Interaction between Path and Type Constraints

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

XML [7], which is emerging as an important standard for data exchange on the World-Wide Web, highlights the importance of semistructured data. Although the XML standard itself does not require any schema or type system, a number of proposals [6, 17, 19] have been developed that roughly correspond to data definition languages. These allow one to constrain the structure of XML data by imposing a schema on it. These and other proposals also advocate the need for integrity constraints, another form of constraints that should, for example, be capable of expressing inclusion constraints and inverse relationships. The latter have recently been studied as path constraints in the context of semistructured data [4, 9]. It is likely that future XML proposals will involve both forms of constraints, and it is therefore appropriate to understand the interaction between them.

This paper investigates that interaction. In particular it studies constraint implication problems, which are important both in understanding the semantics of type/constraint systems and in query optimization. A number of results on path constraint implication are established in the presence and absence of type systems. These results demonstrate that adding a type system may in some cases simplify reasoning about path constraints and in other cases make it harder. For example, it is shown that there is a path constraint implication problem that is decidable in PTIME in the untyped context, but that becomes undecidable when a type system is added. On the other hand, there is an implication problem that is undecidable in the untyped context, but becomes not only decidable in cubic time but also finitely axiomatizable when a type system is imposed.

1 Introduction

Among the numerous proposals for adding structure or semantics to XML documents [7], several [6, 17, 18, 19] advocate the need for integrity constraints. However, concrete proposals for constraint systems have yet to be developed. Whether such constraints will be specified as extensions to existing type systems such as XML-Data [19], SOX [17], DCD [6], or whether they will be added as independent constructs, is not yet clear, and, in all probability, they will be added in both ways. XLink [21], for example, is independent of any type system and can express simple co-reference constraints. It is therefore appropriate to study constraints and type systems separately and to understand their interaction.

Integrity constraints for semistructured data were originally studied as path constraints in [4]. While these constraints could specify inclusions between paths, they were not expressive enough to capture, say, inverse constraints. Extensions were studied in [9] to overcome this limitation. The central technical problem investigated in these papers has been the question of constraint implication: given that certain constraints are known to hold, does it follow that some other constraint is necessarily satisfied? A number of decidability and undecidability results were established in these papers for semistructured data, i.e., data unconstrained by any type system or schema. In this paper, we extend the work reported in [9] by investigating the interaction between type systems and constraint systems. An interesting result presented here is that adding a type system may in some cases simplify the analysis of path constraint implication and in other cases make it harder. On the one hand, we exhibit an implication problem associated with path constraints that is undecidable in the context of semistructured data, but that becomes decidable in cubic-time when a (restricted) type system is added. On the other hand, we give an example of a constraint implication problem that is decidable in PTIME in the untyped context, but that becomes undecidable when a (generic) type system is imposed. The practical interest of these implication problems is addressed in Section 2.

An example. To cast the problem concretely, the structure represented in Figure 1 describes an XML document. It is an example of semistructured data and could be expressed in a number of other data formats. In semistructured data models, data is represented as a rooted, edge-labeled, directed graph [1, 8]. In Figure 1, vertices denote XML elements, and edges emanating from those nodes indicate attributes and relationships with other elements. For example, an edge labeled `book` from the root node r connects to a node representing a book element. This book node may have several `author` edges connected to person nodes, and `ref` edges connected to other book nodes. It may also have edges labeled with `ISBN`, `title` and `year`.

Typical path constraints on this graph describe an inverse relationship between `author` and `wrote`. This can be expressed as:

$$\forall x\,(book(r,\,x) \rightarrow \forall y\,(author(x,\,y) \rightarrow wrote(y,\,x)))$$
$$\forall x\,(person(r,\,x) \rightarrow \forall y\,(wrote(x,\,y) \rightarrow author(y,\,x)))$$

Here r is a constant denoting the root of the graph, variables x and y range over vertices, and the predicates denote edge

[*]This work was partly supported by the Army Research Office (DAAH04-95-1-0169) and NSF Grant CCR92-16122.

[†]Supported by a graduate fellowship from the Institute for Research in Cognitive Science, University of Pennsylvania.

[‡]Supported by NSF Grant CCR-9403447.

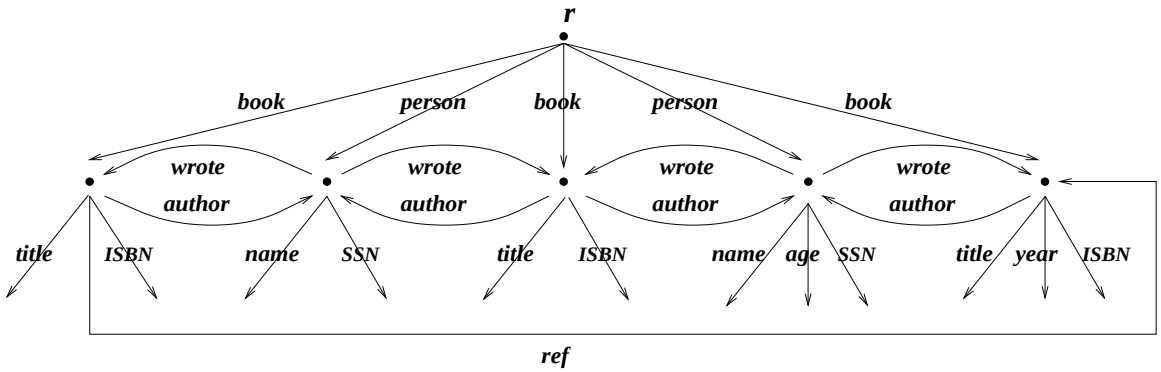

Figure 1: Representation of an XML document

labels. A path in the graph is a sequence of edge labels, which can be expressed as a formula $\alpha(x, y)$ denoting that α is a sequence of edge labels from vertex x to y. For example, $book \cdot author(r, x)$ is a path from root r to some vertex x in Figure 1. The first constraint above states that for any book node x and any y, if x has an `author` edge connected to y, then y must have a `wrote` edge connected to x. Similarly, the second constraint states that for any person node x and any y, if x has a `wrote` edge connected to y, then y must have an `author` edge connected to x.

Note that we have introduced these constraints *before* any mention of a type system. These are the kind of constraints that have been studied in [4, 9].

In addition we may also want to impose a *type* on the document. For example, a type specified in XML-Data [19] would be:

```
<elementType  id = "book">
  <attribute name="author"  range="#person"/>
  <attribute name="ref"     range="#book"/>
  <element type="#ISBN"/>
  <element type="#title"/>
  <element type="#year" occurs="optional"/>
</elementType>

<elementType  id = "person">
  <attribute name="wrote" range= "#book"/>
  <element type="#SSN"/>
  <element type="#name"/>
  <element type="#age" occurs="optional"/>
</elementType>

<elementType  id = "title">
  <string/>
</elementType>
  ...
```

This type specifies that a book node must have a `title` edge connected to a string node, its `author` and `ref` edges must connect to person and book nodes respectively, etc.

Types also constrain the data, but in a very different fashion. We are therefore interested in the interaction between these two forms of constraints.

Word and path constraints. A class of constraints, called *word constraints*, was introduced and studied in [4]. Referring to Figure 1, typical word constraints are:

$$\forall x\,(book \cdot author(r, x) \rightarrow person(r, x))$$
$$\forall x\,(person \cdot wrote(r, x) \rightarrow book(r, x))$$
$$\forall x\,(book \cdot ref(r, x) \rightarrow book(r, x))$$

Suppose Figure 1 represents a bibliography database at University of Pennsylvania. Let us refer to this database as Penn-bib. Abusing object-oriented database terms, the word constraints above assert that an author of a book in Penn-bib must be in the database "extent" of `person` in Penn-bib, a book written by a person in Penn-bib must occur in Penn-bib "extent" of `book`, etc. These are typical integrity constraints and were called *extent constraints* in [9]. It was shown in [4] that in the context of semistructured data, the implication and finite implication problems for word constraints are decidable in PTIME.

The class of path constraints studied in [9], P_c, is a mild generalization of word constraints. The inverse constraints above are in P_c but are not word constraints. As another example, consider Penn-bib again. This database may have links to external resources, such as bibliography databases at MIT and Warner. Call them MIT-bib and Warner-bib, respectively. These databases can be viewed as components of Penn-bib, and therefore, are called *local databases* of Penn-bib. In our graph representation, this can be depicted by adding two edges emanating from the root r of Penn-bib that are labeled with `MIT`, `Warner`, and lead to MIT-bib and Warner-bib, respectively. It is natural to expect the constraints given above to hold on these local databases. For example, the inverse constraints on MIT-bib include:

$$\forall x\,(MIT \cdot book(r, x) \rightarrow \forall y\,(author(x, y) \rightarrow wrote(y, x)))$$
$$\forall x\,(MIT \cdot person(r, x) \rightarrow \forall y\,(wrote(x, y) \rightarrow author(y, x)))$$

Constraints on local databases are called *local database constraints*. Again, these are P_c constraints but are not examples of word constraints. As demonstrated in [9], P_c 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. They are useful for, among other things, specifying and querying XML documents.

In [9], it was shown that in the context of semistructured data, the implication and finite implication problems for P_c are undecidable. However, several decidable fragments of P_c were identified. Each of these fragments properly contains the class of word constraints, and is capable of expressing extent, inverse and local database constraints.

Also considered in [4] was a class of constraints in which paths are represented by regular expressions. The decidability of the implication problems for this general constraint language was established in [4] for semistructured data. This constraint language differs from the constraint language P_c of [9] in expressive power. On the one hand, the language

of [4] allows a more general form of path expressions than P_c. On the other hand, it cannot capture inverse and local database constraints, whereas these constraints are expressible in P_c. Indeed, the language of [4] is contained in $L^2_{\infty\omega}$, the two variable fragment of the infinitary language $L_{\infty\omega}$, whereas P_c expresses constraints which are not $L^2_{\infty\omega}$ definable, as observed in [9]. Since the constraint language P_c is neither included in $L^2_{\infty\omega}$ nor categorized as a quantifier prefix fragment of first-order logic, our results concerning the implication problems for P_c are orthogonal to classical work on the decision problem for fragments of first-order logic (cf. [5]). In comparing the current work to [4], it should also be noted that [4] does not consider the question of logical implication in the context of typed data. The aim of this paper is to explore the interaction between type systems and simple integrity constraints of P_c. We do not consider here constraints defined in terms of regular expressions.

Type systems. In this paper, we consider two object-oriented data models. One is a generic type system, referred to as $\mathcal{M}^+$. This model supports classes, sets, records and recursive data structures. It is similar to those studied in [2, 3, 11]. The other model, $\mathcal{M}$, is a restriction of $\mathcal{M}^+$. It supports classes, records and recursive data structures, but does not allow sets. Databases of $\mathcal{M}$ are comparable to feature structures studied in feature logics [23].

We use these models to demonstrate the impact of different type constructs such as record and set on path constraint implication. One may want to study the interaction between path constraints and richer type systems such as those studied in [6, 17, 19]. However, by the results established in this paper, path constraint implication will be undecidable in the context of these more general type systems.

Constraints in object-oriented databases – a retrospective. While there has been considerable recent activity [12, 13, 16, 22] in optimizing object-oriented queries in the presence of constraints, there has, to our knowledge, been almost no work on the formulation of constraints, let alone the study of the implication problem. In [22] a rather general approach is taken: constraints are represented as queries that are true, and a general framework for program optimization is used to deal with both the optimization and the implication problem. In this setting, constraints are at least as expressive as first-order logic, and the issue of what classes of constraints have decidable implication problems is not separated from the general optimization problem.

Given the semistructured representation that we have adopted, we can cleanly separate typing issues from other constraints. Consider the following ODL [11] specification (loosely related to our previous example) which defines *Book* and *Person* classes:

```
interface Book
   (extent book)                                   (B1)
   {  attribute String title;
      relationship set<Person> author              (B2)
               inverse Person::wrote;              (B3)
   }

interface Person
   (extent person)                                 (P1)
   {  attribute String name;
      relationship set<Book> wrote                 (P2)
               inverse Book::author;               (P3)
   }
```

Strike out the **extent** and **inverse** declarations at lines B1, B3, P1, P3, and change **relationship** to **attribute** on lines B2 and P2. One is now left with a standard object-oriented class/type declaration. In fact it is a declaration that can be expressed directly in a language such as C++ with type templates.

We can consider the **extent** and **inverse** declarations as added constraints:

- *Extent constraints.* For any book b, $b.author$ is a subset of the extent *person*. Similarly, for any person p, $p.wrote$ is a subset of extent *book*.
- *Inverse constraints.* For any book b and for any p in $b.author$, b is a member of $p.wrote$. Similarly, for any person p and for any b in $p.wrote$, p is a member of $b.author$.

Thus, if we consider a database instance to be a graph (such as Figure 1 suitably modified) we can understand an ODL schema as imposing two kinds of constraints: (a) type constraints, which dictate the general structure of the graph, and (b) path constraints which dictate inclusions among certain sets of objects. We should remark that type constraints cannot be expressed as path constraints and *vice versa*.

From recent work [4, 9] on path constraints we have developed a reasonable understanding – in the context of semistructured (i.e. untyped) data – of the interesting decision problems for such constraints. There are useful restrictions of path constraints with a decidable implication problem. One might be tempted to think that the imposition of a type system, which imposes some regularity on the data, would be to generate new classes of path constraints with decidable implication problems. This may be the case. However one of the main results of this paper is to establish the possibly surprising result that the presence of types actually *complicates* the implication problem for path constraints: there are decidable path constraint problems that become undecidable in the presence of types. Moreover the type used in the construction of this result is not particularly "pathological".

Interaction. In Sections 4 and 5, we will show how imposing a schema on the data can alter the computational complexity of the path constraint implication problem in unexpected ways. For orientation, we provide intuitive background here. An implication problem for a logical language L is determined by a collection of structures $\mathcal{S}$ which interpret that language. We say that a finite set Σ of L sentences $\mathcal{S}$-*implies* an L sentence φ just in case for every structure $G \in \mathcal{S}$, if $G \models \Sigma$, then $G \models \varphi$. Suppose we are given two classes of structures $\mathcal{S}' \subset \mathcal{S}$, each interpreting L. In general, the computational complexity of the $\mathcal{S}$-implication problem for L may bear no obvious connection to the complexity of the $\mathcal{S}'$-implication problem for L. A justly famous example of this is given by the case where L is the collection of all first-order sentences with a single binary relation and $\mathcal{S}$ and $\mathcal{S}'$ are the classes of all relational structures and all finite relational structures respectively. Then, the completeness theorem for first-order logic and Church's Theorem together tell us that the $\mathcal{S}$-implication problem for L is r.e.-complete, while Trahktenbrot's Theorem tells us that the $\mathcal{S}'$-implication problem for L is co-r.e.-complete (see, e.g., [5]). Note that in this example, $\mathcal{S}'$ is not first order definable over $\mathcal{S}$.

In Sections 4 and 5 we will study implication problems for collections of path constraints which can be represented as proper fragments L^* of first-order logic. Again, let $\mathcal{S}$

be the collection of all structures. When we consider the $\mathcal{S}$-implication problem for L^* in the context of a type constraint Φ, what we really mean is the $\mathcal{S}''$-implication problem for L^* where $\mathcal{S}''$ is the collection of structures in $\mathcal{S}$ which satisfy the type constraint Φ. In Section 4, we will give examples where the $\mathcal{S}$-implication problem for L^* is undecidable, but the $\mathcal{S}''$-implication problem for L^* is decidable. This sort of situation is quite familiar. For example, the $\mathcal{S}$-implication problem for first-order logic is undecidable, but the $\mathcal{S}''$-implication problem for first-order logic is decidable when $\mathcal{S}''$ is the collection of linear orderings (and this collection *is* determined by a first order "constraint"). On the other hand, in Section 5, we exhibit situations in which the $\mathcal{S}$-implication problem for L^* is decidable, but the $\mathcal{S}''$-implication problem for L^* is undecidable. This possibility is perhaps a bit less familiar, namely the possibility that by imposing a restriction on a collection of structures we can turn a decidable implication problem into an undecidable implication problem. Indeed, in the context where L is the collection of all first-order sentences and the restriction itself is first order, this is clearly *impossible*, since in this case, the implication problem for the restricted class is simply a special case of the unrestricted implication problem. But in the context of the interaction between path and type constraints, this is precisely not the case. Namely, the type constraints we consider *cannot* be expressed in the path constraint languages in question. We hope this observation will clarify the results of Section 5, which exhibit a path constraint implication problem which is decidable with respect to a collection of structures $\mathcal{S}$, but is undecidable with respect to the collection of structures $G \in \mathcal{S}$ which satisfy a given type constraint Φ.

Organization. The remainder of the paper is organized as follows. Section 2 reviews the formal definition of P_c constraints, and describes two (finite) implication problems associated with P_c constraints, namely, *the (finite) implication problem for* P_c and *the (finite) implication problem for local extent constraints*. Section 3 presents a semistructured data model and the two object-oriented models $\mathcal{M}^+$ and $\mathcal{M}$. It also describes type constraints of $\mathcal{M}^+$ and $\mathcal{M}$. Section 4 investigates the (finite) implication problem for P_c in the context of semistructured data and in the object-oriented model $\mathcal{M}$. It first strengthens the undecidability result reported in [9] by showing that this problem is also undecidable on untyped data for a "small" fragment of P_c. It then shows that the undecidability result breaks down when the type system $\mathcal{M}$ is added. More precisely, it shows that in the context of $\mathcal{M}$, the implication and finite implication problems for P_c are not only decidable in cubic-time but also finitely axiomatizable. Section 5 demonstrates that adding a type system does not necessarily "help" in constraint implication problems. More specifically, it shows that on untyped data, the (finite) implication problem for local extent constraints is decidable in PTIME. However, when a type of $\mathcal{M}^+$ is imposed, this problem becomes undecidable. Finally, Section 6 briefly describes other results established in the full paper [10], and identifies directions for further work.

2 Path constraints

We first review the path constraint language P_c introduced in [9], and then describe two implication problems associated with P_c constraints. In Sections 4 and 5, we shall show that these problems have wildly different complexities in the context of untyped data as opposed to typed data.

2.1 Path constraint language P_c

The vocabulary of the constraint language is specified by a relational signature

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

where r is a constant and E is a finite set of binary relation symbols. A σ-structure $(|G|,\ r^G,\ E^G)$ can be depicted as an edge-labeled, rooted, directed graph, in which $|G|$ is the set of vertices, r^G the root, and E^G the set of labeled edges. For example, the graph in Figure 1 can be viewed as such a structure (referred to as G_0).

A path is a finite sequence of labels of E. Following [9], we define a *path* to be a formula $\alpha(x, y)$ which has one of the following forms:

- $x = y$, denoted by $\epsilon(x, y)$ and called the *empty path*;
- $\exists z(K(x, z) \wedge \beta(z, y))$, 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. Intuitively, if x and y are vertices in a σ-structure G, $\alpha(x, y)$ is true in G just when y is reachable from x by following a sequence of labeled edges α. We write $\alpha(x, y)$ as α when the parameters x and y are clear from the context.

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

- $\beta(x, y)$, if $\alpha = \epsilon$;
- $\exists\, u\,(K(x, u) \wedge (\alpha'(u, z) \cdot \beta(z, y)))$, if $\alpha(x, z)$ is of the form $\exists\, u\,(K(x, u) \wedge \alpha'(u, z))$.

A path α is said to be a *prefix* of β, denoted by $\alpha \preceq_p \beta$, if there exists γ, such that $\beta = \alpha \cdot \gamma$.

Referring to G_0 given in Figure 1, there is node x such that $person \cdot wrote \cdot ref(r,\ x)$ is true in G_0. In first-order logic, this path can be expressed as

$$\exists\, y\,(person(r,\ y) \wedge \exists\, z\,(wrote(y,\ z) \wedge ref(z,\ x))).$$

The prefixes of this path are ϵ, $person$, $person \cdot wrote$ and itself.

Formally, P_c constraints can be defined as follows.

Definition 2.1 [9]: A *path constraint* φ is an expression of either the *forward* form

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

or the *backward* form

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

Here α, β, γ are paths. The path α is called the *prefix* of φ, denoted by $pf(\varphi)$.

The set of all path constraints is denoted by P_c. ∎

A forward constraint of P_c asserts that for any vertex x that is reached from the root r by following path α and for any vertex y that is reached from x by following path β, y is also reachable from x by following path γ. Similarly, a backward P_c constraint states that for any x that is reached from r by following α and for any y that is reached from x by following β, x is also reachable from y by following γ.

For example, all the integrity constraints encountered in Section 1 are in P_c. These include extent, inverse and local database constraints.

A proper subclass of P_c was introduced and investigated in [4]:

Definition 2.2 [4]: A *word constraint* is an expression of the form

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

where β and γ are paths. The set of all word constraints is denoted by P_w. ∎

In other words, a word constraint is a forward constraint of P_c with its prefix being the empty path ϵ. For example, the extent constraints given in Section 1 are word constraints, whereas the inverse and local database constraints are not.

2.2 Implication problems

To take advantage of path constraints, it is important to be able to reason about them. This gives rise to the question of logical implication of path constraints. In general, we may know that a set of path constraints is satisfied by a database. The question of logical implication is: what other path constraints are necessarily satisfied by the database? As shown in [9], path constraint implication is useful for, among other things, query optimization and constraint checking.

Below we describe implication and finite implication of P_c constraints. These notions will be refined in different database contexts in Section 3.

We assume the standard notions of model and implication from first-order logic [15]. Let G be a structure and φ be a P_c constraint. We use $G \models \varphi$ to denote that G *satisfies* φ (i.e., G *is a model of* φ). Let Σ be a finite set of P_c constraints. We use $G \models \Sigma$ to denote that G *satisfies* Σ (i.e., G *is a model of* Σ). That is, for every $\phi \in \Sigma$, $G \models \phi$.

The *implication problem for* P_c is the problem to determine, given any finite subset $\Sigma \cup \{\varphi\}$ of P_c, whether every model of Σ also satisfies φ. Similarly, the *finite implication problem for* P_c is the problem to determine whether every finite model of Σ also satisfies φ.

For example, let Σ be the set consisting of all the P_c constraints given in Section 1, and φ_0 be the constraint

$$\forall x\,(MIT(r, x) \rightarrow \forall y\,(book \cdot ref(x, y) \rightarrow book(x, y))).$$

The question whether every (finite) model of Σ also satisfies φ_0 is an instance of the (finite) implication problem for P_c.

In Section 4, we shall show that the implication and finite implication problems for P_c are undecidable in the context of untyped data. In contrast, these problems are not only decidable in cubic-time but also finitely axiomatizable in the context of an object-oriented model.

In light of this undecidability result on untyped data, we next consider a special case of P_c constraint implication, namely, (finite) implication of local extent constraints. To illustrate this, consider the database Penn-bib described in Section 1. This database has local databases MIT-bib, Warner-bib, etc. Extent constraints on these local databases are called *local extent constraints*. For example, the following are extent constraints on MIT-bib, and thus are local extent constraints of Penn-bib:

$$\forall x\,(MIT(r, x) \rightarrow \forall y\,(book \cdot author(x, y) \rightarrow person(x, y)))$$
$$\forall x\,(MIT(r, x) \rightarrow \forall y\,(person \cdot wrote(x, y) \rightarrow book(x, y)))$$

Suppose we want to know whether every model of these constraints also satisfies the constraint φ_0 given above, which is also a local extent constraint on MIT-bib. In addition, we consider this implication in the presence of constraints on other local databases, such as the following on Warner-bib:

$$\forall x\,(Warner \cdot book(r, x) \rightarrow \forall y\,(author(x, y) \rightarrow wrote(y, x)))$$
$$\forall x\,(Warner \cdot person(r, x) \rightarrow \forall y\,(wrote(x, y) \rightarrow author(y, x)))$$

More precisely, let Σ_0 be the set consisting of the two local extent constraints on MIT-bib and the constraints on Warner-bib given above. We are interested in whether every (finite) model of Σ_0 also satisfies φ_0.

In general, when represented in a global environment, constraints on a local database are augmented with a common prefix. For example, the constraints on MIT-bib are represented with common prefix MIT in Penn-bib. Thus we use the following notion to describe local extent constraints.

Definition 2.3: Let α be a path and K a binary relation symbol. A constraint φ of P_c is said to be *bounded by* α *and* K if it is of the form

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

where $\beta \neq \epsilon$ and $K \not\preceq_p \beta$ (i.e., K is not a prefix of β).

A *subset* Σ *of* P_c *with prefix bounded by* α *and* K is a finite subset of P_c such that for each $\varphi \in \Sigma$, either φ is bounded by α and K, or for some path α', $pf(\varphi) = \alpha \cdot \alpha'$ and $K \not\preceq_p \alpha'$. In addition, if $\alpha' = \epsilon$, then φ is of the form $\forall x\,(\alpha(r, x) \rightarrow \forall y\,(\epsilon(x, y) \rightarrow K(x, y)))$. Here $pf(\varphi)$ denotes the prefix of φ, as described in Definition 2.1. ∎

For example, Σ_0 given above is a subset of P_c with prefix bounded by the empty path ϵ and binary relation symbol MIT. In Σ_0, the extent constraints on MIT-bib are bounded by ϵ and MIT, whereas the constraints on Warner-bib are not. Intuitively, let DB be a database and DB_K be a local database connected to DB by path $\alpha \cdot K$. Constraints bounded by α and K can be viewed as local extent constraints on DB_K. A subset of P_c with prefix bounded by α and K consists of such local extent constraints and constraints on other local databases connected to DB by some path $\alpha \cdot \alpha'$, where $K \not\preceq_p \alpha'$. It can be partitioned into Σ_1 and Σ_2, where Σ_1 consists of local extent constraints on DB_K, and Σ_2 contains constraints on other local databases.

Definition 2.4: The *(finite) implication problem for local extent constraints* is the problem of determining, given any finite subset $\Sigma \cup \{\varphi\}$ of P_c with prefix bounded by α and K, where φ is a constraint bounded by α and K, whether every (finite) model of Σ also satisfies φ. ∎

For example, the question whether every (finite) model of Σ_0 also satisfies φ_0 is an instance of the (finite) implication problem for local extent constraints. Note that φ_0 is also bounded by ϵ and MIT.

In Section 5, we shall show that in the untyped context, constraints on other local databases (e.g., constraints in Σ_2) do not interact with implication and finite implication of local extent constraints on DB_K (e.g., constraints in Σ_1). As a result, the implication and finite implication problems for local extent constraints are decidable in PTIME in the context of semistructured data. However, this may no longer be true in the typed context. Indeed, these problems become undecidable in the context of an object-oriented model.

3 Semistructured data vs structured data

In this section, we consider semistructured data versus structured data. More specifically, we investigate three models: a semistructured data model and two object-oriented models. For each of these models, we present an abstraction of

databases in terms of first-order logic. In Sections 4 and 5, we use these abstractions to study path constraint implication in these models.

3.1 Semistructured data model

Semistructured data is characterized as having no type constraints, irregular structure and missing schema [1, 8]. That is, data whose structure is not constrained by a schema. Semistructured data is commonly found on the World-Wide Web, in biological databases and after data integration. In particular, documents of XML [7] are usually viewed as semistructured data [14].

As observed by [1, 8], semistructured data is best modeled as a rooted edge-labeled directed graph, unconstrained by any type system or schema. Along the same lines, we use an abstraction of semistructured databases as (finite) σ-structures. Here σ is a signature of the form (r, E) as described in Section 2, in which r denotes the root and E denotes the edge labels.

Below we refine the notion of path constraint implication in the context of semistructured data. We use $\Sigma \models \varphi$ to denote that Σ *implies* φ. That is, for every σ-structure G, if $G \models \Sigma$, then $G \models \varphi$. Similarly, we use $\Sigma \models_f \varphi$ to denote that Σ *finitely implies* φ. That is, for every finite σ-structure G, if $G \models \Sigma$, then $G \models \varphi$.

In the context of semistructured data, the (finite) implication problem for P_c is the problem to determine, given any finite subset $\Sigma \cup \{\varphi\}$ of P_c, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$). Similarly, the (finite) implication problem for local extent constraints can be formalized in the context of semistructured data.

3.2 Object-oriented model $\mathcal{M}^+$

Next, we consider structured data, by which we mean data constrained by a schema, such as data found for instance in object-oriented databases. In addition, as mentioned in Section 1, there are applications in which data usually considered to be semistructured, such as XML data, is further constrained by a schema.

We first study databases in a generic object-oriented model, $\mathcal{M}^+$. Similar to the models studied in [2, 3, 11], $\mathcal{M}^+$ supports classes, records, sets and recursive structures. We characterize schemas in $\mathcal{M}^+$ in terms of type constraints. In Section 5, we investigate the interaction between these type constraints and path constraints.

3.2.1 Schemas and instances

We describe schemas and instances of $\mathcal{M}^+$ as follows. Assume a fixed countable set of labels, $\mathcal{L}$, and a fixed finite set of *atomic types*, $\mathcal{B}$. Examples of atomic types include *int* and *string*.

Let $\mathcal{C}$ be some finite set of *classes*. The set of *types over* $\mathcal{C}$, $Types^{\mathcal{C}}$, is defined by the syntax:

$$\tau \quad ::= \quad b \mid C \mid \{\tau\} \mid [l_1 : \tau_1, \ldots, l_n : \tau_n]$$

where $b \in \mathcal{B}$, $C \in \mathcal{C}$, and $l_i \in \mathcal{L}$. The notations $\{\tau\}$ and $[l_1 : \tau_1, \ldots, l_n : \tau_n]$ represent *set type* and *record type*, respectively.

A *schema* Δ in $\mathcal{M}^+$ is a triple $(\mathcal{C}, \nu, DBtype)$, where

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

Here we assume that every database of a schema has a unique (persistent) entry point, and $DBtype$ in the schema specifies the type of the entry point.

Example 3.1: The XML document given in Figure 1 can be specified by a schema $(\mathcal{C}, \nu, DBtype)$ in $\mathcal{M}^+$ as follows (optional sub-elements are specified as sets):

- $\mathcal{C}$ consists of $Book$ and $Person$;
- ν maps $Book$ and $Person$ to record types:

$$\begin{aligned} Person &\mapsto [name : string, SSN : string, age : \{int\}, \\ &\qquad wrote : \{Book\}] \\ Book &\mapsto [title : string, ISBN : string, year : \{int\}, \\ &\qquad ref : \{Book\}, author : \{Person\}] \end{aligned}$$

- $DBtype$ is $[person : \{Person\}, book : \{Book\}]$. ∎

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

- π is an *oid* (object identity) *assignment* that maps each $C \in \mathcal{C}$ to a finite set of oids, $\pi(C)$, such that for all $C, C' \in \mathcal{C}$, $\pi(C) \cap \pi(C') = \emptyset$ if $C \neq C'$;
- for each $C \in \mathcal{C}$, μ maps each oid in $\pi(C)$ to a value in $[\![\nu(C)]\!]_\pi$, where

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

  here D_b denotes the domain of atomic type b;
- d is a value in $[\![DBtype]\!]_\pi$, which represents the (persistent) entry point into the database instance.

The set of all database instances of Δ is denoted by $\mathcal{I}(\Delta)$.

3.2.2 Type constraints

We next present an abstraction of databases in $\mathcal{M}^+$. Structured data can be viewed as semistructured data further constrained by a schema. Along the same lines of the abstraction of semistructured data given above, we represent a structured database as a first-order logic structure satisfying a certain type constraint. Such a structure can also be depicted as an edge-labeled, rooted, directed graph, which has a certain "shape" specified by the type constraint. This abstraction simplifies the analysis of the interaction between path constraints and the type system.

To do this, we first define the first-order signature determined by a schema.

Given a schema $\Delta = (\mathcal{C}, \nu, DBtype)$, we define *the set of binary relation symbols*, $E(\Delta)$, and *the set of unary relation symbols*, $T(\Delta)$, as follows:

- $DBtype \in T(\Delta)$ and $\mathcal{C} \subseteq T(\Delta)$;
- For each $\tau \in T(\Delta)$,
  - if $\tau = \{\tau'\}$ (or for some $C \in \mathcal{C}$, $\nu(C) = \{\tau'\}$), then τ' is in $T(\Delta)$ and $*$ is in $E(\Delta)$;
  - if $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or for some class $C \in \mathcal{C}$, $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), then for each $i \in [1, n]$, τ_i is in $T(\Delta)$ and l_i is in $E(\Delta)$.

Note here we use the distinguished binary relation $*$ to denote the set membership relation.

The *signature determined by schema* Δ is

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

where r is a constant symbol (denoting the root), $E(\Delta)$ is the finite set of binary relation symbols (denoting the edge labels) and $T(\Delta)$ is the finite set of unary relation symbols (denoting the sorts or types) defined above.

As an example, the signature determined by the schema given in Example 3.1 is $(r,\ E,\ T)$, where

- r is a constant, which in each instance $(\pi,\ \mu,\ d)$ of the schema intends to name d;
- E includes *person*, *book*, *name*, *SSN*, *wrote*, *age*, *title*, *ISBN*, *year*, *ref*, *author* and $*$;
- T includes *Person*, *Book*, *string*, $\{int\}$, $\{string\}$, $\{Book\}$, $\{Person\}$ and *DBtype*.

We represent an instance I of a schema Δ in $\mathcal{M}^+$ as a $\sigma(\Delta)$-structure G that satisfies a certain type constraint. More specifically, let $\Delta = (\mathcal{C},\ \nu,\ DBtype)$, $I = (\pi,\ \mu,\ d)$ and $G = (|G|,\ r^G,\ E^G,\ T^G)$. We use $|G|$, r^G, E^G and T^G to represent data entities, the entry point d, record labels and set membership, and the types of the data entities, respectively. This structure must satisfy the *type constraint imposed by* Δ, $\Phi(\Delta)$, which specifies restrictions on the edges going out of vertices of different types.

Based on the definition of database instances in $\mathcal{M}^+$, we give $\Phi(\Delta)$ as follows.

- Every element of $|G|$ has a unique type in $T(\Delta)$. In particular, r^G has *DBtype*.
- If an element a of $|G|$ has type τ, then a must satisfy the constraint imposed by τ:
  - If τ is an atomic type b, then a has no outgoing edge.
  - If $\tau = \{\tau'\}$, or τ is a class type C and $\nu(C)$ is $\{\tau'\}$, then all the outgoing edges of a are labeled with $*$ and lead to elements of type τ'.
    In addition, if $\tau = \{\tau'\}$, then for each $b \in |G|$ such that b also has type τ, $a = b$ iff for any $c \in |G|$, $G \models *(a,\ c) \leftrightarrow *(b,\ c)$.
  - If $\tau = [l_1 : \tau_1,\ \ldots,\ l_n : \tau_n]$, or τ is a class type C and $\nu(C) = [l_1 : \tau_1,\ \ldots,\ l_n : \tau_n]$, then a has exactly n outgoing edges. These edges are labeled with l_1, ..., l_n, respectively. In addition, for each $i \in [1, n]$, if $G \models l_i(a,\ o)$ for some $o \in |G|$, then o has type τ_i.
    Moreover, if $\tau = [l_1 : \tau_1,\ \ldots,\ l_n : \tau_n]$, then for each $b \in |G|$ having type τ, $a = b$ iff for any $i \in [1, n]$ and $c \in |G|$, $G \models l_i(a,\ c) \leftrightarrow l_i(b,\ c)$.

In general, for any node a in a graph representing a $\sigma(\Delta)$-structure, $\Phi(\Delta)$ places restrictions on the number of the edges going out of a, on the labels of these edges, and on the types of the nodes to which these edges connect.

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

Because of the type constraint $\Phi(\Delta)$, some sequences of labels in $E(\Delta)$ are not paths in any structure of $\mathcal{U}(\Delta)$. We are not interested in these edge label sequences. We will use $Paths(\Delta)$ to denote the set of *paths over* Δ. That is, for any sequence α of edge labels, $\alpha \in Paths(\Delta)$ iff there is $G \in \mathcal{U}(\Delta)$ such that $G \models \exists\, x\ \alpha(r,\ x)$. In addition, we assume that over any schema Δ in $\mathcal{M}^+$, P_c constraints are defined in terms of paths in $Paths(\Delta)$.

Path constraints of P_c can be naturally interpreted in database instances of a schema Δ in $\mathcal{M}^+$. That is, the notion "$I \models \varphi$" can be defined for an instance I of Δ and a constraint φ of P_c (see [10] for detailed discussions of this notion). Using this notion, the lemma below justifies the abstraction of databases of $\mathcal{M}^+$ defined above.

Lemma 3.1: Let Δ be any schema in $\mathcal{M}^+$. For each I in $\mathcal{I}(\Delta)$, there is $G \in \mathcal{U}_f(\Delta)$, such that

$$\text{for any } \varphi \in P_c,\ \ I \models \varphi \text{ iff } G \models \varphi. \qquad (\dagger)$$

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

In the typed context, path constraint implication is restricted by a schema. More specifically, let Δ be a schema in $\mathcal{M}^+$ and $\Sigma \cup \{\varphi\}$ be a finite subset of P_c. We use $\Sigma \models_\Delta \varphi$ to denote that Σ *implies* φ *over* Δ. That is, for every $G \in \mathcal{U}(\Delta)$, if $G \models \Sigma$ then $G \models \varphi$. Similarly, we use $\Sigma \models_{(f,\,\Delta)} \varphi$ to denote that Σ *finitely implies* φ *over* Δ. That is, for every $G \in \mathcal{U}_f(\Delta)$, if $G \models \Sigma$ then $G \models \varphi$.

In the context of $\mathcal{M}^+$, the (finite) implication problem for P_c is the problem of determining, given any finite subset $\Sigma \cup \{\varphi\}$ of P_c and any schema Δ in $\mathcal{M}^+$, whether $\Sigma \models_\Delta \varphi$ ($\Sigma \models_{(f,\,\Delta)} \varphi$). Similarly, the (finite) implication problem for local extent constraints can also be formalized in the context of $\mathcal{M}^+$.

3.3 Object-oriented model $\mathcal{M}$

We also consider a restriction of $\mathcal{M}^+$, denoted by $\mathcal{M}$. The model $\mathcal{M}$ supports classes, records and recursive structures. However, it does not allow sets. In addition, a record in $\mathcal{M}$ consists of values of atomic types and oids only. More specifically, let $\mathcal{C}$ be some finite set of classes. The set of *types over* $\mathcal{C}$ *in* $\mathcal{M}$ is defined by:

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

where $b \in \mathcal{B}$, $C \in \mathcal{C}$, and $l_i \in \mathcal{L}$.

The notions of schemas and instances in $\mathcal{M}$ can be defined in the same way as in $\mathcal{M}^+$. Databases of $\mathcal{M}$ are comparable to feature structures [23], which have proven useful for representing linguistic data.

Given an $\mathcal{M}$ schema Δ, we define $E(\Delta)$, $T(\Delta)$, $\sigma(\Delta)$, and type constraint $\Phi(\Delta)$ in the same way as in $\mathcal{M}^+$, except that set types are not considered here. Similarly, the notions of $\mathcal{U}_f(\Delta)$, $\mathcal{U}(\Delta)$ and $Paths(\Delta)$ can also be defined. Using $\mathcal{U}_f(\Delta)$ and $\mathcal{U}(\Delta)$, we can define the implication and finite implication problems for P_c and for local extent constraints in the context of $\mathcal{M}$ in the same way as in $\mathcal{M}^+$.

4 Implication of P_c constraints

This section shows that an undecidability result on path constraint implication established for semistructured data collapses when a type of $\mathcal{M}$ is imposed on the data. More specifically, we prove the following:

Theorem 4.1: In the context of semistructured data, the implication and finite implication problems for P_c are undecidable. ∎

Theorem 4.2: In the context of the object-oriented model $\mathcal{M}$, the implication and finite implication problems for P_c are decidable in cubic-time and are finitely axiomatizable. ∎

These theorems show that in some cases, adding a type system may simplify reasoning about path constraints.

4.1 Undecidability on untyped data

Theorem 4.1 was first shown in [9]. Here we strengthen the result by identifying an undecidable fragment of P_c. This "small" fragment of P_c is an even milder generalization of P_w, the class of word constraints introduced in [4] and described in Section 2.

We present the fragment as follows. Recall E, the finite set of binary relation symbols (edge labels) in signature σ defined in Section 2. Let K be a binary relation symbol in E. For each $\psi \in P_w$, where $\psi = \forall\, x\,(\beta(r,\, x) \rightarrow \gamma(r,\, x))$, let

$$\delta(\psi,\, K) = \forall\, x\,(K(r,\, x) \rightarrow \forall\, y(\beta(x,\, y) \rightarrow \gamma(x,\, y))).$$

The fragment is defined by

$$P_w(K) = P_w \cup \{\delta(\psi,\, K) \mid \psi \in P_w\}.$$

In the context of semistructured data, the (finite) implication problem for $P_w(K)$ is the problem to determine, given any finite subset $\Sigma \cup \{\varphi\}$ of $P_w(K)$, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$). The theorem below establishes the undecidability of these problems, from which Theorem 4.1 follows immediately.

Theorem 4.3: In the context of semistructured data, both the implication and finite implication problems for $P_w(K)$ are undecidable. ∎

These undecidability results are rather surprising since $P_w(K)$ generalizes P_w in such a mild way. As shown by [4], the implication and finite implication problems for P_w are decidable in PTIME.

We prove Theorem 4.3 by reduction from the word problem for (finite) monoids. Before we give the proof, we first review the word problem for (finite) monoids.

4.1.1 The word problem for (finite) monoids

Recall the following notions from [2, 20].

Let Γ be a finite alphabet and $(\Gamma^*,\, \cdot,\, \epsilon)$ the free monoid generated by Γ. An *equation* (over Γ) is a pair $(\alpha,\, \beta)$ of strings in Γ^*.

Let $\Theta = \{(\alpha_i, \beta_i) \mid \alpha_i, \beta_i \in \Gamma^*,\ i \in [1, n]\}$ and a *test equation* θ be $(\alpha,\, \beta)$. We use $\Theta \models \theta$ ($\Theta \models_f \theta$) to denote that for every (finite) monoid $(M,\, \circ,\, id)$ and every homomorphism $h : \Gamma^* \rightarrow M$, if $h(\alpha_i) = h(\beta_i)$ for each $i \in [1, n]$, then $h(\alpha) = h(\beta)$.

The *word problem for (finite) monoids* is the problem of determining, given Θ and θ, whether $\Theta \models \theta$ ($\Theta \models_f \theta$).

The following result is well-known (e.g., see [2, 20]).

Theorem 4.4: Both the word problem for monoids and the word problem for finite monoids are undecidable. ∎

4.1.2 Reduction from the word problem

We encode the word problem for (finite) monoids in terms of the (finite) implication problem for $P_w(K)$. Let Γ_0 be a finite alphabet and Θ_0 be a finite set of equations (over Γ_0). Assume

$$\begin{aligned} \Gamma_0 &= \{l_j \mid j \in [1, m]\}, \\ \Theta_0 &= \{(\alpha_i, \beta_i) \mid \alpha_i, \beta_i \in \Gamma_0^*,\ i \in [1, n]\}, \end{aligned}$$

and a first-order logic signature

$$\sigma_0 = (r,\ \Gamma_0 \cup \{K\}),$$

where $K \notin \Gamma_0$, r is a constant symbol, and $\Gamma_0 \cup \{K\}$ is a set of binary relation symbols. Note here that each letter in Γ_0 is a binary relation symbol in σ_0. Thus every $\alpha \in \Gamma_0^*$ can be represented as a path formula, also denoted by α. In addition, we use $\cdot$ to denote the concatenation operator for both paths and strings.

We encode Θ_0 in terms of $\Sigma \subseteq P_w(K)$, which consists of the following: for every $j \in [1, m]$,

$$\begin{aligned} \forall\, x\,(\epsilon(r, x) &\rightarrow K(r, x)), \\ \forall\, x\,(K \cdot l_j(r, x) &\rightarrow K(r, x)), \end{aligned}$$

and for each $(\alpha_i, \beta_i) \in \Theta_0$,

$$\begin{aligned} \forall\, x\,(K(r, x) &\rightarrow \forall\, y\,(\alpha_i(x, y) \rightarrow \beta_i(x, y))), \\ \forall\, x\,(K(r, x) &\rightarrow \forall\, y\,(\beta_i(x, y) \rightarrow \alpha_i(x, y))). \end{aligned}$$

Let $(\alpha,\, \beta)$ be a test equation over Γ_0. We encode $(\alpha,\, \beta)$ as a pair of constraints in P_w:

$$\begin{aligned} \varphi_{(\alpha,\beta)} &= \forall\, x\,(\alpha(r, x) \rightarrow \beta(r, x)) \\ \varphi_{(\beta,\alpha)} &= \forall\, x\,(\beta(r, x) \rightarrow \alpha(r, x)) \end{aligned}$$

The lemma below shows that the encoding above is indeed a reduction from the word problem for (finite) monoids. From this lemma and Theorem 4.4, Theorem 4.3 follows.

Lemma 4.5: In the context of semistructured data, for all $\alpha, \beta \in \Gamma_0^*$,

$$\Theta_0 \models (\alpha,\, \beta) \ \text{ iff } \ \Sigma \models \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}, \tag{a}$$

$$\Theta_0 \models_f (\alpha,\, \beta) \ \text{ iff } \ \Sigma \models_f \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}. \tag{b}$$

∎

Proof sketch: We give a proof sketch of (b). We omit the details of the lengthy proof due to the lack of space, but we encourage the reader to consult [10].

(*if*) Suppose that $\Theta_0 \not\models_f (\alpha,\, \beta)$. Then there exist a finite monoid M and a homomorphism $h : \Gamma_0^* \rightarrow M$ such that for any $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. We define an equivalence relation on Γ_0^* by:

$$\rho \approx \varrho \quad \text{iff} \quad h(\rho) = h(\varrho).$$

For every string $\rho \in \Gamma_0^*$, let $\widehat{\rho}$ be the equivalence class of ρ with respect to $\approx$, and let $o(\widehat{\rho})$ be a distinct node. Then we define a σ_0-structure $G = (|G|,\, r^G,\, E^G)$, such that $|G| = \{o(\widehat{\rho}) \mid \rho \in \Gamma_0^*\}$ and the root $r^G = o(\widehat{\epsilon})$. The binary relations are populated in G as follows: for each $\rho \in \Gamma_0^*$, let $G \models K(o(\widehat{\epsilon}),\, o(\widehat{\rho}))$, and for each $j \in [1, m]$, let $G \models l_j(o(\widehat{\rho}),\, o(\widehat{\rho \cdot l_j}))$. The structure G is shown in Figure 2. It can be verified that G is a finite model of $\bigwedge \Sigma \wedge \neg\varphi_{(\alpha,\beta)}$.

(*only if*) Suppose that there is a finite σ_0-structure G such that $G \models \bigwedge \Sigma \wedge (\neg\varphi_{(\alpha,\beta)} \vee \neg\varphi_{(\beta,\alpha)})$. Then we define another equivalence relation on Γ_0^* by:

$$\begin{aligned} \rho \sim \varrho \ \text{ iff } \ G \models\ & \forall\, x(K(r, x) \rightarrow \forall\, y\,(\rho(x, y) \rightarrow \varrho(x, y))) \wedge \\ & \forall\, x\,(K(r, x) \rightarrow \forall\, y\,(\varrho(x, y) \rightarrow \rho(x, y))). \end{aligned}$$

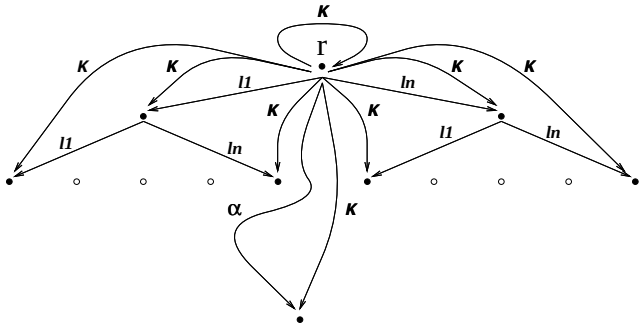

Figure 2: The structure G in the proof of Lemma 4.5

For any $\rho \in \Gamma_0^*$, let $[\rho]$ be the equivalence class of ρ with respect to $\sim$. Then we define $M = \{[\rho] \mid \rho \in \Gamma_0^*\}$, operator $\circ$ by $[\rho] \circ [\varrho] = [\rho \cdot \varrho]$, and $h : \Gamma_0^* \to M$ by $h : \rho \mapsto [\rho]$. It can be verified that $(M, \circ, [\epsilon])$ is indeed a finite monoid, h is a homomorphism, and in addition, for every $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$ but $h(\alpha) \neq h(\beta)$. ∎

4.2 The collapse of the undecidability in $\mathcal{M}$

We next show that in the context of the object-oriented model $\mathcal{M}$, the undecidability result established above no longer holds.

The collapse of the undecidability is due to the following lemma, which can be proved by a straightforward induction on the length of α and by using $\Phi(\Delta)$. On untyped data, this lemma does not hold in general.

Lemma 4.6: Let Δ be an arbitrary schema in $\mathcal{M}$, and $G \in \mathcal{U}(\Delta)$. Then for every α in $Paths(\Delta)$, there is a unique $o \in |G|$, such that $G \models \alpha(r^G, o)$. ∎

Using Lemma 4.6, it is easy to verify the following.

Lemma 4.7: Let Δ be a schema in $\mathcal{M}$, φ be a forward constraint $\forall x\,(\alpha(r, x) \to \forall y\,(\beta(x, y) \to \gamma(x, y)))$, and ψ be a word constraint $\forall x\,(\alpha \cdot \beta(r, x) \to \alpha \cdot \gamma(r, x))$. Then for any $G \in \mathcal{U}(\Delta)$, $G \models \varphi$ iff $G \models \psi$. ∎

Lemma 4.8: Let Δ be a schema in $\mathcal{M}$, φ be a backward constraint $\forall x\,(\alpha(r, x) \to \forall y\,(\beta(x, y) \to \gamma(y, x)))$, and ψ be a word constraint $\forall x\,(\alpha(r, x) \to \alpha \cdot \beta \cdot \gamma(r, x))$. Then for any $G \in \mathcal{U}(\Delta)$, $G \models \varphi$ iff $G \models \psi$. ∎

Based on Lemmas 4.7 and 4.8, we give a finite axiomatization $\mathcal{I}_r$ of P_c constraint implication as follows:

- Reflexivity:

$$\frac{}{\forall x\,(\alpha(r,x) \to \alpha(r,x))}$$

- Transitivity:

$$\frac{\forall x\,(\alpha(r,x) \to \beta(r,x)) \qquad \forall x\,(\beta(r,x) \to \gamma(r,x))}{\forall x\,(\alpha(r,x) \to \gamma(r,x))}$$

- Right-congruence:

$$\frac{\forall x\,(\alpha(r,x) \to \beta(r,x))}{\forall x\,(\alpha \cdot \gamma(r,x) \to \beta \cdot \gamma(r,x))}$$

- Commutativity:

$$\frac{\forall x\,(\alpha(r,x) \to \beta(r,x))}{\forall x\,(\beta(r,x) \to \alpha(r,x))}$$

- Forward-to-word:

$$\frac{\forall x\,(\alpha(r,x) \to \forall y\,(\beta(x,y) \to \gamma(x,y)))}{\forall x\,(\alpha \cdot \beta(r,x) \to \alpha \cdot \gamma(r,x))}$$

- Word-to-forward:

$$\frac{\forall x\,(\alpha \cdot \beta(r,x) \to \alpha \cdot \gamma(r,x))}{\forall x\,(\alpha(r,x) \to \forall y\,(\beta(x,y) \to \gamma(x,y)))}$$

- Backward-to-word:

$$\frac{\forall x\,(\alpha(r,x) \to \forall y\,(\beta(x,y) \to \gamma(y,x)))}{\forall x\,(\alpha(r,x) \to \alpha \cdot \beta \cdot \gamma(r,x))}$$

- Word-to-backward:

$$\frac{\forall x\,(\alpha(r,x) \to \alpha \cdot \beta \cdot \gamma(r,x))}{\forall x\,(\alpha(r,x) \to \forall y\,(\beta(x,y) \to \gamma(y,x)))}$$

The first three inference rules above were proposed in [4] and were shown to be complete for word constraint implication in the context of untyped data. In contrast, these three rules are no longer complete for word constraint implication in the context of $\mathcal{M}$.

Let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c. We use $\Sigma \vdash_{\mathcal{I}_r} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}_r$.

Theorem 4.9: Let Δ be any schema in $\mathcal{M}$. For every finite subset $\Sigma \cup \{\varphi\}$ of P_c,

$$\Sigma \models_{\Delta} \varphi \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \varphi,$$
$$\Sigma \models_{(f,\Delta)} \varphi \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \varphi.$$

∎

As an immediate result, in the context of $\mathcal{M}$, the implication and finite implication problems for P_c coincide and are decidable.

A proof sketch of Theorem 4.9 is as follows. Soundness of $\mathcal{I}_r$ can be verified by induction on the lengths of $\mathcal{I}_r$-proofs. For the proof of completeness, it suffices to show the existence of $G \in \mathcal{U}_f(\Delta)$ such that $G \models \Sigma$ and in addition, if $G \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_r} \varphi$. Owing to the space limit, we omit the lengthy definition of G, but we recommend the interested reader see [10] for a detailed proof.

Based on the axiomatization $\mathcal{I}_r$, a cubic-time algorithm can be given for testing implication and finite implication of P_c constraints in the context of $\mathcal{M}$. By Lemma 4.6, every constraint in Σ is applied at most once by the algorithm. It is because of this property that the algorithm has low complexity. Space limitations do not allow us to include the algorithm. The interested reader should consult [10].

Theorem 4.2 follows from Theorem 4.9 and the existence of the cubic-time algorithm.

5 Implication of local extent constraints

In light of Theorems 4.1 and 4.2, one is tempted to think that adding structure will simplify reasoning about path constraints. However, this is not always the case. This section shows that a decidability result developed for untyped data breaks down when a type of $\mathcal{M}^+$ is imposed on the data.

Theorem 5.1: In the context of semistructured data, the implication and finite implication problems for local extent constraints are decidable in PTIME. ∎

Theorem 5.2: In the context of the object-oriented data model $\mathcal{M}^+$, the implication and finite implication problems for local extent constraints are undecidable. ∎

These theorems demonstrate that adding a type system may also make the analysis of path constraint implication

more difficult. This may seem counterintuitive since at first glance, a type constraint appears to assert that the data has a regular structure and therefore, simplifies reasoning about path constraints. This appearance can be dispelled by noticing that the type constraint places restrictions on the structures considered in implication problems in a different way to path constraints. More specifically, let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c. In the untyped context, we may be able to find in PTIME a structure G such that $G \models \bigwedge \Sigma \wedge \neg\varphi$. However, when a schema Δ is imposed on the data, we may have that $G \notin \mathcal{U}(\Delta)$. That is, G is excluded from the set of structures considered in implication problems because of the type constraint $\Phi(\Delta)$ determined by Δ. Worse still, $\Phi(\Delta)$ may constrain the structure of the data in such a peculiar way that it is undecidable whether there is $H \in \mathcal{U}(\Delta)$ such that $H \models \bigwedge \Sigma \wedge \neg\varphi$.

5.1 Decidability on untyped data

We first show Theorem 5.1. The idea of the proof is by reduction to word constraint implication. It has been shown in [4] that in the context of untyped data, the implication and finite implication problems for P_w are decidable in PTIME.

We first define a function f that is used in the further construction of the reduction. Let α be a path and φ be a P_c constraint. Then $f(\alpha, \varphi)$ is defined to be the P_c constraint

- $\forall x\,(\alpha \cdot \rho(r, x) \rightarrow \forall y(\beta(x, y) \rightarrow \gamma(x, y)))$, if φ is of the form $\forall x\,(\rho(r, x) \rightarrow \forall y(\beta(x, y) \rightarrow \gamma(x, y)))$ (i.e., a forward constraint); or
- $\forall x\,(\alpha \cdot \rho(r, x) \rightarrow \forall y(\beta(x, y) \rightarrow \gamma(y, x)))$, if φ is of the form $\forall x\,(\rho(r, x) \rightarrow \forall y(\beta(x, y) \rightarrow \gamma(y, x)))$ (i.e., a backward constraint).

Recall the definition of the (finite) implication problem for local extent constraints from Definition 2.4. Let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c with prefix bounded by path α and binary relation symbol K, where φ is also bounded by α and K. By Definition 2.3, Σ can be partitioned into Σ_K and Σ_r:

$$\begin{aligned} \Sigma_K &= \{\phi \mid \phi \in \Sigma,\ \phi \text{ is bounded by } \alpha \text{ and } K\}, \\ \Sigma_r &= \Sigma \setminus \Sigma_K. \end{aligned}$$

In addition, for each $\phi \in \Sigma_K \cup \{\varphi\}$, ϕ is a forward constraint and the prefix of ϕ, $pf(\phi)$, is $\alpha \cdot K$. For each $\psi \in \Sigma_r$, $pf(\psi)$ is of the form $\alpha \cdot \alpha'$, where α' is a path such that $K \not\preceq_p \alpha'$, i.e., K is not a prefix of α'.

The reduction is defined in two steps. First, using f and α, we define a function g_1 such that for every $\phi \in \Sigma \cup \{\varphi\}$, $\phi = f(\alpha, g_1(\phi))$. That is, g_1 removes α from the prefix of ϕ. Let $\varphi^1 = g_1(\varphi)$ and

$$\begin{aligned} \Sigma_K^1 &= \{g_1(\phi) \mid \phi \in \Sigma_K\}, \\ \Sigma_r^1 &= \{g_1(\psi) \mid \psi \in \Sigma_r\}. \end{aligned}$$

Second, using f and K, we define another function g_2 such that for any $\phi \in \Sigma_K^1 \cup \{\varphi^1\}$, $\phi = f(K, g_2(\phi))$. That is, g_2 further removes K from the prefix of ϕ. Now let φ^2 be $g_2(\varphi^1)$ and $\Sigma_K^2 = \{g_2(\phi) \mid \phi \in \Sigma_K^1\}$. Clearly, $\Sigma_K^2 \subseteq P_w$ and $\varphi^2 \in P_w$. The functions g_1 and g_2 establish a reduction:

Lemma 5.3: In the context of semistructured data,

$$\Sigma \models \varphi \quad \text{iff} \quad \Sigma_K^1 \cup \Sigma_r^1 \models \varphi^1 \quad \text{iff} \quad \Sigma_K^2 \models \varphi^2, \qquad \text{(a)}$$
$$\Sigma \models_f \varphi \quad \text{iff} \quad \Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1 \quad \text{iff} \quad \Sigma_K^2 \models_f \varphi^2. \qquad \text{(b)}$$

■

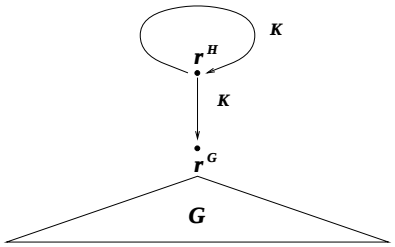

Figure 3: The structure H in the proof of Lemma 5.3

This lemma suffices to show Theorem 5.1. For if it holds, then the (finite) implication problem for local extent constraints is reduced to the (finite) implication problem for P_w. Note that given Σ and φ, α and K can be determined in linear-time. In addition, the functions g_1 and g_2 are computable in linear-time. Therefore, the PTIME decidability of the (finite) implication problem for local extent constraints follows from the PTIME decidability of the (finite) implication problem for P_w.

Next, we give a proof sketch of Lemma 5.3 (b). We omit the details of the proof due to the lack of space, but we suggest the reader consult [10].

Proof sketch: We first show that $\Sigma \models_f \varphi$ if and only if $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$. If $\bigwedge \Sigma_K^1 \wedge \bigwedge \Sigma_r^1 \wedge \neg\varphi^1$ has a finite model G_1, then we construct a structure G by adding to G_1 a new root r^G and a path α from r^G to r^{G_1}. It is easy to verify that G is a finite model of $\bigwedge \Sigma \wedge \neg\varphi$. Conversely, suppose that $\bigwedge \Sigma \wedge \neg\varphi$ has a finite model G. Assume that φ is of the form $\forall x\,(\alpha \cdot K(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y)))$. Thus by $G \models \neg\varphi$, there are vertices a, b, c in G such that $G \models \alpha(r^G, a) \wedge K(a, b) \wedge \beta(b, c) \wedge \neg\gamma(b, c)$. We construct a structure G_1 from G by letting a be the new root. It can be verified that G_1 is a finite model of $\bigwedge \Sigma_K^1 \wedge \bigwedge \Sigma_r^1 \wedge \neg\varphi^1$.

We next proceed to show that $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$ if and only if $\Sigma_K^2 \models_f \varphi^2$. The argument given above suffices to show that if $\Sigma_K^2 \models_f \varphi^2$ then $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$. Conversely, assume that $\bigwedge \Sigma_K^2 \wedge \neg\varphi^2$ has a finite model G. Based on G, we construct a structure H as shown in Figure 3. More specifically, let H be $(|H|, r^H, E^H)$, where $|H| = |G| \cup \{r^H\}$, the root node r^H is a new vertex which is not in $|G|$, and $E^H = E^G \cup \{K(r^H, r^H), K(r^H, r^G)\}$. By Definitions 2.3 and 2.4, it can be verified that H is indeed a finite model of $\bigwedge \Sigma_K^1 \wedge \bigwedge \Sigma_r^1 \wedge \neg\varphi^1$. ■

5.2 The breakdown of the decidability in $\mathcal{M}^+$

Next, we show that the decidability result established above breaks down in the context of $\mathcal{M}^+$. More specifically, we prove Theorem 5.2 by reduction from the word problem for (finite) monoids.

Recall Γ_0 and Θ_0 described in Section 4.1. Using Γ_0, we define an $\mathcal{M}^+$ schema $\Delta_1 = (\mathcal{C}, \nu, DBtype)$, where

- $\mathcal{C} = \{C, C_s, C_l\}$,
- ν is defined by:

$$\begin{aligned} C &\mapsto [l_1 : C, \ldots, l_m : C] \\ C_s &\mapsto \{C\} \\ C_l &\mapsto [a : C, b : C_s, K : C_l] \end{aligned}$$

where $a, b, K \notin \Gamma_0$.

- $DBtype = [l : C_l]$, where $l \notin \Gamma_0$.

Note here that each letter in Γ_0 is a record label of C, and thus is in $E(\Delta_1)$. Hence every $\alpha \in \Gamma_0^*$ can be represented as a path formula, also denoted by α.

We encode Θ_0 in terms of a finite set Σ, which consists of the following P_c constraints:

1. $\forall x\,(l \cdot K(r,x) \rightarrow \forall y\,(a(x,y) \rightarrow b \cdot *(x,y)))$;
2. for each $j \in [1,m]$,
$$\forall x\,(l \cdot K(r,x) \rightarrow \forall y\,(b \cdot * \cdot l_j(x,y) \rightarrow b \cdot *(x,y)));$$
3. for each $(\alpha_i,\ \beta_i) \in \Theta_0$,
$$\forall x\,(l \cdot b \cdot *(r,x) \rightarrow \forall y\,(\alpha_i(x,y) \rightarrow \beta_i(x,y)));$$
4. $\forall x\,(l(r,x) \rightarrow \forall y\,(\epsilon(x,y) \rightarrow K(x,y)))$.

We encode a test equation $(\alpha,\ \beta)$ over Γ_0 by the constraint:

$$\varphi_{(\alpha,\beta)} = \forall x\,(l \cdot K(r,x) \rightarrow \forall y\,(a \cdot \alpha(x,y) \rightarrow a \cdot \beta(x,y))).$$

By Definition 2.3, it is easy to see that $\Sigma \cup \{\varphi_{(\alpha,\beta)}\}$ is a subset of P_c with prefix bounded by l and K. More specifically, this set can be partitioned into Σ_r and Σ_K:

- Σ_K consists of $\varphi_{(\alpha,\beta)}$ as well as those defined in (1) and (2). These constraints are bounded by l and K.
- Σ_r consists of the constraints specified in (3) and (4), which are not bounded by l and K. In addition, for any $\phi \in \Sigma_r$, the prefix of ϕ, $pf(\phi)$, is either $l \cdot b \cdot *$ or l. In particular, if $pf(\phi) = l$, then ϕ is the constraint given in (4).

The lemma below shows that this encoding is a reduction from the word problem for (finite) monoids. Theorem 5.2 follows from this lemma and Theorem 4.4.

Lemma 5.4: In the context of $\mathcal{M}^+$, for all $\alpha, \beta \in \Gamma_0^*$,

$$\Theta_0 \models (\alpha,\ \beta) \quad \text{iff} \quad \Sigma \models_{\Delta_1} \varphi_{(\alpha,\beta)}, \tag{a}$$
$$\Theta_0 \models_f (\alpha,\ \beta) \quad \text{iff} \quad \Sigma \models_{(f,\,\Delta_1)} \varphi_{(\alpha,\beta)}. \tag{b}$$

■

The proof of this lemma uses the following property of Δ_1: For any $G \in \mathcal{U}(\Delta_1)$, there are unique vertices o_l, o_K in G such that $G \models l(r^G,\ o_l) \wedge K(o_l,\ o_K)$. In addition, if $G \models \Sigma$, then $o_l = o_K$. This holds due to the type constraint $\Phi(\Delta_1)$. A structure satisfying $\Phi(\Delta_1)$ and Σ must have the form shown in Figure 4. Unlike in semistructured data, here $\Sigma \models_{\Delta_1} \varphi_{(\alpha,\beta)}$ is no longer equivalent to $\Sigma_K \models_{\Delta_1} \varphi_{(\alpha,\beta)}$. That is, Σ_r interacts with $\Sigma_K \models_{\Delta_1} \varphi_{(\alpha,\beta)}$. We do not include the proof of this lemma due to the lack of space. The interested reader should see [10] for a detailed proof.

It should be mentioned that the proof of Theorem 5.1 is not applicable here. Note that the structure H shown in Figure 3 is not in $\mathcal{U}(\Delta_1)$, because of type constraint $\Phi(\Delta_1)$.

6 Conclusion

Two forms of constraints have been proposed separately for specifying semantics of XML data, namely, type constraints [6, 17, 19] and path constraints [4, 9]. In this paper, we have investigated their interaction. We have demonstrated that adding a type system may in some cases simplify the analysis of path constraint implication, and in other cases make it harder. More specifically, we have studied how P_c constraints introduced in [9] interact with two type systems.

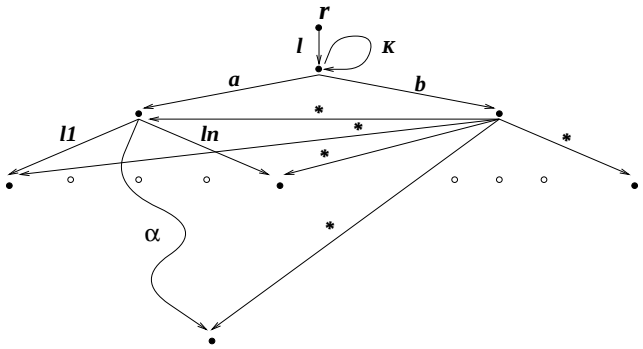

Figure 4: The structure G in the proof of Lemma 5.4

One of the type systems, $\mathcal{M}^+$, is an object-oriented model similar to those studied in [2, 3, 11]. It supports classes, records and sets. The other, $\mathcal{M}$, is a restriction of $\mathcal{M}^+$. On the one hand, we have shown that the implication and finite implication problems for P_c are undecidable in the context of semistructured data, but they become not only decidable in cubic-time but also finitely axiomatizable when a type of $\mathcal{M}$ is added. On the other hand, we have also shown that the implication and finite implication problems for local extent constraints, which constitute a fragment of P_c, are decidable in PTIME in the untyped context. However, when a type of $\mathcal{M}^+$ is imposed, these problems become undecidable.

Other results established in the full paper. Due to the lack of space, several results reported in [10] are not included in this paper. Below we mention some of them. We encourage the reader to consult [10].

Recall $P_w(K)$ described in Section 4.1. Similarly, given a path α, $P_w(\alpha)$ is defined to be a generalization of the class of word constraints as follows. For each $\psi \in P_w$, where $\psi = \forall x\,(\beta(r,\ x) \rightarrow \gamma(r,\ x))$, let

$$\delta(\psi,\ \alpha) = \forall x\,(\alpha(r,\ x) \rightarrow \forall y\,(\beta(x,\ y) \rightarrow \gamma(x,\ y))).$$

Then $P_w(\alpha)$ is defined by

$$P_w(\alpha) = P_w \cup \{\delta(\psi,\ \alpha) \mid \psi \in P_w\}.$$

Theorem 6.1: In the context of $\mathcal{M}^+$, the implication and finite implication problems for $P_w(\alpha)$ and therefore, for P_c, are undecidable. ■

Another object-oriented model, $\mathcal{M}_f^+$, was also studied in [10]. This model is the same as $\mathcal{M}^+$ except that it supports finite sets instead of sets. The major difference between $\mathcal{M}^+$ and $\mathcal{M}_f^+$ is described as follows. For any schema Δ in $\mathcal{M}^+$, the set of structures satisfying the type constraint $\Phi(\Delta)$, $\mathcal{U}(\Delta)$, is definable in first-order logic. In contrast, for a schema Δ in $\mathcal{M}_f^+$, $\mathcal{U}(\Delta)$ may not be first order definable. As a result, the equivalence of the implication problem and the finite implication problem for path constraints in $\mathcal{M}_f^+$ does not necessarily lead to the decidability of these problems.

It was shown in [10] that the results developed for $\mathcal{M}^+$ also hold for $\mathcal{M}_f^+$. The proofs of some of these results, however, are quite different from the analogous proofs for $\mathcal{M}^+$ for the reason mentioned above.

Theorem 6.2: In the context of $\mathcal{M}_f^+$, the implication and finite implication problems for $P_w(\alpha)$, for P_c, and for local extent constraints are all undecidable. ■

The main results of [10] are summarized in Table 1.

	(finite) implication problem for $P_w(\alpha)$	*(finite) implication problem for local extent constraints*	*(finite) implication problem for* P_c
semistructured data model	undecidable	decidable (PTIME)	undecidable
object-oriented model $\mathcal{M}$	decidable (cubic-time)	decidable (cubic-time)	decidable (cubic-time)
object-oriented model $\mathcal{M}^+$	undecidable	undecidable	undecidable
object-oriented model $\mathcal{M}^+_f$	undecidable	undecidable	undecidable

Table 1: The main results of the paper

Further research. It would be interesting to study the interaction between path constraints and less strict type systems such as the object relational data model.

To include path constraints in XML documents to specify the semantics of the data, it is important to have a path constraint syntax that conforms to XML and XML DTD [7]. In [10], we offered a preliminary proposal. To describe in this syntax external links such as those studied in [21], much more remains to be done.

Acknowledgements. The authors thank Leonid Libkin, Val Tannen and Victor Vianu for valuable comments and discussions.

References

[1] S. Abiteboul. "Querying semistructured data". In *Proc. 6th Int'l. Conf. on Database Theory (ICDT'97)*, 1997.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesly, 1995.

[3] S. Abiteboul and P. C. Kanellakis. "Object identity as a query primitive". In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, 1989.

[4] S. Abiteboul and V. Vianu. "Regular path queries with constraints". In *Proc. 16th ACM Symp. on Principles of Database Systems (PODS'97)*, 1997.

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

[6] T. Bray, C. Frankston, and A. Malhotra. "Document Content Description for XML". W3C Note NOTE-dcd-19980731. See `http://www.w3.org/TR/NOTE-dcd`.

[7] T. Bray, J. Paoli and C. M. Sperberg-McQueen. "Extensible Markup Language (XML) 1.0". W3C Recommendation REC-xml-19980210. Available as `http://www.w3.org/REC-xml`.

[8] P. Buneman. "Semistructured data". Tutorial in *Proc. 16th ACM Symp. on Principles of Database Systems (PODS'97)*, 1997.

[9] P. Buneman, W. Fan, and S. Weinstein. "Path constraints on semistructured and structured data". In *Proc. 17th ACM Symp. on Principles of Database Systems (PODS'98)*, 1998.

[10] P. Buneman, W. Fan and S. Weinstein. "Interaction between path and type constraints". Technical report MS-CIS-98-16, Department of Computer and Information Science, University of Pennsylvania, 1998. Available as `ftp://ftp.cis.upenn.edu/pub/papers/db-research/tr9816.ps.gz`.

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

[12] U. S. Chakravarthy, J. Grant, and J. Minker. "Foundations of semantic query optimization for deductive databases". In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, San Mateo, California, 1988.

[13] S. Cluet and C. Delobel. "A general framework for the optimization of object-oriented queries". In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, 1992.

[14] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. "XML-QL: a query language for XML". W3C Note NOTE-xml-ql-19980819. Available as `http://www.w3.org/TR/NOTE-xml-ql`.

[15] H. B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.

[16] D. Florescu, L. Raschid, and P. Valduriez. "A methodology for query reformulation in CIS using semantic knowledge". *Special issue on Formal Methods in Cooperative Information Systems*, Vol. 5(4), 1996.

[17] M. Fuchs, M. Maloney, and A. Milowski. "Schema for object-oriented XML". W3C Note NOTE-SOX-19980930. See `http://www.w3.org/TR/NOTE-SOX`.

[18] O. Lassila and R. R. Swick. "Resource Description Framework (RDF) model and syntax specification". W3C Working Draft WD-rdf-syntax-19981008. Available as `http://www.w3.org/TR/WD-rdf-syntax`.

[19] A. Layman, E. Jung, E. Maler, H. S. Thompson, J. Paoli, J. Tigue, N. H. Mikula, and S. De Rose. "XML-Data". W3C Note NOTE-XML-data-980105. See `http://www.w3.org/TR/1998/ NOTE-XML-data`.

[20] H. R. Lewis and C. H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1981.

[21] E. Maler and S. De Rose. "XML Linking language (XLink)". W3C Working Draft WD-xlink-19980303. See `http://www.w3.org/TR/WD-xlink`.

[22] L. Popa and V. Tannen. "An equational chase for path-conjunctive queries, constraints, and views". In *Proc. of 7th Int.'l Conf. on Database Theory (ICDT'99)*, 1999.

[23] W. C. Rounds. "Feature logics". In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*. Elsevier, 1997.