall c\,(\exists s\,(Students(r,s) \wedge Taking(s,c)) \rightarrow Courses(r,c))$$
$$\forall s\,(\exists c\,(Courses(r,c) \wedge Enrolled(c,s)) \rightarrow Students(r,s))$$

These constraints are examples of "word constraints" studied in [3]; the implication problems for word constraints were shown to be decidable in semistructured databases there.

Inverse Constraints. These are common in object-oriented databases [11]. With respect to our student/course schema, the inverse between `Taking` and `Enrolled` is expressed as:

$$\forall s\,c\,(Students(r,s) \wedge Taking(s,c) \rightarrow Enrolled(c,s))$$
$$\forall c\,s\,(Courses(r,c) \wedge Enrolled(c,s) \rightarrow Taking(s,c))$$

Such constraints cannot be expressed as word constraints or even by the more general path constraints given in [3].

Local Database Constraints. In database integration it is sometimes desirable to make one database a component of another database, or to build a "database of databases". Suppose, for example, we wanted to bring together a number of student/course databases as described above. We might write something like:

```
class School-DB{
   DB-identifier:  string;
   Students:   set(student);   // as defined above
   Courses:    set(course);    // as defined above
}

Schools:  set(School-DB);
```

Now we may want certain constraints to hold on components of this database. For example, the "extent constraints" described above now hold on each member of the `Schools` set. Here we refer to a component database such as a member of the set `Schools` as a *local database* and its constraints as *local database constraints*. Extending our graph representation by adding `Schools` edges from the new root node to the roots of local databases, the local extent constraints are:

$$\forall d\,c\;\;(Schools(r,d) \wedge \exists s\,(Students(d,s) \wedge Taking(s,c)) \rightarrow Courses(d,c))$$
$$\forall d\;\;s\;\;(Schools(r,d) \wedge \exists c\,(Courses(d,c) \wedge Enrolled(c,s)) \rightarrow Students(d,s))$$

Again, these cannot be stated as word constraints.

These considerations give rise to the question whether there is a natural generalization of the constraints of [3] which will capture these slightly more complicated forms. Here we consider a class of path constraints of either the form

$$\forall x\,y\,(\alpha(r,x) \wedge \beta(x,y) \rightarrow \gamma(x,y)),$$

or the form

$$\forall x\,y\,(\alpha(r,x) \wedge \beta(x,y) \rightarrow \gamma(y,x)),$$

where $\alpha(x,y)$ ($\beta(x,y)$, $\gamma(x,y)$) represents a path from node x to node y.

This class of path constraints can be used to express all the constraints we have so far encountered. These path constraints are useful not only for optimizing queries, but also for describing structure, in the context of structured data or semistructured data. Surprisingly, the implication problem for this mild generalization of word constraints is undecidable in semistructured databases. However, certain restricted cases are decidable, and these cases are sufficient to express at least the constraints we have described above.

Another issue is the interaction between constraints and the type system. The type system or schema definition may also be viewed as imposing a constraint on the data. In general we can no longer expect results developed for semistructured data to hold when a type is imposed on the data. Indeed, the proof of the decidability of word constraint implication given in [3] breaks down in the presence of type constraints and again, only in restricted type systems do we have decidability results on word constraint implication.

The rest of the paper is organized as follows. Section 2 formally presents our path constraint language, P, and establishes the undecidability of its associated implication problems in the context of semistructured databases. Section 3 identifies several fragments of P, and shows that the implication and finite implication problems for each of these fragments are decidable in semistructured databases. Section 4 studies the interaction between type constraints and path constraints, and establishes some simple decidability results for two specific type systems: a "generic" object-oriented type system and a type system based on ACeDB [14], which is a database management system popular with biologists. Section 5 summarizes our results and identifies directions for further work.

2 Path constraints on untyped data

In this section, we investigate the path constraint language and its associated implication problems in the context of semistructured data, by which we mean data whose structure is not constrained by a schema. We first present an abstraction of semistructured databases, and define the path constraint language, P, in terms of first-order logic. We then show that, despite the simple syntax of the language P, its associated implication problem is r.e. complete and its finite implication problem is co-r.e. complete.

We assume the standard notations used in first-order logic.

2.1 Path constraint language P

Semistructured data is usually represented as an edge-labeled (rooted) directed graph, e.g., in UnQL [7] and in OEM [2]. See [1] for a survey of semistructured data models. Along the same lines, here we use an abstraction of semistructured databases as (finite) first-order logic structures of signature

$$\sigma = (r,\ E),$$

where r is a constant denoting the root and E is a finite set of binary relations denoting the edge labels.

A path, i.e., a sequence of labels, can be represented as a logic formula with two free variables. More specifically, a path is a formula $\alpha(x,y)$ of one of the following forms:

- $x = y$, denoted $\epsilon(x,y)$ and called an *empty path*;
- $K(x,y)$, where $K \in E$; or
- $\exists z(K(x,z) \wedge \beta(z,y))$, where $K \in E$ and $\beta(z,y)$ is a path.

Here the free variables x and y denote the tail and head nodes of the path, respectively. We write $\alpha(x, y)$ as α when the parameters x and y are clear from the context.

The path constraint language P is formalized as follows.

Definition 2.1: A *path constraint* φ is an expression of either the *forward* form

$$\forall x\, y\, (\alpha(r, x) \wedge \beta(x, y) \rightarrow \gamma(x, y)),$$

or the *backward* form

$$\forall x\, y\, (\alpha(r, x) \wedge \beta(x, y) \rightarrow \gamma(y, x)),$$

where α, β, γ are paths. The path α is called the *prefix* of φ. The paths α, β and γ are denoted $pf(\varphi)$, $lt(\varphi)$ and $rt(\varphi)$, respectively.

The set of all path constraints is denoted P. ∎

For example, all the path constraints presented in the last section are constraints in the set P.

Next, we identify several special subclasses of P.

We call a path constraint φ of P a *simple path constraint* if $pf(\varphi) = \epsilon$. That is, φ is of either the form

$$\forall y\, (\beta(r, y) \rightarrow \gamma(r, y)),$$

or the form

$$\forall y\, (\beta(r, y) \rightarrow \gamma(y, r)).$$

The set of all simple path constraints is denoted P_s.

A proper subclass of simple path constraints, called *word constraints* and denoted P_w, was introduced and investigated in [3]. A word constraint can be represented as

$$\forall y\, (\beta(r, y) \rightarrow \gamma(r, y)),$$

where β and γ are paths.

As observed by [3], every word constraint (in fact, every simple path constraint) can be expressed by a sentence in two-variable first-order logic (FO^2), the fragment of first-order logic consisting of all relational sentences with at most two distinct variables. Recently, [12] has shown that the satisfiability problem for FO^2 is NEXPTIME-complete by establishing that any satisfiable FO^2 sentence has a model of size exponential in the length of the sentence. The decidability of the implication and finite implication problems for word constraints follows immediately. In fact, [3] directly established (without reference to the embedding into FO^2) that the implication problems for word constraints are in PTIME.

In contrast to word constraints, many path constraints in P are not expressible in FO^2.

Example 2.1: Consider the structures G and G' given in Figure 2. It is easy to verify, using the 2-pebble Ehrenfeucht-Fraïssé style game [5], that G and G' are equivalent in FO^2. However, G and G' are distinguished by the path constraint

$$\varphi = \forall x\, y\, (K(r, x) \wedge K(x, y) \rightarrow \exists z(K(x, z) \wedge K(z, y))),$$

because $G \models \varphi$ but $G' \not\models \varphi$. This shows that φ is not expressible in FO^2. ∎

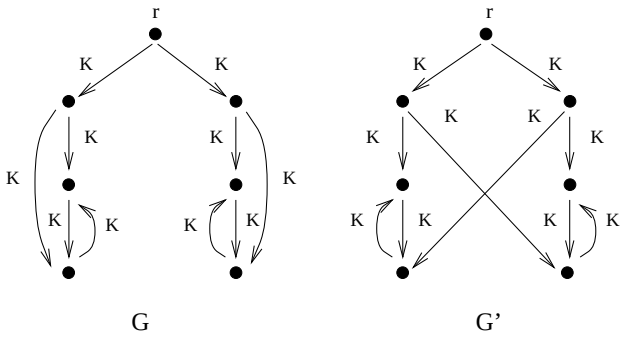

Figure 2: Structures distinguishable by P

2.2 Implication problems for P

The (finite) implication problem for P is the problem of determining, given any finite set $\Sigma \cup \{\varphi\}$ of sentences in P, whether all the (finite) σ-structures satisfying Σ are also models of φ.

Theorem 2.1: The implication problem for P is r.e. complete, and the finite implication problem for P is co-r.e. complete. ∎

In fact, this result also holds for two proper subclasses of P. One of the subclasses, P_f, is the set of all the constraints of the forward form in P. The other, P_+, is the set

$$\{\varphi \mid \varphi \in P,\ lt(\varphi) \neq \epsilon,\ rt(\varphi) \neq \epsilon\}.$$

For P_+ and P_f we have the following theorems, from which Theorem 2.1 follows immediately.

Theorem 2.2: The implication problem for P_+ is r.e. complete, and the finite implication problem for P_+ is co-r.e. complete. ∎

Theorem 2.3: The implication problem for P_f is r.e. complete, and the finite implication problem for P_f is co-r.e. complete. ∎

Proof sketch: Theorem 2.2 is proved by reduction from the halting problem for two-register machines [6]. More specifically, the idea of the proof is to show that the set

$$S(P_+) = \{\bigwedge \Sigma \wedge \neg\varphi \mid \varphi \in P_+,\ \Sigma \subset P_+,\ \Sigma \text{ is finite}\}$$

is a *conservative reduction class* [6]. That is, we show that there is a recursive function f from the set of all first-order sentences, FO, to $S(P_+)$, such that for all $\psi \in FO$,

- ψ is satisfiable, i.e., ψ has a model, iff $f(\psi)$ is satisfiable; and
- ψ is finitely satisfiable, i.e., ψ has a finite model, iff $f(\psi)$ is finitely satisfiable.

To show that $S(P_+)$ is a conservative reduction class, we prove the existence of a *semi-conservative reduction* from FO to $S(P_+)$. That is, there is a recursive function $h : FO \rightarrow S(P_+)$, such that for all $\psi \in FO$,

- if ψ is not satisfiable, then $h(\psi)$ is not satisfiable; and

- if ψ is finitely satisfiable, then $h(\psi)$ is finitely satisfiable.

This suffices, since [6] establishes that if there is a semi-conservative reduction from FO to a recursive subclass of FO, then the subclass is a conservative reduction class.

To establish the existence of a semi-conservative reduction from FO to $S(P_+)$, we encode the following two-register machine M by a sentence of $S(P_+)$. The machine M has two halting states: state 1 and 2. Given any sentence ψ in FO, M halts at state 1 if and only if ψ is not satisfiable, and halts at state 2 if and only if ψ is finitely satisfiable. The existence of such a two-register machine was established in [15] (see also [6]). See [8] for the detailed proof of Theorem 2.2. ∎

The proof of Theorem 2.3 is similar and can also be found in [8].

3 Restricted path constraint implication in untyped data

The undecidability results given in the last section suggest that we search for fragments of P whose implication problems are decidable, and yet retain sufficient expressive power of the full language. In this section, we identify several fragments of P which share the following properties. First, they each properly contain the set of word constraints. Second, each of them is not included in two-variable first-order logic. Third, they allow the formulation of many semantic relations which are of interest from the point of view of database theory, such as extent constraints, inverse relationships and local database constraints. And finally, they each possess decidable implication problems in the context of semistructured databases.

Before we present these fragments, we first define some basic notations.

The *concatenation* of paths $\alpha(x,z)$ and $\beta(z,y)$, denoted $\alpha(x,z)\cdot\beta(z,y)$ or simply $\alpha\cdot\beta$, is the path

- $\beta(x,y)$, if $\alpha=\epsilon$;
- $\exists\, z\,(K(x,z)\wedge\beta(z,y))$, if $\alpha=K$ for some $K\in E$;
- $\exists\, u\,(K(x,u)\wedge(\alpha'(u,z)\cdot\beta(z,y)))$, if $\alpha(x,z)$ is of the form $\exists\, u\,(K(x,u)\wedge\alpha'(u,z))$, where $K\in E$ and α' is a path.

The *length* of path α, $|\alpha|$, is defined by:

$$|\alpha| \;=\; \begin{cases} 0 & \text{if } \alpha=\epsilon \\ 1 & \text{if } \alpha=K \\ 1+|\beta| & \text{if } \alpha=K\cdot\beta \end{cases}$$

Next, we introduce several fragments of P, demonstrate their expressive power, and establish the decidability of their associated implication problems. We also present a mild generalization of P, P^c, and show that the decidability results for the fragments of P investigated in this section also hold for the analogous fragments of P^c.

3.1 Prefix restricted implication

The implication problems for simple path constraints, which are known to be decidable, can be viewed as a restricted form of the implication problems for P. More specifically, the implication problems for P_s are the implication problems for P under the following restriction: for any given finite subset of P in the implication problems, the prefix of each constraint in the subset is the empty path.

By replacing this prefix restriction with a weaker one, we define the prefix restricted implication problems for P as follows.

Definition 3.1: A finite subset of P is called a *prefix restricted subset* of P if the prefixes of all the constraints in the set have the same length.

The *prefix restricted (finite) implication problem for P* is the problem of determining, given any prefix restricted subset $\Sigma\cup\{\varphi\}$ of P, whether all the (finite) models of Σ are also models of φ. ∎

Obviously, the implication problems for word constraints are special cases of the prefix restricted implication problems for P. Moreover, in contrast to word constraint implication, prefix restricted implications cannot be stated in two-variable first-order logic. A convenient argument for this is that $\{\varphi\}$, where φ is the constraint given in Example 2.1, is a prefix restricted subset of P.

Many cases of integrity constraint implication commonly found in databases are examples of the prefix restricted implication problem for P. Among these are some implications for inverse constraints and local database constraints. As an example, consider the set consisting of the two local inverse constraints on School-DB databases given in Section 1:

$$\begin{aligned}&\forall\, s\; c\;(\exists\, d\;(Schools(r,d)\wedge Students(d,s))\wedge Taking(s,c)\\ &\qquad\rightarrow Enrolled(c,s))\\ &\forall\, c\; s\;(\exists\, d\;(Schools(r,d)\wedge Courses(d,c))\wedge Enrolled(c,s)\\ &\qquad\rightarrow Taking(s,c))\end{aligned}$$

and the constraint

$$\begin{aligned}&\forall\, s_1\; s_2\;(\exists\, d\;(Schools(r,d)\wedge Students(d,s_1))\wedge\epsilon(s_1,s_2)\\ &\qquad\rightarrow\exists\, c\,(Taking(s_1,c)\wedge Enrolled(c,s_2))).\end{aligned}$$

This set is a prefix restricted subset of P.

Another example of prefix restricted implication is the implication of the constraint

$$\begin{aligned}&\forall\, x\; y\;(cities(r,x)\wedge\exists\, z\;(connect(x,z)\wedge connect(z,y))\\ &\qquad\rightarrow connect(y,x))\end{aligned}$$

from the constraints:

$$\begin{aligned}&\forall\, x\; y\;(cities(r,x)\wedge\exists\, z\;(connect(x,z)\wedge connect(z,y))\\ &\qquad\rightarrow connect(x,y))\\ &\forall\, x\; y\;(cities(r,x)\wedge connect(x,y)\rightarrow connect(y,x))\end{aligned}$$

Theorem 3.1: The prefix restricted implication and finite implication problems for P are decidable in semistructured databases. ∎

Proof sketch: The idea of the proof is to establish the *small model property* [6] for the satisfiability corresponding to the prefix restricted implications. More specifically, let S_p be

$$\{\bigwedge\Sigma\wedge\neg\varphi \mid \Sigma\cup\{\varphi\} \text{ is a prefix restricted subset of } P\}.$$

We show that there is a recursive function s such that for each $\psi\in S_p$, if ψ is satisfiable, then ψ has a model of size at most $s(|\psi|)$, where $|\psi|$ stands for the length of ψ.

To establish the small model property for S_p, we use a path label criterion to characterize whether a σ-structure satisfies a sentence of S_p. More specifically, given a model G of a satisfiable sentence ψ of S_p, we label each node of G with paths in ψ. The path label of G, $LB(G, \psi)$, is the collection of the labels of all the nodes in G. This path label has the following properties:

- for any structure H, if $LB(H, \psi) = LB(G, \psi)$, then $H \models \psi$ iff $G \models \psi$; and
- there is a structure H of size at most $2^{2^{2|\psi|}}$, such that $LB(H, \psi) = LB(G, \psi)$.

To establish the existence of the "small" model H, the restriction on prefixes described in Definition 3.1 is used.

See [9] for the detailed proof. ∎

3.2 Sub-language P_β

Some cases of path constraint implication are not examples of the prefix restricted implication. For instance, the set consisting of the two extent constraints and the two inverse constraints on student/course databases given in Section 1 is not a prefix restricted subset of P.

The constraints in the last example, however, are in the sub-language P_β defined below.

Definition 3.2: A *β-restricted path constraint* φ is a constraint of P with $|lt(\varphi)| \leq 1$. That is, either $lt(\varphi) = \epsilon$, or $lt(\varphi) = K$ for some $K \in E$.

The set of all simple path constraints and all β-restricted path constraints is denoted by P_β. ∎

Note that the class of word constraints is a proper subset of P_β. In addition, not all constraints in P_β are expressible in two-variable first-order logic. Indeed, the constraint φ given in Example 2.1 is in P_β, but is not in FO^2.

Theorem 3.2: The implication and finite implication problems for P_β are decidable in semistructured databases. ∎

Proof sketch: In the same way as in the proof of Theorem 3.1, we establish the small model property for satisfiability of the following set of sentences:

$$S(P_\beta) = \{\bigwedge \Sigma \wedge \neg\varphi \mid \varphi \in P_\beta,\ \Sigma \subset P_\beta,\ \Sigma \text{ is finite}\}.$$

To do so, we use a path labeling mechanism similar to the one employed in the proof of Theorem 3.1. Given a model G of a satisfiable sentence ψ in $S(P_\beta)$, we apply the mechanism to label each node of G with paths in ψ, and therefore, obtain the label of G with respect to ψ. In addition, we show that there is a model H of ψ such that each node of H has a unique path label. The size of H is, therefore, bounded by the cardinality of the label of H with respect to ψ, which is at most $2^{2|\psi|^2}$. Thus the small model property is established.

The detailed proof can be found in [9]. ∎

Example 3.1: The decidability of the implication and finite implication problems for P_β is useful for, among other things, query optimization. To illustrate this, consider the student/course databases given in Section 1 as semistructured databases, i.e., in the absence of schema. Suppose we want to find the names of all the courses enrolled by students who are taking course "Chem3". In the syntax of Lorel [2], which is a language for querying semistructured data, this query can be expressed as Q_1:

```
Q1:   select C.CName
      from   r.Courses C,
             C.Enrolled.Taking.CName N
      where  N = "Chem3"
```

Given the two extent constraints and the two inverse constraints described in Section 1, it can be shown that Q_1 is equivalent to Q_2 expressed below:

```
Q2:   select N
      from   r.Courses C',
             C'.Enrolled.Taking.CName N
      where  C'.CName = "Chem3"
```

To show this, the following constraints are also used:

$$\forall s\, c\, (\exists\, c_1\ (Courses(r, c_1) \wedge Enrolled(c_1, s)) \wedge Taking(s, c) \rightarrow Enrolled(c, s))$$
$$\forall c\, (\exists\, c_1\ (Courses(r, c_1) \wedge \exists\, s\ (Enrolled(c_1, s) \wedge Taking(s, c))) \rightarrow Courses(r, c))$$

There are several things to note about these two constraints. First, they are in the language P_β. Second, they are implied by the extent and inverse constraints given in Section 1, which, as mentioned earlier, are in P_β themselves. Finally, the decidability of the implication problems for P_β forms the basis on which we can determine this implication.

It should also be noted that Q_2 is in most cases more efficient than Q_1. Indeed, Q_2 complies with the familiar optimization principle originating in relational database theory: performing selections as early as possible. ∎

3.3 Extended implications for P_β

Consider the set consisting of the local extent constraints and the local inverse constraints of School-DB databases. This set is neither a prefix restricted subset of P nor a subset of P_β. However, the constraints in this set share the following property: they all are constraints of student/course databases augmented with a common prefix `Schools`. In general, when represented in a global environment, path constraints of a local database are augmented with a common prefix.

This example motivates the following extension of P_β.

Definition 3.3: Let α be a path and φ a constraint in P_β. The *extension of φ with prefix α*, denoted $\delta(\varphi, \alpha)$, is the constraint defined either by

$$\forall x\, y\, (\alpha \cdot pf(\varphi)(r, x) \wedge lt(\varphi)(x, y) \rightarrow rt(\varphi)(x, y))$$

when φ is of the forward form, or by

$$\forall x\, y\, (\alpha \cdot pf(\varphi)(r, x) \wedge lt(\varphi)(x, y) \rightarrow rt(\varphi)(y, x))$$

when φ is of the backward form, where $\cdot$ is the path concatenation operator, and pf, lt and rt are defined in Definition 2.1.

Let α be a path and Σ a finite subset of P_β. The *extension of Σ with prefix α* is the subset of P defined by

$$\{\delta(\varphi, \alpha) \mid \varphi \in \Sigma\}.$$

Such a set is called a *prefix extended subset of* P_β.

The *extended (finite) implication problem for* P_β is the problem of determining, given any prefix extended subset $\Sigma \cup \{\varphi\}$ of P_β, whether all the (finite) models of Σ are also models of φ. ∎

For instance, the set described in the last example is a prefix extended subset of P_β.

Note that the (finite) implication problem for P_β is a special case of the extended (finite) implication problem for P_β. As an immediate result, the implications of word constraints are special cases of the extended implications of P_β. Moreover, extended implications of P_β cannot be stated in two-variable first-order logic.

Theorem 3.3: The extended implication and finite implication problems for P_β are decidable in semistructured databases. ∎

Proof sketch: We prove the theorem by reduction to the implication problems for P_β, whose decidability is established by Theorem 3.2.

Let $Paths$ denote the set of all paths, and $S_e(P_\beta)$ be the set

$$\{\bigwedge \Sigma \wedge \neg\varphi \mid \Sigma \cup \{\varphi\} \text{ is a prefix extended subset of } P_\beta\}.$$

Recall the set $S(P_\beta)$ defined in the proof of Theorem 3.2. We define a surjective mapping

$$f : S(P_\beta) \times Paths \to S_e(P_\beta),$$

such that $f(\bigwedge \Theta \wedge \neg\theta, \alpha) = \bigwedge \Sigma \wedge \neg\varphi$ iff $\Sigma \cup \{\varphi\}$ is the extension of $\Theta \cup \{\theta\}$ with prefix α. We then show that for each $\psi \in S(P_\beta)$ and path α,

- ψ is satisfiable iff $f(\psi, \alpha)$ is satisfiable; and
- ψ is finitely satisfiable iff $f(\psi, \alpha)$ is finitely satisfiable. In addition, if ψ has a finite model of size N then $f(\psi, \alpha)$ has a model of size $N + |\alpha|$.

Since f is surjective, by the proof of Theorem 3.2 and the argument above, the satisfiability of $S_e(P_\beta)$ has the small model property. In particular, for each $\phi \in S_e(P_\beta)$, if ϕ is satisfiable, then it has a model of size at most $2^{2^{|\phi|^2}}$.

See [9] for the detailed proof. ∎

3.4 Conjunctive path constraints

We next show that the decidability results given above also hold for an extension of path constraints. This extension is defined as follows.

Definition 3.4: A *conjunctive path constraint* ϕ is an expression of either the *forward* form

$$\forall x\, y\, (\bigwedge_{\alpha \in A} \alpha(r, x) \wedge \bigwedge_{\beta \in B} \beta(x, y) \to \gamma(x, y)),$$

or the *backward* form

$$\forall x\, y\, (\bigwedge_{\alpha \in A} \alpha(r, x) \wedge \bigwedge_{\beta \in B} \beta(x, y) \to \gamma(y, x)),$$

where A, B are non-empty finite sets of paths, and are denoted $pf(\phi)$ and $lt(\phi)$, respectively. The path γ is denoted by $rt(\phi)$.

The set of all conjunctive path constraints is denoted by P^c. ∎

Conjunctive path constraints are useful for, among other things, describing structure of semistructured data. To illustrate this, consider the following conjunctive path constraints, which, in the context of structured databases, would be inclusion constraints on database extents:

$$\forall x\, y\, (dept(r, x) \wedge ta(x, y) \to student(x, y))$$
$$\forall x\, y\, (dept(r, x) \wedge ta(x, y) \to employee(x, y))$$
$$\forall x\, y\, (dept(r, x) \wedge (student(x, y) \wedge employee(x, y)) \to ta(x, y))$$

Abusing object-oriented database terms, these constraints indicate that

- TA of a department is a "subclass" of both Student and Employee of the department; and
- the "extent" of TA is the intersection of the "extents" of Student and Employee.

Below we define fragments of P^c analogous to the fragments of P discussed above.

Definition 3.5: A finite subset Σ of P^c is called a *prefix restricted subset* of P iff for all ϕ, ψ in Σ, all the paths in $pf(\phi) \cup pf(\psi)$ have the same length.

The *prefix restricted (finite) implication problem for* P^c is the problem of determining, given any prefix restricted subset $\Sigma \cup \{\phi\}$ of P^c, whether all the (finite) models of Σ are also models of ϕ. ∎

Definition 3.6: A *simple conjunctive path constraint* ϕ is a constraint of P^c with $pf(\phi) = \{\epsilon\}$.

A *β-restricted conjunctive path constraint* ϕ is a constraint of P^c such that for each $\beta \in lt(\phi)$, $|\beta| \leq 1$.

The set of all simple conjunctive path constraints and all β-restricted conjunctive path constraints is denoted by P^c_β. ∎

Definition 3.7: Let ρ be a path and ϕ a constraint in P^c_β. The *extension of ϕ with prefix ρ*, denoted $\delta(\phi, \rho)$, is the constraint in P^c defined either by

$$\forall x\, y\, (\bigwedge_{\alpha \in pf(\phi)} \rho \cdot \alpha(r, x) \wedge \bigwedge_{\beta \in lt(\phi)} \beta(x, y) \to rt(\phi)(x, y))$$

when ϕ is of the forward form, or by

$$\forall x\, y\, (\bigwedge_{\alpha \in pf(\phi)} \rho \cdot \alpha(r, x) \wedge \bigwedge_{\beta \in lt(\phi)} \beta(x, y) \to rt(\phi)(y, x))$$

when ϕ is of the backward form.

Let ρ be a path and Σ a finite subset of P^c_β. The *extension of Σ with prefix ρ* is the subset of P^c defined by

$$\{\delta(\phi, \rho) \mid \phi \in \Sigma\}.$$

Such a set is called a *prefix extended subset of* P^c_β.

The *extended (finite) implication problem for* P^c_β is the problem of determining, given any prefix extended subset $\Sigma \cup$

$\{\phi\}$ of P_β^c, whether all the (finite) models of Σ are also models of ϕ. ∎

The following decidability results can be verified analogously to Theorem 3.1, 3.2 and 3.3, respectively.

Theorem 3.4: The prefix restricted implication and finite implication problems for P^c are decidable in semistructured databases. ∎

Theorem 3.5: The implication and finite implication problems for P_β^c are decidable in semistructured databases. ∎

Theorem 3.6: The extended implication and finite implication problems for P_β^c are decidable in semistructured databases. ∎

4 Path constraints on typed data

In this section, we investigate path constraints on structured data, by which we mean data constrained by a schema. We first show that there is interaction between path constraints and type constraints. In other words, results on path constraint implication in the context of semistructured databases may no longer hold in the typed context. We then investigate the class of word constraints for databases of two particular models. One of the models is a strictly typed object-oriented data model. The other is an object-oriented model based on ACeDB [14] which, while it is often considered a semistructured model [7], has in fact a separate type system that allows more flexibility than object-oriented types. We present an abstraction of the databases in these models in terms of first-order logic, and establish the decidability of word constraint implication in these models.

4.1 Impact of type constraints

In structured databases, path constraint implication is restricted by a schema. More specifically, the implication problem for path constraints over a schema Δ is the problem of determining, given a finite set $\Sigma \cup \{\varphi\}$ of path constraints, whether all the database instances of Δ that satisfy Σ are also models of φ. Here an instance of the schema Δ has a particular structure specified by Δ. In other words, an instance of Δ must satisfy certain type constraints imposed by Δ. In contrast, a semistructured database is free of type constraints.

Here we address the question whether there is interaction between type constraints and path constraints. We show that some results on path constraint implication in semistructured databases no longer hold in the presence of types. For example, consider the implication problems for the path constraint language P. In the context of semistructured databases, as established by Theorem 2.1, the implication problems are undecidable. In the typed context, however, the implication problem for P over a schema is decidable as long as the schema does not contain recursive types, i.e., self-referential data structures. This is because in any instance of such a schema, there are only finitely many navigation paths. In other words, the language P over the schema has only finitely many sentences up to equivalence, and therefore, its associated implication problem is decidable.

As another example to illustrate the impact of type constraints, consider word constraint implication. A proof of the decidability of word constraint implication in semistructured databases was presented in [3]. However, we will show that the proof breaks down in an object-oriented data model.

Because of the interaction between type constraints and path constraints, there is need to investigate path constraint implication in the typed context. Below we focus on the class of word constraints, which is properly contained in every fragment of the path constraint language studied in the last section.

4.2 An object-oriented model

In this section, we investigate word constraint implication in an object-oriented data model.

4.2.1 The data model

We first present the data model.

Assume a fixed countable set of labels, $\mathcal{L}$, and a fixed finite set of *base types*, $\mathcal{B}$. Let $\mathcal{C}$ be some finite set of *classes*. The set of *Types over* $\mathcal{C}$, $Types^{\mathcal{C}}$, is defined by the syntax:

$$\begin{array}{rcl} t & ::= & b \mid C \\ \tau & ::= & t \mid \{t\} \mid [l_1 : t_1, \ldots, l_n : t_n] \end{array}$$

where $b \in \mathcal{B}$, $C \in \mathcal{C}$, and $l_i \in \mathcal{L}$. The notations $\{t\}$ and $[l_1 : t_1, \ldots, l_n : t_n]$ represent *set type* and *record type*, respectively. We reserve τ to range over $Types^{\mathcal{C}}$.

A *schema* is a triple $\Delta = (\mathcal{C}, \nu, DBtype)$, where $\mathcal{C}$ is a finite set of classes, $DBtype \in Types^{\mathcal{C}} \setminus (\mathcal{B} \cup \mathcal{C})$, and ν is a mapping: $\mathcal{C} \to Types^{\mathcal{C}}$ such that for each $C \in \mathcal{C}$, $\nu(C) \notin \mathcal{B} \cup \mathcal{C}$.

Example 4.1: An example of schema is $(\mathcal{C}, \nu, DBtype)$, where $\mathcal{C}$ consists of a single class $Person$, ν maps $Person$ to a record type $[name : string, spouse : Person]$, and $DBtype$ is $\{Person\}$. ∎

A *database instance* of schema $(\mathcal{C}, \nu, DBtype)$ is a triple $I = (\pi, \mu, d)$, where

(1) π is an *oid assignment* that maps each $C \in \mathcal{C}$ to a finite set of oids, $\pi(C)$, such that for all $C, C' \in \mathcal{C}$,

$$\pi(C) \cap \pi(C') = \emptyset \text{ if } C \neq C';$$

(2) for each $C \in \mathcal{C}$, μ maps each oid in $\pi(C)$ to a value in $[\![\nu(C)]\!]_\pi$, where

$$\begin{array}{rcl} [\![b]\!]_\pi & = & D_b, \\ [\![C]\!]_\pi & = & \pi(C), \\ [\![\{\tau\}]\!]_\pi & = & \{V \mid V \subseteq [\![\tau]\!]_\pi, V \text{ is finite}\}, \\ [\![[l_1 : \tau_1, ..., l_n : \tau_n]]\!]_\pi & = & \{[l_1 : v_1, ..., l_n : v_n] \mid \\ & & \qquad v_i \in [\![\tau_i]\!]_\pi, i \in [1, n]\}; \end{array}$$

here D_b denotes the domain of base type b;

(3) d is a value in $[\![DBtype]\!]_\pi$, which represents a (persistent) entry point into the database.

We denote the set of all database instances of schema Δ by $\mathcal{I}(\Delta)$.

4.2.2 Abstraction of databases

We next present an abstraction of databases in the object-oriented model. Since structured data can be viewed as

semistructured data further constrained by a schema, along the same lines of the abstraction of semistructured databases given in Section 2, we represent a structured database as a first-order logic structure satisfying certain type constraints determined by its schema. Such a structure can be depicted as an edge-labeled rooted directed graph, which has particular structural properties determined by the schema.

Below we define the first-order signature determined by a schema. Two components of the signature are described as follows.

Let $\Delta = (\mathcal{C}, \nu, DBtype)$ be a schema. We define *the set of binary relations* and *the set of types determined by* Δ, denoted $E(\Delta)$ and $T(\Delta)$, respectively, as the smallest sets having the following properties:

- $DBtype \in T(\Delta)$ and $\mathcal{C} \subseteq T(\Delta)$;
- if $DBtype = \{t\}$ (or for some $C \in \mathcal{C}$, $\nu(C) = \{t\}$), then t is in $T(\Delta)$ and $*$ is in $E(\Delta)$;
- if $DBtype = [l_1 : t_1, \ldots, l_n : t_n]$ (or for some $C \in \mathcal{C}$, $\nu(C) = [l_1 : t_1, \ldots, l_n : t_n]$), then for $i \in [1, n]$, t_i is in $T(\Delta)$ and l_i is in $E(\Delta)$.

Note here, for ease of presenting type constraints below, we use the distinguished binary relation $*$ to denote the set membership relation. This differs slightly from the presentation in Section 1.

The *signature determined by schema* Δ, $\sigma(\Delta)$, is a triple

$$(r, E(\Delta), R(\Delta)),$$

where r is a constant (denoting the root), $E(\Delta)$ is the finite set of binary relations (denoting the labels) defined above, and $R(\Delta)$ is the finite set of unary relations (denoting the sorts) defined by $\{R_\tau \mid \tau \in T(\Delta)\}$.

For example, the signature determined by the schema given in Example 4.1 is (r, E, R), where

- r is a constant, which in each instance (π, μ, d) of the schema intends to name d;
- $E = \{*, name, spouse\}$; and
- $R = \{R_{DBtype}, R_{Person}, R_{string}\}$.

The type constraints determined by a schema can be formulated as sentences in two-variable logic with counting [6], the extension of FO^2 with counting quantifiers. In particular, below we use the counting quantifier $\exists!$, whose semantics is described as follows: structure G satisfies $\exists! x\, \psi(x)$ if and only if there exists a unique element a of G such that $G \models \psi(a)$.

We present type constraints as follows. For each τ in $T(\Delta)$, the *constraint imposed by* τ is the sentence ϕ_τ defined as follows:

(1) if $\tau = b$, then ϕ_τ is

$$\forall x\, (R_\tau(x) \rightarrow \forall y\, (\bigwedge_{l \in E(\Delta)} \neg l(x, y)));$$

(2) if for some $C \in \mathcal{C}$, $\tau = C$ and $\nu(C) = \{t\}$ (or $\tau = DBtype = \{t\}$), then ϕ_τ is

$$\forall x\, (R_\tau(x) \rightarrow \forall y\, (\bigwedge_{l \in E(\Delta) \setminus \{*\}} \neg l(x, y)) \wedge \forall y\, (*(x, y) \rightarrow R_t(y)));$$

(3) if for some $C \in \mathcal{C}$, $\tau = C$ and $\nu(C) = [l_1 : t_1, \ldots, l_n : t_n]$ (or $\tau = DBtype = [l_1 : t_1, \ldots, l_n : t_n]$), then ϕ_τ is

$$\forall x\, (R_\tau(x) \rightarrow \forall y\, (\bigwedge_{l \in E(\Delta) \setminus \{l_1, \ldots, l_n\}} \neg l(x, y)) \wedge \bigwedge_{i \in [1,n]} (\exists!\, y\, l_i(x, y) \wedge \forall y\, (l_i(x, y) \rightarrow R_{t_i}(y)))).$$

The *type constraint imposed by schema* Δ is the sentence

$$\Phi(\Delta) = R_{DBtype}(r) \wedge \bigwedge_{\tau \in T(\Delta)} \phi_\tau \wedge \forall x\, (\bigvee_{\tau \in T(\Delta)} R_\tau(x) \wedge \bigwedge_{\tau \in T(\Delta)} (R_\tau(x) \rightarrow \bigwedge_{\tau' \in T(\Delta) \setminus \{\tau\}} \neg R_{\tau'}(x))).$$

Accordingly, we present an abstraction of databases in the object-oriented model as follows. Its justification will be given later in the paper.

An *abstract database of a schema* Δ is a finite structure G of the signature $\sigma(\Delta)$ such that $G \models \Phi(\Delta)$. We denote the set of all abstract databases of a schema Δ by $\mathcal{U}_f(\Delta)$.

We use $\mathcal{U}(\Delta)$ to denote the set of all the structures of signature $\sigma(\Delta)$ satisfying the following conditions: For each $G \in \mathcal{U}(\Delta)$, $G \models \Phi(\Delta)$; and for each set type $\tau \in T(\Delta)$ and each $o \in R_\tau^G$, there are only finitely many o' in G such that $G \models *(o, o')$. Here R_τ^G denotes the unary relation R_τ in G.

4.2.3 Word constraints

We next present word constraints in the presence of types. To do so, we first define paths and types of paths over a schema.

Given schema $\Delta = (\mathcal{C}, \nu, DBtype)$, the set of *paths over* Δ, $Paths(\Delta)$, and *the type of path* α *in* $Paths(\Delta)$, $type(\alpha)$, are defined inductively as follows:

(1) the empty path ϵ is in $Paths(\Delta)$ and $type(\epsilon) = DBtype$;

(2) for any $\alpha \in Paths(\Delta)$, where $type(\alpha) = \tau$,

- if for some $C \in \mathcal{C}$, $\tau = C$ and $\nu(C) = \{t\}$ (or $\tau = DBtype = \{t\}$), then $\alpha \cdot *$ is a path in $Paths(\Delta)$ and $type(\alpha \cdot *) = t$;
- if for some $C \in \mathcal{C}$, $\tau = C$ and $\nu(C) = [l_1 : t_1, \ldots, l_n : t_n]$ (or $\tau = DBtype = [l_1 : t_1, \ldots, l_n : t_n]$), then for $i \in [1, n]$, $\alpha \cdot l_i$ is in $Paths(\Delta)$ and $type(\alpha \cdot l_i) = t_i$.

As in the untyped context, a path can be represented as a formula $\alpha(x, y)$ with two free variables x and y denoting the tail and head nodes of the path, respectively.

A *word constraint* φ *over schema* Δ is an expression of the form

$$\forall x\, (\alpha(r, x) \rightarrow \beta(r, x)),$$

where α and β are in $Paths(\Delta)$ and $type(\alpha) = type(\beta)$.

We denote the set of all word constraints over schema Δ as $P_w(\Delta)$, or simply as P_w when Δ is understood.

Example 4.2: The sentences

$$\phi = \forall x\, (*(r, x) \rightarrow \exists y\, (*(r, y) \wedge spouse(y, x)))$$
$$\varphi = \forall x\, (\exists y\, (*(r, y) \wedge spouse(y, x)) \rightarrow *(r, x))$$

are word constraints over the schema given in Example 4.1. In any instance (π, μ, d) of the schema, ϕ and φ are interpreted as

$$\forall x\,(x \in d \rightarrow \exists y\,(y \in d \wedge y.spouse = x)),$$
$$\forall x\,(\exists y\,(y \in d \wedge y.spouse = x) \rightarrow x \in d),$$

respectively. Here $y.spouse$ stands for the projection of record y at attribute $spouse$, and d is a subset of $\pi(Person)$. The constraint ϕ says "each person in the set d is the spouse of someone in d", and φ says "if a person is the spouse of someone in d, then the person is in d". ∎

As illustrated by the example above, word constraints over a schema Δ can be naturally interpreted in database instances of Δ. Likewise, the notion "$I \models \varphi$" can be defined for instance I of Δ and constraint φ of $P_w(\Delta)$.

The agreement between databases and their abstraction with respect to word constraints is revealed by the lemma below, which justifies the abstraction defined above. The proof of the lemma is straightforward and can be found in [10].

Lemma 4.1: Let Δ be a schema. For each $I \in \mathcal{I}(\Delta)$, there is $G \in \mathcal{U}_f(\Delta)$, such that

(†) for all $\varphi \in P_w(\Delta)$, $I \models \varphi$ iff $G \models \varphi$.

Similarly, for each $G \in \mathcal{U}_f(\Delta)$, there is $I \in \mathcal{I}(\Delta)$, such that (†) holds. ∎

4.2.4 Word constraint implication

The *(finite) implication problem for $P_w(\Delta)$ over schema Δ* is the problem of determining, given a finite subset $\Sigma \cup \{\varphi\}$ of $P_w(\Delta)$, whether each $G \in \mathcal{U}(\Delta)$ ($G \in \mathcal{U}_f(\Delta)$) that satisfies Σ is also a model of φ.

The decidability of word constraint implication in the context of semistructured databases was established in [3] by showing that a particular set of inference rules is sound and complete for the implication. This set consists of three rules: reflexivity, transitivity and right congruence. However, this set of rules is no longer complete for word constraint implication in the object-oriented model, and as a result, the proof given in [3] no longer holds in this typed context. To see this, consider the constraints ϕ and φ given in Example 4.2. It is not hard to see that using the set of inference rules above, φ is not provable from ϕ. More specifically, it can be shown that if φ were provable from ϕ using this set of rules, then the length of $lt(\varphi)$ would be strictly less than the length of $rt(\varphi)$. However, by the type constraint imposed by the schema given in Example 4.1, $\{\phi\} \models \varphi$ indeed holds. More specifically, let $I = (\pi, \mu, d)$ be any instance of the schema satisfying ϕ, $s = \{x.spouse \mid x \in d\}$, and $|d|$, $|s|$ denote the cardinalities of d and s, respectively. By the type constraint imposed by record type, $|s| \leq |d|$. By $I \models \phi$, $d \subseteq s$. Hence $d = s$, and consequently, $I \models \varphi$.

Next, we show that in the object-oriented model, word constraint implication is indeed decidable.

Proposition 4.2: Over arbitrary schema in the object-oriented model, the implication and finite implication problems for word constraints are decidable. ∎

Proof sketch: The decidability of the finite implication follows from the decidability of finite satisfiability problem of two-variable logic with counting [6], since the type constraints are expressible in two-variable logic with counting and all the word constraints are in FO^2.

By this result, for the decidability of the implication it suffices to show that the implication and finite implication coincide. That is, over arbitrary schema Δ and for each finite subset $\Sigma \cup \{\varphi\}$ of $P_w(\Delta)$, if $\bigwedge \Sigma \wedge \neg\varphi$ has a model in $\mathcal{U}(\Delta)$, then it has a model in $\mathcal{U}_f(\Delta)$. See [10] for the detailed proof. ∎

In two special cases, word constraint implication is decidable in PTIME. One is word constraint implication over *record schema*, by which we mean a schema that does not contain any set type. The other, referred to as *$*$-form (finite) implication*, is implication $\Sigma \models \varphi$ where each constraint in $\Sigma \cup \{\varphi\}$ is of the form

$$\forall x\,(\alpha(r, x) \rightarrow \beta \cdot *(r, x)).$$

Here $\cdot$ is the path concatenation operator defined in the last section.

The proofs of the next three propositions follow closely the argument in [3] for the PTIME decidability of word constraint implication in semistructured databases, and can be found in [10].

Proposition 4.3: Over arbitrary record schema in the object-oriented model, the implication and finite implication problems for word constraints are decidable in PTIME in the size of the implication and the size of the schema. ∎

Proof sketch: The set of inference rules consisting of reflexivity, transitivity, right congruence and commutativity is sound and complete for the finite implication over record schema. ∎

Proposition 4.4: Over arbitrary schema in the object-oriented model, the $*$-form implication and finite implication problems for word constraints are decidable in PTIME in the size of the implication and the size of the schema. ∎

4.3 The ACeDB model

We next consider word constraint implication in an object-oriented model based on ACeDB [14]. This model does not have an explicit set construct, and in addition, it does not interpret a record type as a function from attributes to corresponding domains. More specifically, a value of a record type $[l_1 : t_1, \ldots, l_n : t_n]$ is a finite subset of

$$(\{l_1\} \times [\![t_1]\!]) \cup \ldots \cup (\{l_n\} \times [\![t_n]\!]),$$

where $[\![t_i]\!]$ denotes the domain of t_i.

The ACeDB based model is defined in the same way as the object-oriented model defined above, except the difference aforementioned. Similarly, the abstraction of the databases and word constraints in the model are defined as before, except that the constraint imposed by a record type $\tau = [l_1 : t_1, \ldots, l_n : t_n]$ is now defined by

$$\forall x\,(R_\tau(x) \rightarrow \forall y\,(\bigwedge_{l \in E(\Delta) \setminus \{l_1, \ldots, l_n\}} \neg l(x, y)) \wedge \bigwedge_{i \in [1,n]} \forall y\,(l_i(x, y) \rightarrow R_{t_i}(y))).$$

Proposition 4.5: Over arbitrary schema in the ACeDB model, the implication and finite implication problems for word constraints are decidable in PTIME in the size of the implication and the size of the schema. ∎

5 Conclusions

We have presented a class of path constraints, P, and investigated its associated implication problems. These constraints are important in both structured and semistructured data for specifying natural integrity constraints. They are not only a fundamental part of the semantics of the data; they are also important in query optimization. For example, the familiar inverse constraints that occur in object-oriented databases can be stated as path constraints of P.

In the context of semistructured data, we have shown that, despite the simple syntax of the language P, its associated implication problem is r.e. complete and its finite implication problem is co-r.e. complete. In light of these undecidability results, we have also identified several fragments of P which suffice to express many interesting semantic relations such as local database constraints and inverse constraints, and we have established the decidability of the implication and finite implication problems associated with each of these fragments.

In the context of structured data, we have shown that type constraints interact with path constraints. Because of this interaction, we have investigated word constraint implication in the context of two practical object-oriented models. We have presented abstractions for the databases in these models in terms of first-order logic, and we have established the decidability of word constraint implication in each of these models.

However, much work remains to be done.

Path constraint implication in general type systems. Path constraint implication in the presence of types is a rich source of questions. The diversity of the settings of data models and constraint languages raises a great number of implication problems. So far we have only investigated word constraint implication in the contexts of two practical yet restricted object-oriented type systems. Questions left open include implication problems for other fragments of P, such as those identified in Section 3, in the context of more general data models.

The complexity of reasoning about path constraints. The complexity of path constraint implication can probably be improved. Currently we are investigating methodology and automated tools for reasoning about path constraints with satisfactory average-case performance in practice.

Incremental path constraint satisfaction. Satisfaction checking is an essential issue in connection with path constraints. Equally important is incremental path constraint satisfaction. Databases are dynamic in the sense that they are subject to updates. Small updates to a large database often causes small changes in the outcome. The challenge is to check path constraint satisfaction incrementally by examining the parts of databases affected by updates, rather than by re-checking the entire databases from scratch. Incremental satisfaction offers a promising approach to maintaining path constraints efficiently, and requires serious research.

Methodology for using constraints in optimization. The use of path constraints in query optimization has been briefly addressed in this paper. The need for an in-depth study of this comes from the quest for a query optimizer based on both path constraints and algebraic rewrite rules.

Acknowledgements. The authors thank Victor Vianu, Val Tannen and Susan Davidson for helpful discussions.

References

[1] S. Abiteboul. "Querying semi-structured data". In *Proc. ICDT*, 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Weiner. "The lorel query language for semistructured data". *Journal of Digital Libraries*, 1(1), 1997.

[3] S. Abiteboul and V. Vianu. "Regular path queries with constraints", In *Proc. ACM Symp. on Principles of Database Systems*, 1997.

[4] F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an object-oriented database system: the story of O2*. Morgan Kaufmann, San Mateo, California, 1992.

[5] J. Barwise. "On Moschovakis closure ordinals". *Journal of Symbolic Logic*, 42:292-296, 1977.

[6] E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Springer, 1997.

[7] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. "A query language and optimization techniques for unstructured data". In *Proc. ACM SIGMOD International Conf. on Management of Data*, pp. 505-516, 1996.

[8] P. Buneman, W. Fan, and S. Weinstein. "Some undecidable implication problems for path constraints". Technical Report MS-CIS-97-14, Department of Computer and Information Science, University of Pennsylvania, 1997.

[9] P. Buneman, W. Fan, and S. Weinstein. "The decidability of some restricted implication problems for path constraints". Technical Report MS-CIS-97-15, Department of Computer and Information Science, University of Pennsylvania, 1997.

[10] P. Buneman, W. Fan, and S. Weinstein. "Path constraints in the presence of types". Technical Report MS-CIS-97-16, Department of Computer and Information Science, University of Pennsylvania, 1997.

[11] R. G. G. Cattell (ed.). *The object-oriented standard: ODMG-93* (Release 1.2). Morgan Kaufmann, San Mateo, California, 1996.

[12] E. Grädel, P. Kolaitis, and M. Vardi. "On the decision problem for two-variable first-order logic". *Bulletin of Symbolic Logic*, 3(1): 53-69, 1997.

[13] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. "The ObjectStore Database system". *Comm. ACM*, 34(10): 51-63, October 1991.

[14] 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.

[15] H. Wang. "Dominoes and the $\forall\exists\forall$-case of the decision problem". In *Proc. Symp. on Mathematical Theory of Automata*, Brooklyn Polytechnic Institute, pp. 23-55, 1962.