Interaction between path and type constraints in semistructured data

Interaction between Path and Type Constraints

PETER BUNEMAN
University of Edinburgh
and
WENFEI FAN
Bell Laboratories
and
SCOTT WEINSTEIN
University of Pennsylvania

Path constraints are capable of expressing inclusion and inverse relationships and have proved useful in modeling and querying semistructured data [Abiteboul and Vianu 1999; Buneman et al. 2000]. Types also constrain the structure of data and are commonly found in traditional databases. There has also been work on imposing structure or a type system on semistructured data for, e.g., storing and querying semistructured data in a traditional database system [Alon et al. 2001; Deutsch et al. 1999; Florescu and Kossmann 1999; Shanmugasundaram et al. 1999]. One wants to know whether complexity results for reasoning about path constraints established in the untyped (semistructured) context could carry over to traditional databases, and vice versa. It is therefore appropriate to understand the interaction between types and path constraints. In addition, XML [Bray et al. 1998], which may involve both an optional schema (e.g., DTDs or XML Schema [Thompson et al. 2001]) and integrity constraints, highlights the importance of the study of the interaction.

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. It shows that path constraints interact with types in a highly intricate way. For that purpose 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.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Types and structures; H.2.1 [**Database Management**]: Data models; F.4.3 [**Mathematical Logic and Formal Languages**]: Decision problems

General Terms: Algorithms, Design, Languages, Theory

Additional Key Words and Phrases: integrity constraints, types, semistructured data, implication

The paper is based on an extended abstract that appeared in *Proceedings of the 18th ACM Symposium on Principles of Database Systems* (Philadelphia, PA, May), 1999, pp. 56-67.
Email: peter@cis.upenn.edu, wenfei@research.bell-labs.com, weinstein@linc.cis.upenn.edu.
Wenfei Fan is on leave from Temple University, and is supported in part by NSF Career Award IIS-0093168.

1. INTRODUCTION

Integrity constraints have recently been studied as path constraints [Abiteboul and Vianu 1999; Buneman et al. 2000] for semistructured data. These constraints are capable of expressing a fundamental part of the semantics of the data such as inclusion and inverse relationships, and are valuable and effective in query optimization [Buneman et al. 1999; Deutsch et al. 1999], data integration [Florescu et al. 1996] and query translation [Lee and Chu 2000; Manolescu et al. 2001]. Semistructured data does not have a pre-imposed type system or schema, and is typically modeled as an edge-labeled graph [Abiteboul et al. 2000]. However there has also been a host of work on imposing structure or types on semistructured data for, among other things, storing and querying semistructured data using traditional database systems [Alon et al. 2001; Deutsch et al. 1999; Florescu and Kossmann 1999; Shanmugasundaram et al. 1999]. In addition, although the XML standard (eXtensible Markup Language [Bray et al. 1998]) itself does not require any schema or type system, a number of proposals [Davidson et al. 1999; Layman et al. 1998; Thompson et al. 2001] 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 support or advocate integrity constraints [Layman et al. 1998; Thompson et al. 2001; Buneman et al. 2001]. With these comes the need to understand the distinction between types and integrity constraints that is commonly made in traditional database systems, and to study their interaction.

Types and path constraints. It is worth examining types and integrity constraints specified in a traditional database schema. As an example, consider the following ODL [Cattell 2000] schema:

```
class Person                          class Book
   ( extent person)                      ( extent book)
   { attribute string ssn;               { attribute string title;
     attribute string name;                attribute set<Book> ref;
     relationship set<Book> wrote          relationship set<Person> author
              inverse Book::author;}                inverse Person::wrote;}
```

If we strike out the **extent** and **inverse** declarations, and change the **relationship** assertions to type declarations such as

```
attribute set<Book> wrote;          attribute set<Person> author;
```

then we are 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. These classes or types are an essential part of any program that constructs or queries data. Without them the program is meaningless. Contrast this with the situation in most XML query languages where no types are needed ("type" errors such as mis-spelled tag names show up not as static errors but as empty answers). In addition, this schema also defines integrity constraints: the **extent** and **inverse** declarations specify the following: (a) *Inclusion constraints.*

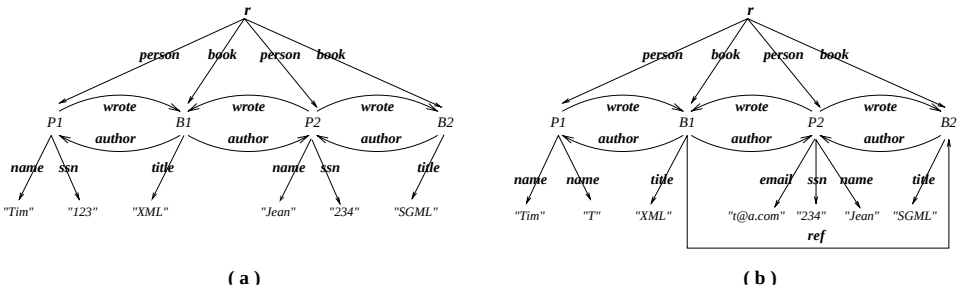

Fig. 1. Graph representation of an object-oriented database and semistructured data

For any person p, $p.wrote$ is included in the extent *book*. Similarly, for any book b, $b.author$ is a subset of the extent *person* and $b.ref$ is included in the extent *book*. (b) *Inverse constraints.* For any person p and any book b, if p wrote b then an author of b is p, and vice versa. Although such constraints cannot be expressed in any object-oriented system, they can be expressed in most schema definition languages, and – just as the static analysis of types is important for program correctness and efficiency – the static analysis of such constraints is important for query optimization [Deutsch et al. 1999; Florescu et al. 1996] and query translation [Lee and Chu 2000; Manolescu et al. 2001].

The distinction between types and integrity constraints appears to be dictated largely by what conventional programming languages treat as types. If we represent a database instance as a graph, then these are both constraints which restrict the structure of the graph interpretation of the data. In other words, in the world of semistructured data, both (traditional) types and (traditional) integrity constraints are constraints on the data.

To cast the problem concretely, Fig. 1 (a) represents an instance of the person/book database as a rooted, edge-labeled, directed graph. The database consists of two sets (extents), and we express this by a root node r with edges emanating from it that are labeled either `person` or `book`. These connect to nodes that respectively represent person and book objects which have edges emanating from them that respectively describe the structure of persons and books. These structures are constrained by different type constructs such as sets and records: each person must have a single `name` edge and a single `ssn` edge connected to a string node, and multiple `wrote` edges connected to book nodes. Similarly, each book must have a unique `title` edge connected to a string node, zero or more `ref` edges connected to book nodes and multiple `author` edges connected to person nodes. Contrast this with Fig. 1 (b), which represents semistructured data that is free of any type constraints. In particular, in Fig. 1 (b) a person may have multiple `name` edges, and optional `email` and `ssn` edges.

Integrity constraints on a graph can be expressed in terms of navigation paths, and are referred to as *path constraints*. For example, the inverse constraints can be written 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 a graph, variables x and y range over vertices, and the predicates denote edge 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 path from x to y. For example, $book \cdot author(r, x)$ is a path from root r to some node x in Fig. 1. The first constraint above asserts 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. They express an inverse relationship between `wrote` of `person` and `author` of `book`. Along the same lines, inclusion constraints can be expressed as:

$$\begin{aligned} \forall x\,(book \cdot author(r, x) &\rightarrow person(r, x)), \\ \forall x\,(person \cdot wrote(r, x) &\rightarrow book(r, x)), \\ \forall x\,(book \cdot ref(r, x) &\rightarrow book(r, x)). \end{aligned}$$

The first constraint states that an `author` of a `book` must be a `person`; similarly for `wrote` of `person` and `ref` of `book`. Observe that these path constraints are independent of any type system. In particular, they hold on both the object-oriented database (Fig. 1 (a)) and the semistructured data (Fig. 1 (b)). Like types, they also constrain the structure of the data, but in a very different fashion.

When it comes to XML, the story is more interesting. Like semistructured data, XML data does not require a type system or schema, but it may have an optional DTD (Document Type Definition). DTDs and other forms of specifications developed for XML, e.g., XML Schema [Thompson et al. 2001], XML Data [Layman et al. 1998], SOX [Davidson et al. 1999], can also be viewed as constraints. For example, Fig. 2 shows a schema described in XML Data, which specifies types as well as (limited) inclusion and inverse constraints (with the `range` and `correlative` declarations). If we treat `href` attributes as references, then an XML document can be represented as a graph (instead of a node-labeled tree as commonly assumed). More specifically, it may be represented either as the structure in Fig. 1 (a) or the one in Fig. 1 (b) depending on whether the schema is imposed on the data.

We should remark that path constraints may exist in untyped (semistructured) and typed (structured) contexts alike. Type constraints cannot be expressed as path constraints and *vice versa*.

Implication of path constraints. One of the most important technical problems associated with path constraints is the question of *implication:* given that certain path constraints Σ are known to hold, does it follow that some other path constraint φ is necessarily satisfied? Implication is important in, among other things, data integration. For example, one may want to know whether a constraint φ holds in a mediator interface. This cannot be verified directly since the mediator interface does not contain data. One way to verify φ is to show that it is implied by constraints that are known to hold [Florescu et al. 1996]. Other important applications of implication include query optimization [Buneman et al. 1999; Deutsch et al. 1999] and database design normalization [Ramakrishnan and Gehrke 2000].

The analysis of path constraint implication needs to be conducted in two different settings. In the untyped (semistructured) context, the question is to determine whether for any edge-labeled graph free of types, if it satisfies path constraints

```
<elementType  id = "person">
  <element type="#ssn"/>
  <element type="#name"/>
  <element type="#wrote"
           occurs="ONEORMORE"/>
</elementType>

<elementType  id = "wrote">
  <empty/>
  <domain type="#person"/>
  <correlative type="#author"/>
  <attribute name="href" dt="uri"
             range="#book"/>
</elementType>

<elementType  id = "ssn">
  <string/>
</elementType>

<elementType  id = "name">
  <string/>
</elementType>
```

```
<elementType  id = "book">
  <element type="#title"/>
  <element type="#ref"
           occurs="ONEORMORE"/>
  <element type="#author"
           occurs="ONEORMORE"/>
</elementType>

<elementType  id = "author">
  <empty/>
  <domain type="#book"/>
  <correlative type="#wrote"/>
  <attribute name="href" dt="uri"
             range="#person"/>
</elementType>

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

<elementType  id = "ref">
  <empty/>
<domain type="#book"/>
<attribute name="href" dt="uri"
           range="#book"/>
</elementType>
```

Fig. 2. A schema in XML Data

Σ then it must also satisfy path constraint φ. In the typed (structured) context, it is to determine whether for any edge-labeled graph that *satisfies certain type constraints*, if it satisfies Σ then it also satisfies φ. One wants to know whether complexity results for reasoning about path constraints established in the untyped context still hold in the typed context, and vice versa. This is important because on the one hand, one may want to impose structure on semistructured data, and on the other hand, one may want to create semistructured/XML views for traditional databases [Baru et al. 1999]. In particular, one needs to deal with XML data both in the untyped context (when a document does not come with a DTD or a schema) and in the typed context (when a DTD or a schema is imposed on a document).

Contribution. From recent work [Abiteboul and Vianu 1999; Buneman et al. 2000] on path constraints we have developed a reasonable understanding – in the context of semistructured (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 that the presence of types may actually *complicate* 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". More specifically,

—We treat types and path constraints both as constraints on the graph representation of data, and investigate the impact of the presence of types on path constraint implication. The type constraints are determined by a variety of type constructs found in traditional database systems, including records, sets, classes and recursive data structures.

—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.

That is, we show that that adding a type system may in some cases simplify the analysis of path constraint implication and in other cases make it harder. There are also practical interests in the implication problems studied.

Interaction - a logic retrospective. To understand why imposing a schema on the data can alter the computational complexity of the path constraint implication problem in unexpected ways, 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 (cf. [Börger et al. 1997]). Note that in this example, $\mathcal{S}'$ is not first order definable over $\mathcal{S}$.

We shall 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 5, we shall 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 6, 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 6, 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.

Related work. Integrity constraints for semistructured data were first studied as path constraints in [Abiteboul and Vianu 1999]. A path constraint of [Abiteboul and Vianu 1999] has the form $\forall\ x\ (p(r,\ x) \rightarrow q(r,\ x))$, where p, q are (regular) path expressions and r is a constant denoting the root of a graph. While these constraints could specify inclusions between paths, they were not expressive enough to capture, say, inverse constraints. Extensions were studied in [Buneman et al. 2000] to overcome this limitation. Let us refer to the constraint language of [Buneman et al. 2000] as P_c. In [Buneman et al. 2000], 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 is capable of expressing inclusion and inverse constraints. Since the constraint language P_c is not categorized as a quantifier prefix fragment of first-order logic, these results concerning the implication problems for P_c are orthogonal to classical work on the decision problem for fragments of first-order logic (cf. [Börger et al. 1997]). This paper extends [Buneman et al. 2000] by exploring the interaction between type systems and simple integrity constraints of P_c, whereas [Abiteboul and Vianu 1999; Buneman et al. 2000] do not consider the question of logical implication in the context of typed data.

There has also been recent work on the interaction between DTDs and integrity constraints for XML [Arenas et al. 2002; Fan and Libkin 2001], which is quite different from the work reported in this paper. First, the constraints studied there are simple keys and foreign keys, which cannot express path constraints (in particular, inclusion and inverse relationships) and vice versa. Second, the type constraints induced by DTDs are defined in terms of regular expressions, which are different from the type constructs (e.g., sets, records, classes) found in traditional database systems. Third, those papers consider XML data which is a special case of semistructured data and is modeled as a node-labeled tree, whereas this paper investigates general semistructured data modeled as an edge-labeled graph.

Path functional constraints have also been studied (see, e.g., [Ito and Weddell 1995; van Bommel and Weddell 1994]). These constraints are not capable of expressing inclusion and inverse constraints. Furthermore, they are studied in the context of traditional database systems.

Description logics (see, e.g., [Calvanese et al. 2001]) reason about concept sub-

sumption, which can express inclusion assertions similar to path constraints. There has been work on specifying constraints on semistructured data by means of description logics [Calvanese et al. 1998]. One of the most expressive description logics used in the database context is $\mathcal{ALCQI}_{reg}$ [Calvanese et al. 2001]. We should remark here that our path constraints are not expressible in $\mathcal{ALCQI}_{reg}$.

Organization. The remainder of the paper is organized as follows. Section 2 defines path constraints. Section 3 presents type constraints induced by various type constructs. Section 4 describes two implication problems for path constraints in the untyped and typed contexts. Section 5 investigates one of the implication problems and shows that it is undecidable in the context of semistructured data but the undecidability result breaks down when a simple type system is added. Section 6 demonstrates that adding a type system does not necessarily "help" in constraint implication problems. It establishes the decidability of another implication problem in the untyped context, and shows that the problem becomes undecidable when a generic type system is imposed. Section 7 presents some extensions of the results of the previous sections. Finally, Section 8 summarizes the main results of the paper.

2. PATH CONSTRAINTS

In this section we formally define path constraints.

A graph model. 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 is the root, and E^G is the set of labeled edges.

Semistructured data is characterized as having no pre-imposed type system or schema [Abiteboul et al. 2000], such as data commonly found on the World Wide Web, in biological databases and after data integration. Semistructured data is typically modeled as a rooted, edge-labeled, directed graph, and can be represented as a σ-structure. For example, the graph in Fig. 1 (b) can be viewed as such a structure. As opposed to semistructured data, structured data is constrained by a schema, such as data found for instance in relational and object-oriented databases. In the next section we shall see that structured data can also be represented as a first-order logic structure that satisfies certain type constraints.

Paths. To define path constraints, we first present the notions of paths.

A *path* is a sequence of edge labels. Formally, paths are defined by the syntax:

$$\rho \ ::= \ \epsilon \ \mid \ K \cdot \rho$$

Here ϵ is the empty path, $K \in E$, and $\cdot$ denotes path concatenation. A path can be expressed as a first-order logic formula $\rho(x, y)$ with two free variables x and y, which denote the tail and head nodes of the path, respectively:

—$\epsilon(x, y)$ is interpreted as $x = y$;

—$K \cdot \rho(x, y)$ is interpreted as $\exists z(K(x, z) \wedge \rho(z, y))$.

For example, $person \cdot wrote \cdot title(x, y)$ and $book \cdot ref \cdot ref(x, y)$ are paths.

We interpret paths as follows. Let G be a structure, $\rho(x, y)$ be a path formula and a, b be nodes in $|G|$. We use $G \models \rho(a, b)$ to denote that there is a path ρ from a to b in G. We say that b is *reachable from a by following ρ* if $G \models \rho(a, b)$. For example, referring to the structure G given in Fig. 1 (a) (or (b)), there is node a such that $G \models person \cdot wrote \cdot title(r, a)$.

A path ρ is said to be a *prefix* of ϱ, denoted by $\rho \preceq_p \varrho$, if there exists γ such that $\varrho = \rho \cdot \gamma$.

The *length* of path ρ, $|\rho|$, is defined as follows: $|\rho| = 0$ if $\rho = \epsilon$; $|\rho| = 1 + |\varrho|$ if ρ can be written as $K \cdot \varrho$, where $K \in E$. For example, $|person \cdot wrote \cdot title| = 3$.

Path constraints. We are now ready to define path constraints.

DEFINITION 2.1. *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, denoted by $pf(\varphi), lt(\varphi), rt(\varphi)$, respectively, and α is called the prefix *of φ. We use P_c to denote the set of all path constraints.*

A structure G *satisfies* φ, denoted by $G \models \varphi$, iff

—if φ is a forward constraint, then for any vertex x reachable from the root r by following α and for any vertex y reachable from x by following β, y is also reachable from x by following γ;

—if φ is a backward constraint, then 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.

Observe that P_c constraints are *relative* constraints: they are defined on sub-graphs (sub-structures) that are rooted at nodes reachable by following their prefix. In other words, forward and backward constraints expresses inclusion and inverse relationships relative to their prefix.

A proper subclass of P_c was introduced and investigated in [Abiteboul and Vianu 1999], namely *word constraints* of the form

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

where β and γ are paths. Observe that a word constraint is a forward constraint of P_c with its prefix being the empty path ϵ. For example, the inclusion constraints given in Section 1 are word constraints, whereas the inverse constraints are not. In contrast to general P_c constraints, P_w constraints are *absolute* constraints that hold on the entire graph (structure). We use P_w to denote the set of all word constraints.

For a set Σ of path constraints, we use $G \models \Sigma$ to denote that for any $\varphi \in \Sigma$, $G \models \varphi$.

Examples. In Section 1 we have seen that path constraints are capable of expressing inclusion and inverse relationships. Below we further demonstrate their expressive power by examples.

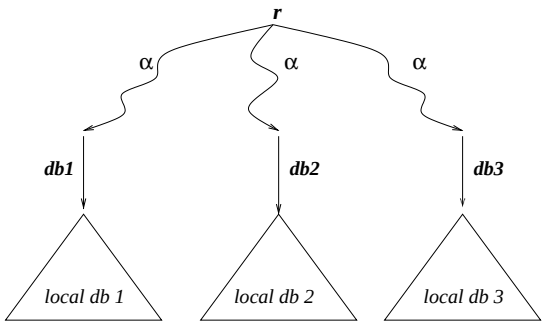

Fig. 3. Local databases in data integration

In data integration it is sometimes desirable to make one database a component of another database, or to build a "database of databases". For example, we want to bring together a number of person/book databases of the form depicted in Fig. 1, each consisting of a collection of person and book records, as shown in Fig. 3. One may want the constraints given in Section 1 to hold a component databases db_i. These can be expressed as path constraints:

$$\forall\, x\, (\alpha.db_i \cdot book(r,\, x) \rightarrow \forall\, y\, (author(x,\, y) \rightarrow wrote(y, x))),$$
$$\forall\, x\, (\alpha.db_i \cdot person(r,\, x) \rightarrow \forall\, y\, (wrote(x,\, y) \rightarrow author(y,\, x))),$$

$$\forall\, x\, (\alpha.db_i(r,\, x) \rightarrow \forall\, y\, (book \cdot author(x,\, y) \rightarrow person(x,\, y))),$$
$$\forall\, x\, (\alpha.db_i(r,\, x) \rightarrow \forall\, y\, (person \cdot wrote(x,\, y) \rightarrow book(x,\, y))),$$
$$\forall\, x\, (\alpha.db_i(r,\, x) \rightarrow \forall\, y\, (book \cdot ref(x,\, y) \rightarrow book(x,\, y))).$$

We refer to these as *local database constraints* as they are constraints on component databases.

Abiteboul and Vianu [1999] studied a generalization of word constraints defined in terms of regular path expressions. Along the same lines, P_c constraints can be generalized. In this paper we do not consider constraints of this general form because, as we shall see in Sections 5 and 6, P_c constraints suffice to explore the interaction between types and path constraints.

3. TYPE CONSTRAINTS

In this section we present two object-oriented models and examine type constraints induced by a variety of types constructs. In Sections 5 and 6, we shall investigate path constraint implication in the presence of these type constraints.

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

We first study a generic object-oriented model, $\mathcal{M}^+$. Similar to the models studied in [Abiteboul et al. 1995; Abiteboul and Kanellakis 1989; Cattell 2000], $\mathcal{M}^+$ supports classes, records, sets and recursive structures.

Schema and instances. We describe schema and instances of $\mathcal{M}^+$ as follows.

Assume a countable set $\mathcal{L}$ of labels, and a finite set $\mathcal{B}$ of *atomic types* (e.g., *int* and *string*). Let $\mathcal{C}$ be a finite set of *classes*. The set of *types over* $\mathcal{C}$, $Types^{\mathcal{C}}$, is

defined by:

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

where $b \in \mathcal{B}$, $C \in \mathcal{C}$, and $l_i \in \mathcal{L}$. The notations $\{.\}$ and $[.]$ 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 $\nu(C) \notin \mathcal{B} \cup \mathcal{C}$ for each class $C \in \mathcal{C}$, 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 $DBtype$ specifies the type of the entry point.

For example, the person/book database given in Fig. 1 (a) can be specified by $\Delta_0 = (\mathcal{C}, \nu, DBtype)$ in $\mathcal{M}^+$, where $\mathcal{C}$ consists of $Book$ and $Person$, ν maps $Book$ and $Person$ to record types:

$$\begin{aligned} Person &\mapsto [name : string, ssn : string, wrote : \{Book\}] \\ Book &\mapsto [title : string, ref : \{Book\}, author : \{Person\}] \end{aligned}$$

and $DBtype = [person : \{Person\}, book : \{Book\}]$.

A *database instance* of a schema $(\mathcal{C}, \nu, DBtype)$ is a triple (π, μ, 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

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

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.

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

Type constraints induced by $\mathcal{M}^+$. 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 as the abstraction of semistructured data, we represent a structured database as a first-order logic structure satisfying a certain type constraint.

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: (1) $DBtype \in T(\Delta)$ and $\mathcal{C} \subseteq T(\Delta)$; (2) 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 mem is in $E(\Delta)$;

—if $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or for some $C \in \mathcal{C}$, $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), 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 '*mem*' to denote the set membership relation.

The *signature determined by a 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 Δ_0 given above is (r, E, T), where r is a constant, which in each instance (π, μ, d) of the schema intends to name d; E includes *person*, *book*, *name*, *ssn*, *wrote*, *title*, *ref*, *author* and *mem*; and T includes $Person$, $Book$, *string*, $\{Book\}$, $\{Person\}$ and $DBtype$.

We represent an instance I of a schema Δ as a (finite) $\sigma(\Delta)$-structure G that satisfies a certain type constraint. More specifically, given $\Delta = (\mathcal{C}, \nu, DBtype)$ and $I = (\pi, \mu, d)$, there is a $\sigma(\Delta)$-structure $G = (|G|, r^G, E^G, T^G)$ such that $|G|$, r^G, E^G and T^G 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, denoted by $\Phi(\Delta)$, which specifies restrictions on the edges going out of vertices of different types. More specifically, for each τ in $T(\Delta)$, let the first order logic sentence $\forall x\,(\tau(x) \rightarrow \phi_\tau(x))$ be the *constraint determined by* τ, then

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

That is, every element of $|G|$ has a unique type τ in $T(\Delta)$ and it must satisfy the constraint imposed by τ. In particular, r^G has $DBtype$. We define ϕ_τ as follows. Let a be an element of $|G|$ with type τ. Then a must satisfy $\phi_\tau(x)$ defined by:

—If τ is an atomic type b, then a has no outgoing edge. That is,

$$\phi_\tau(x) \;=\; \forall y\,(\bigwedge_{l \in E(\Delta)} \neg l(x, y)).$$

—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 *mem* and they lead to elements of type τ'. That is,

$$\phi_\tau(x) \;=\; \forall y\,(\bigwedge_{l \in E(\Delta)\setminus\{mem\}} \neg l(x, y)) \;\wedge \forall y\,(mem(x, y) \rightarrow \tau'(y)).$$

In addition, if $\tau = \{\tau'\}$, then for each $b \in |G|$ such that b also has type τ, $a = b$ iff for any $c \in |G|$, $G \models mem(a, c) \leftrightarrow mem(b, c)$. In other words, two sets are equal iff they have the same elements. That is, $\phi_\tau(x)$ also has the conjunct:

$$\forall y\,((\tau(y) \wedge \forall z\,(mem(x, z) \leftrightarrow mem(y, z))) \;\rightarrow\; x = y).$$

—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

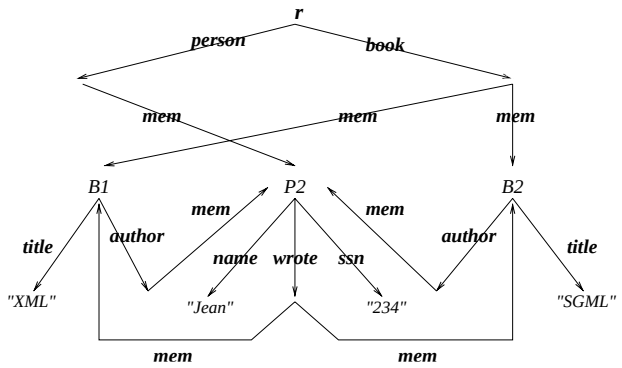

Fig. 4. Graph representation of an abstract database in $\mathcal{M}^+$

o has type τ_i. That is,

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

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 other words, two records are equal iff they have the same attributes. That is, $\phi_\tau(x)$ also has the conjunct:

$$\forall y\,((\tau(y) \wedge \bigwedge_{i\in[1,n]} \forall z\,(l_i(x,\,z) \leftrightarrow l_i(y,\,z))) \;\rightarrow\; x = y).$$

Here to simplify the description, we use the counting quantifier $\exists!$, whose semantics is as follows: structure G satisfies $\exists!x\,\psi(x)$ if and only if there exists a unique element a of G such that $G \models \psi(a)$ (see, e.g., [Börger et al. 1997] for detailed discussions of counting quantifiers). It should be noted that $\exists!$ is definable in first-order logic.

An *abstract database of a schema* Δ is a finite $\sigma(\Delta)$-structure G satisfying $\Phi(\Delta)$, i.e., $G \models \Phi(\Delta)$. 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 $\Phi(\Delta)$.

An abstract database Δ can also be depicted as an edge-labeled, rooted, directed graph, which has a certain "shape" specified by the type constraint $\Phi(\Delta)$. For example, a $\sigma(\Delta_0)$-structure representing person/book database is depicted in Fig. 4. Contrast this with Fig. 1 (a). These two graphs are slightly different because of the explicit treatment of the set membership relation "*mem*". However, there is a straightforward transformation from the former to the latter: if we replace $person \cdot mem$ with $person$, $book \cdot mem$ with $book$, $wrote \cdot mem$ with $wrote$ and $author \cdot mem$ with $author$ in Fig. 4, then we obtain a graph representation in the same form as the one shown in Fig. 1 (a).

Path constraints revisited. Next, we refine the definitions of paths and path constraints in the context of $\mathcal{M}^+$, and justify the abstraction of databases given above with respect to path constraint satisfiability.

Given the signature $(r,\, E(\Delta),\, T(\Delta))$ determined by a schema Δ, one could define

paths and path constraints using binary predicates in $E(\Delta)$ in the same way as in Section 2. These definitions, however, are somewhat too coarse in the context of the object-oriented model. Because of the type constraint $\Phi(\Delta)$, some paths are not meaningful in structures of $\mathcal{U}(\Delta)$. That is, there exists path $\alpha(x, y)$ defined in this way such that for all $G \in \mathcal{U}(\Delta)$ and all $a, b \in |G|$, $G \not\models \alpha(a, b)$. Such paths are said to be *undefined over* Δ. A path constraint containing undefined paths is satisfied by either all the structures in $\mathcal{U}(\Delta)$ or by none of them. We are not interested in such constraints.

We use $Paths(\Delta)$ to denote the set of *paths over* Δ. Intuitively, $\alpha \in Paths(\Delta)$ iff there is $G \in \mathcal{U}(\Delta)$ such that $G \models \exists\, x\ \alpha(r, x)$. We define P_c constraints over a schema Δ in terms of paths in $Paths(\Delta)$. Formally, these notions are defined as follows.

Let an $\mathcal{M}^+$ schema Δ be $(\mathcal{C}, \nu, DBtype)$. The set of *paths over schema* 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 $\tau = \{\tau'\}$ or for some $C \in \mathcal{C}$, $\tau = C$ and $\nu(C) = \{\tau'\}$, then $\alpha \cdot mem$ is a path in $Paths(\Delta)$ and $type(\alpha \cdot mem) = \tau'$;

—if $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ or there exists a class C in $\mathcal{C}$ such that $\tau = C$ and $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$, then for each $i \in [1, n]$, $\alpha \cdot l_i$ is in $Paths(\Delta)$ and $type(\alpha \cdot l_i) = \tau_i$.

A P_c *constraint* φ *over schema* Δ 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

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

such that (1) if φ is of the forward form, then $\alpha \cdot \beta \in Paths(\Delta)$, $\alpha \cdot \gamma \in Paths(\Delta)$ and moreover, $type(\alpha \cdot \beta) = type(\alpha \cdot \gamma)$; (2) if φ is of the backward form, then $\alpha \in Paths(\Delta)$, $\alpha \cdot \beta \cdot \gamma \in Paths(\Delta)$ and moreover, $type(\alpha) = type(\alpha \cdot \beta \cdot \gamma)$. In particular, a *word constraint* over Δ is of the form $\forall\, x\, (\beta(r, x) \rightarrow \gamma(r, x))$, where β and γ are in $Paths(\Delta)$ such that $type(\beta) = type(\gamma)$. We denote the set of all P_c and word constraints over Δ by $P_c(\Delta)$ and $P_w(\Delta)$, respectively. When Δ is understood from the context, we write $P_c(\Delta)$ and $P_w(\Delta)$ simply as P_c and P_w, respectively.

Observe that these syntactic restrictions can be checked in quadratic time in the sizes of path constraint and schema. These restrictions ensure that path constraints make sense in structures of $\mathcal{U}(\Delta)$. Without the restriction $\alpha \cdot \beta \cdot \gamma \in Paths(\Delta)$ on a backward constraint, for example, the constraint is either always true or always false in all structures of $\mathcal{U}(\Delta)$, depending on whether $\alpha \cdot \beta \in Paths(\Delta)$ or not.

For example, the following are P_c constraints over the schema Δ_0 given earlier:

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

Note that these constraints are presented here in a slightly different way from Section 1 because of the explicit treatment of the set membership relation.

In an instance (π, μ, d) of Δ_0, these constraints are interpreted as:

$$\forall x\,(\exists p\,(p \in d.person \wedge x \in p.wrote) \rightarrow x \in d.book),$$
$$\forall x\,(\exists b\,(b \in d.book \wedge x \in b.author) \rightarrow x \in d.person),$$

$$\forall x\,(x \in d.person \rightarrow \forall y\,(y \in x.wrote \rightarrow x \in y.author)),$$
$$\forall x\,(x \in d.book \rightarrow \forall y\,(y \in x.author \rightarrow x \in y.wrote)).$$

Here $v.l$ is the projection of record v on attribute l, and $v \in s$ means that v is an element of set s.

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

The lemma below justifies the abstraction of structured databases defined above. It reveals the agreement between databases and their abstraction with respect to path constraints.

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

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

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

Proof: Let $\Delta = (\mathcal{C}, \nu, DBtype)$.
(1) Given $I \in \mathcal{I}(\Delta)$, we construct $G \in \mathcal{U}_f(\Delta)$ such that for each $\varphi \in P_c(\Delta)$, $I \models \varphi$ iff $G \models \varphi$.

Let $I = (\pi, \mu, d)$. We define V to be the smallest set such that $d \in V$ and for every $v \in V$, (a) if v is a set (or v is an object and $\mu(v)$ is a set), then every element of v (or $\mu(v)$) is in V; (b) if v is a record (or v is an object and $\mu(v)$ is a record), then every attribute of v (or $\mu(v)$) is in V.

For any $v \in V$, let $o(v)$ be a distinct node. Let $G = (|G|, r^G, E^G, T^G)$, where (a) $|G| = \{o(v) \mid v \in V\}$; (b) $r^G = o(d)$; (c) for each $o(v) \in |G|$ and $\tau \in \mathcal{T}(\Delta)$, $G \models \tau^G(o(v))$ iff v is of type τ, where τ^G denotes the unary relation in G named by τ; and (d) for all $o(v), o(v') \in |G|$,

—for each $l \in \mathcal{L} \cap E(\Delta)$, $G \models l(o(v), o(v'))$ iff $v' = v.l$ (or $v' = \mu(v).l$ if v is an object);

—$G \models mem(o(v), o(v'))$ iff $v' \in v$ (or $v' \in \mu(v)$ if v is an object).

It is straightforward to verify by *reductio* that $G \in \mathcal{U}_f(\Delta)$ and for each $\varphi \in P_c(\Delta)$, $G \models \varphi$ iff $I \models \varphi$.

(2) Given $G = (|G|, r^G, E^G, T^G)$ in $\mathcal{U}_f(\Delta)$, we define $I = (\pi, \mu, d)$ in $\mathcal{I}(\Delta)$ such that for every $\varphi \in P_c(\Delta)$, $I \models \varphi$ iff $G \models \varphi$. To simplify the discussion, we assume that for every base type b, its domain D_b is infinite. By this assumption, there exists an injective mapping $g_b : b^G \rightarrow D_b$, where b^G is the unary relation in G denoting the sort b.

For any $C \in \mathcal{C}$, let $\pi(C) = C^G$, where C^G is the unary relation in G denoting the class C. We then define a mapping $f : |G| \to \bigcup_{\tau \in T(\Delta)} [\![\tau]\!]_\pi$ that instantiates the objects as follows: For each $o \in |G|$, (a) if $o \in C^G$ for some $C \in \mathcal{C}$, then let $f(o) = o$; (b) if $o \in b^G$ for some base type b, then let $f(o) = g(o)$; (c) if $o \in \tau^G$ and $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$, then let $f(o) = [l_1 : f(o_1), \ldots, l_n : f(o_n)]$, where for each $i \in [1, n]$, $o_i \in |G|$ and $G \models l_i(o, o_i)$; (d) if $o \in \tau^G$ and $\tau = \{\tau'\}$, then let $f(o) = \{f(o') \mid o' \in |G|,\ G \models mem(o, o')\}$. Note that f is well-defined since G is finite and $G \models \Phi(\Delta)$. Now let $d = f(r^G)$ and for each $C \in \mathcal{C}$ and $o \in \pi(C)$,

—if $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$, then let $\mu(o) = [l_1 : f(o_1), \ldots, l_n : f(o_n)]$, where for $i \in [1, n]$, $o_i \in |G|$ and $G \models l_i(o, o_i)$.

—if $\nu(C) = \{\tau\}$, then let $\mu(o) = \{f(o') \mid o' \in |G|,\ G \models mem(o, o')\}$.

Again, this is well-defined. Moreover, it is easy to verify that $I \in \mathcal{I}(\Delta)$, and $G \models \varphi$ iff $I \models \varphi$. □

From Lemma 3.1 follows immediately the corollary below.

COROLLARY 3.2. *Let Δ be any schema in $\mathcal{M}^+$ and $\Sigma \cup \{\varphi\}$ be any finite subset of $P_c(\Delta)$. Then there is $I \in \mathcal{I}(\Delta)$ such that $I \models \bigwedge \Sigma \wedge \neg\varphi$ if and only if there is $G \in \mathcal{U}_f(\Delta)$ such that $G \models \bigwedge \Sigma \wedge \neg\varphi$.*

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

We also consider a restriction $\mathcal{M}$ of $\mathcal{M}^+$. 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 $l_i \in \mathcal{L}$.

Schema and instances in $\mathcal{M}$ are defined in the same way as in $\mathcal{M}^+$. So are the notions of $E(\Delta)$, $T(\Delta)$, $\sigma(\Delta)$, $\Phi(\Delta)$, $\mathcal{U}_f(\Delta)$, $\mathcal{U}(\Delta)$, and $P_c(\Delta)$ for a schema Δ in $\mathcal{M}$. It is easy to verify that Lemma 3.1 also holds in the context of $\mathcal{M}$. Databases of $\mathcal{M}$ are comparable to feature structures [Rounds 1997], which have proven useful in representing linguistic data.

4. IMPLICATION PROBLEMS FOR PATH CONSTRAINTS

In this section we describe two implication problems associated with path constraints, which will be investigated in the rest of the paper.

4.1 Implication of P_c constraints

We first present implication and finite implication of P_c constraints. Let $\Sigma \cup \{\varphi\}$ be a finite set of P_c constraints.

In the untyped (semistructured) context, we use $\Sigma \models \varphi$ to denote that Σ *implies* φ, i.e., for any σ-structure G, if $G \models \Sigma$ then $G \models \varphi$. We use $\Sigma \models_f \varphi$ to denote that Σ *finitely implies* φ. That is, for any finite σ-structure G, if $G \models \Sigma$ then $G \models \varphi$.

The (*unrestricted*) *implication problem for P_c in the untyped context* is the problem to determine, given any finite set Σ and φ of P_c constraints, whether $\Sigma \models \varphi$ or not. Similarly, the *finite implication problem for P_c* to determine whether $\Sigma \models_f \varphi$.

For example, let Σ_0 denote the P_c constraints given in Section 1 and φ_0 be

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

The question whether every (finite) model of Σ_0 also satisfies φ_0 is an instance of the (finite) implication problem for P_c. It is easy to verify that $\Sigma_0 \models \varphi_0$ and $\Sigma_0 \models_f \varphi_0$ indeed hold.

In the typed context, path constraint implication is restricted by a schema. More specifically, let Δ be a schema in $\mathcal{M}^+$ (or $\mathcal{M}$). 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 $\mathcal{M}^+$ (resp. $\mathcal{M}$), the unrestricted (finite) implication problem for P_c is the problem of determining, given any finite set Σ and φ of P_c constraints and any schema Δ in $\mathcal{M}^+$ (resp. $\mathcal{M}$), whether $\Sigma \models_\Delta \varphi$ ($\Sigma \models_{(f,\,\Delta)} \varphi$).

As mentioned in Section 1, P_c constraint implication is important in, among other things, query optimization and data integration.

4.2 Implication of local database constraints

As observed in Section 2, local database constraints are useful in data integration. When represented in a global database, constraints on a local database are usually augmented with a common prefix which is a path from the root of the global database to the component database. For example, referring to Fig. 3, inclusion constraints on local database 1 have the form:

$$\forall\, x\, (\alpha \cdot db1(r,\, x) \rightarrow \forall\, y\, (book \cdot author(x,\, y) \rightarrow person(x,\, y))),$$
$$\forall\, x\, (\alpha \cdot db1(r,\, x) \rightarrow \forall\, y\, (person \cdot wrote(x,\, y) \rightarrow book(x,\, y))),$$
$$\forall\, x\, (\alpha \cdot db1(r,\, x) \rightarrow \forall\, y\, (book \cdot ref(x,\, y) \rightarrow book(x,\, y))).$$

One might be interested in implication of inclusion constraints on a particular local database, e.g., whether the constraints above imply the following constraint on database 1:

$$\forall\, x\, (\alpha \cdot db1(r,\, x) \rightarrow \forall\, y\, (book \cdot ref \cdot ref(x,\, y) \rightarrow book(x,\, y))).$$

Furthermore, one wants to know whether the implication would be affected if constraints on other local databases are present, such as inverse constraints on local database 2:

$$\forall\, x\, (\alpha \cdot db2 \cdot book(r,\, x) \rightarrow \forall\, y\, (author(x,\, y) \rightarrow wrote(y, x))),$$
$$\forall\, x\, (\alpha \cdot db2 \cdot person(r,\, x) \rightarrow \forall\, y\, (wrote(x,\, y) \rightarrow author(y,\, x))).$$

To formalize this implication problem, we use the following notations.

DEFINITION 4.1. *A set Σ_K of P_c constraints is said to be* bounded by a path α and a label K *if each φ in Σ_K has 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 set Σ_r of P_c constraints is said to be excluded by α and K *if for each $\phi \in \Sigma_r$, $pf(\phi) = \alpha \cdot \alpha'$ for some path α' and $K \not\preceq_p \alpha'$; in particular, when $\alpha' = \epsilon$, β does not contain any label except K. Here $pf(\phi)$ denotes the prefix of ϕ, as described in Definition 2.1.*

Intuitively, Σ_K consists of local inclusion constraints on a local database connected to the root by $\alpha \cdot K$, and Σ_r includes constraints that are not defined on that particular local database. A special case is $\alpha' = \epsilon$, when the constraint is not defined on any local database. To simplify our proofs (to avoid dealing with various labels in our constructions), in this case we restrict β to be either ϵ or a path of K labels.

In the untyped context, the *unrestricted (finite) implication problem for local inclusion constraints* is to determine, given any finite sets Σ_K, Σ_r and φ of P_c constraints, whether $\Sigma_K \cup \Sigma_r \models \varphi$ ($\Sigma_K \cup \Sigma_r \models_f \varphi$), where $\Sigma_K \cup \{\varphi\}$ is bounded by a path α and a label K, and Σ_r is excluded by α and K. Similarly, the unrestricted (finite) implication problem for local inclusion constraints is defined in the context of $\mathcal{M}^+$ (resp. $\mathcal{M}$).

4.3 A road map

We shall use the (finite) implication problems described above to demonstrate the interaction between path and type constraints. More specifically, in the next section we shall study implication and finite implication of P_c constraints: we show that these problems are undecidable in the context of semistructured data (Theorem 5.1), and that they become decidable in the context of the data model $\mathcal{M}$ (Theorem 5.2). In fact we show the undecidability result also holds for a small fragment of P_c, denoted by $P_w(\alpha)$ (Lemma 5.3). These tell us that the presence of a type system in some cases may simplify the analysis of path constraint implication. In Section 6 we shall investigate implication and finite implication of local inclusion constraints: we show that although these problems are decidable in PTIME in the untyped setting (Theorem 6.1), they become undecidable in the context of the model $\mathcal{M}^+$ (Theorem 6.2). These demonstrate that in other cases adding a type system may make the implication analysis more intriguing.

To give a complete picture of path constraint implication in different settings, in Section 7 we establish the undecidability of the (finite) implication problems for the fragment $P_w(\alpha)$ in the context of $\mathcal{M}^+$ (Theorem 7.1). Furthermore, we study a restriction of the model $\mathcal{M}^+$, denoted by $\mathcal{M}^*$, which only allows finite sets instead of general (possibly infinite) sets. Although we show that this slightly different semantics does not change the undecidability of the (finite) implication problems for P_c, $P_w(\alpha)$ and for local inclusion constraints (Theorem 7.2), it makes unrestricted implication and finite implication of these constraints equivalent (Lemma 7.3). We also compare unrestricted implication and finite implication in other settings (see Corollaries 5.7, 7.4, 7.5 and 7.6).

5. SIMPLIFICATION OF PATH CONSTRAINT IMPLICATION WITH TYPES

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. The main results of the section are the following:

THEOREM 5.1. *In the context of semistructured data, the implication and finite implication problems for P_c are undecidable.*

THEOREM 5.2. *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.

We prove Theorems 5.1 and 5.2 in Sections 5.1 and 5.2, respectively.

5.1 Undecidability on untyped data

Theorem 5.1 was first shown in [Buneman et al. 2000]. Here we strengthen the result by identifying an undecidable fragment of P_c. This "small" fragment of P_c is a mild generalization of P_w, the class of word constraints introduced by Abiteboul and Vianu [1999] and described in Section 2.

We present the fragment as follows. Recall E, the finite set of binary relation symbols (edge labels) in the 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

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

That is, $P_w(K)$ consists of P_w constraints and extensions of P_w constraints with a common prefix K.

In the context of semistructured data, the (finite) implication problem for $P_w(K)$ is the problem to determine, given any finite set Σ and φ of $P_w(K)$ constraints, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$). The lemma below establishes the undecidability of these problems, from which Theorem 5.1 follows immediately.

LEMMA 5.3. *In the context of semistructured data, both the implication and finite implication problems for $P_w(K)$ are undecidable.*

Contrast this with [Abiteboul and Vianu 1999], which shows that the implication and finite implication problems for P_w are decidable in PTIME. Lemma 5.3 tells us that the problems become undecidable when P_w is generalized in such a mild way. We thank an anonymous referee for observing a similarity between $P_w(K)$ constraint implication and subsumption in a description logic extended with a role-value-map construct: in particular, the subsumption problem was proved undecidable by reduction from the word problem for groups [Schmidt-Schauß 1989].

We prove Lemma 5.3 by reduction from the word problem for (finite) monoids. Before we give the reduction, we first review the word problem for (finite) monoids.

The word problem for (finite) monoids. A *monoid* is a triple $(M,\, \circ,\, 1)$, where M is a nonempty set, $\circ$ is an associative binary relation on M, and 1 is an element of M that is the identity for $\circ$. That is, for any $a \in M$, $1 \circ a = a = a \circ 1$. A monoid $(M,\, \circ,\, 1)$ is said to be finite if M is finite.

Let Γ be a finite alphabet. The *free monoid generated by* Γ is $(\Gamma^*,\, \cdot,\, \epsilon)$, where Γ^* is the set of all finite strings with letters in Γ, '$\cdot$' is the concatenation operator on strings, and ϵ is the empty string.

An *equation* (over Γ) is a pair $(\alpha,\ \beta)$ of strings in Γ^*. Let $\Theta = \{(\alpha_i, \beta_i) \mid i \in [1,n]\}$ be a finite set of equations, and $\theta = (\alpha,\ \beta)$ be a *test equation*. Then $\Theta \models \theta$ ($\Theta \models_f \theta$) iff for every (finite) monoid $(M,\ \circ,\ 1)$ and every homomorphism $h : \Gamma^* \to M$, 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 to determine, given Θ and θ, whether $\Theta \models \theta$ ($\Theta \models_f \theta$).

The following result is well-known (cf. [Abiteboul et al. 1995]).

LEMMA 5.4. *The word problem for monoids and the word problem for finite monoids are undecidable.*

Reduction from the word problem. We reduce the word problem for (finite) monoids to 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

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

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:

$$\forall x\,(\epsilon(r,x) \to K(r,x)),$$

for every $j \in [1,m]$,

$$\forall x\,(K \cdot l_j(r,x) \to K(r,x)),$$

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

$$\forall x\,(K(r,x) \to \forall y\,(\alpha_i(x,y) \to \beta_i(x,y))), \quad \forall x\,(K(r,x) \to \forall y\,(\beta_i(x,y) \to \alpha_i(x,y))).$$

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

$$\varphi_{(\alpha,\beta)} = \forall x\,(\alpha(r,x) \to \beta(r,x)), \qquad \varphi_{(\beta,\alpha)} = \forall x\,(\beta(r,x) \to \alpha(r,x)).$$

The lemma below shows that the encoding above is indeed a reduction from the word problem for (finite) monoids. From this and Lemma 5.4 follows Lemma 5.3.

LEMMA 5.5. *In the context of semistructured data, for all* $\alpha, \beta \in \Gamma_0^*$,

$$\Theta_0 \models (\alpha,\ \beta) \ \textit{iff}\ \ \Sigma \models \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}, \qquad \Theta_0 \models_f (\alpha,\ \beta) \ \textit{iff}\ \ \Sigma \models_f \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}.$$

Proof: We prove the lemma for finite implication only. The proof for implication is similar.

(*if*) Suppose $\Theta_0 \not\models_f (\alpha,\ \beta)$. We show $\Sigma \not\models_f \forall x\,(\alpha(r,x) \to \beta(r,x))$ by constructing a finite σ_0-structure G such that $G \models \Sigma$ but $G \not\models \forall x\,(\alpha(r,x) \to \beta(r,x))$. Specifically, by $\Theta_0 \not\models_f (\alpha,\ \beta)$, there exist a finite monoid $(M,\ \circ,\ 1)$ and a homomorphism $h : \Gamma_0^* \to M$ such that for any $i \in [1,n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. We are to construct G based on M and h such that strings α, β are coded with paths preserving their relationship.

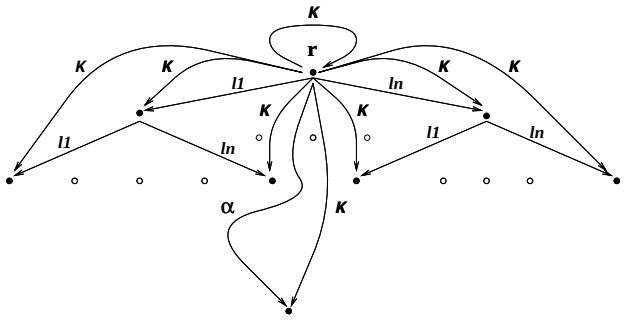

Fig. 5. The structure G in the proof of Lemma 5.5

To do this, we first define some notations. Based on M and h, we define an equivalence relation $\approx$ on Γ_0^* as follows:

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

For every $\rho \in \Gamma_0^*$, let $\widehat{\rho}$ be the equivalence class of ρ with respect to $\approx$. Let $C_{\Theta_0} = \{\widehat{\rho} \mid \rho \in \Gamma_0^*\}$.

Using these notations, we construct a structure $G = (|G|, r^G, E^G)$ as shown in Fig. 5, as follows. (1) For each $\widehat{\rho} \in C_{\Theta_0}$, let $o(\widehat{\rho})$ be a distinct node. Then we define $|G| = \{o(\widehat{\rho}) \mid \widehat{\rho} \in C_{\Theta_0}\}$. (2) $r^G = o(\widehat{\epsilon})$. (3) The binary relations are populated as follows: For each $\widehat{\rho} \in C_{\Theta_0}$, let $G \models K(o(\widehat{\epsilon}), o(\widehat{\rho}))$. In addition, for each $j \in [1, m]$, let $G \models l_j(o(\widehat{\rho}), o(\widehat{\rho \cdot l_j}))$.

By the construction of G, it is easy to see that for every $\rho \in \Gamma_0^*$ and $j \in [1, m]$, $o(\widehat{\rho \cdot l_j})$ is the unique node such that $G \models l_j(o(\widehat{\rho}), o(\widehat{\rho \cdot l_j}))$. This is because h is a homomorphism, and as a result, if $\rho_1 \approx \rho_2$, then $h(\rho_1 \cdot l_j) = h(\rho_1) \circ h(l_j) = h(\rho_2) \circ h(l_j) = h(\rho_2 \cdot l_j)$.

Using this property of G, it is also easy to verify the following claims.

Claim 1: G is finite.

To prove this, it suffices to show that C_{Θ_0} is finite. Specifically, consider function $f : C_{\Theta_0} \to M$ defined by $f : \widehat{\rho} \mapsto h(\rho)$. Clearly, f is well-defined, total and injective. Thus because M is finite, so is C_{Θ_0}.

Claim 2: $G \models \Sigma$.

Observe that $G \models \forall\, x\, (\epsilon(r, x) \to K(r, x))$ and $G \models \forall\, x\, (K \cdot l_j(r, x) \to K(r, x))$ for each $j \in [1, m]$ are immediate from the construction of G. To see that for each $(\alpha_i, \beta_i) \in \Theta_0$, $G \models \forall\ x\ (K(r, x) \to \forall\, y\, (\alpha_i(x, y) \to \beta_i(x, y)))$ and $G \models \forall\ x\ (K(r, x) \to \forall\, y\ (\beta_i(x, y) \to \alpha_i(x, y)))$, observe the following. First, by assumption, $\alpha_i \approx \beta_i$ for any $i \in [1, n]$. In addition, for every $\rho \in \Gamma_0^*$, $h(\rho \cdot \alpha_i) = h(\rho \cdot \beta_i)$ because h is a homomorphism. Therefore, $\rho \cdot \alpha_i \approx \rho \cdot \beta_i$. That is, $\widehat{\rho \cdot \alpha_i} = \widehat{\rho \cdot \beta_i}$. Second, by the construction of G, for any $o \in |G|$, $o = o(\widehat{\rho})$ for some $\rho \in \Gamma_0^*$. Moreover, for each $\varrho \in \Gamma_0^*$, it can be shown by a straightforward induction on $|\varrho|$ that there is a unique $o' \in |G|$ such that $G \models \varrho(o(\widehat{\rho}), o')$ and $o' = o(\widehat{\rho \cdot \varrho})$. Thus for each $o(\widehat{\rho})$ such that $G \models K(o(\widehat{\epsilon}), o(\widehat{\rho}))$, $o(\widehat{\rho \cdot \alpha_i})$ is the unique node in $|G|$ such that $G \models \alpha_i(o(\widehat{\rho}), o(\widehat{\rho \cdot \alpha_i}))$. Similarly, we have $G \models \beta_i(o(\widehat{\rho}), o(\widehat{\rho \cdot \beta_i}))$. By $o(\widehat{\rho \cdot \alpha_i}) = o(\widehat{\rho \cdot \beta_i})$, we have $G \models \beta_i(o(\widehat{\rho}), o(\widehat{\rho \cdot \alpha_i}))$. Therefore, for each $i \in [1, n]$,

$G \models \forall x\,(K(r,x) \rightarrow \forall y\,(\alpha_i(x,y) \rightarrow \beta_i(x,y)))$. Similarly, it can be shown that $G \models \forall x\,(K(r,x) \rightarrow \forall y\,(\beta_i(x,y) \rightarrow \alpha_i(x,y)))$. Therefore, $G \models \Sigma$.
Claim 3: $G \not\models \forall x\,(\alpha(r,x) \rightarrow \beta(r,x))$.

As in Claim 2, we can show $G \models \alpha(o(\widehat{\epsilon}),\, o(\widehat{\alpha}))$ and $G \models \beta(o(\widehat{\epsilon}),\, o(\widehat{\beta}))$. In addition, $o(\widehat{\beta})$ is the unique node in $|G|$ such that $G \models \beta(o(\widehat{\epsilon}),\, o(\widehat{\beta}))$. By assumption, we have $\alpha \not\approx \beta$. Thus $\widehat{\alpha} \neq \widehat{\beta}$ and $o(\widehat{\alpha}) \neq o(\widehat{\beta})$. From this follows $G \models \alpha(o(\widehat{\epsilon}),\, o(\widehat{\alpha})) \wedge \neg\beta(o(\widehat{\epsilon}),\, o(\widehat{\alpha}))$. That is, $G \not\models \forall x\,(\alpha(r,x) \rightarrow \beta(r,x))$.

(*only if*) Suppose $\Sigma \not\models_f \forall x\,(\alpha(r,x) \rightarrow \beta(r,x)) \wedge \forall x\,(\beta(r,x) \rightarrow \alpha(r,x))$. We show $\Theta_0 \not\models_f (\alpha, \beta)$ by defining a finite monoid $(M, \circ, 1)$ and a homomorphism $h : \Gamma_0^* \rightarrow M$ such that for any $i \in [1,n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. More specifically, by assumption, there exists a finite σ_0-structure G such that $G \models \Sigma$ but $G \not\models \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}$. We define the monoid and homomorphism such that paths α, β in G are coded with strings preserving the same relationship.

To do this, we define another equivalence relation on Γ_0^*. Without loss of generality, assume that there is $o \in |G|$ such that $G \models \alpha(r^G,\, o) \wedge \neg\beta(r^G, o)$. Based on G, we define an equivalence relation $\sim$ on Γ_0^*:

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

Then by $G \models \Sigma$, for any $i \in [1,n]$, $\alpha_i \sim \beta_i$. By $G \models \forall x\,(\epsilon(r,x) \rightarrow K(r,x))$, we have $G \models K(r^G,\, r^G)$. In addition, by $G \models \alpha(r^G,\, o) \wedge \neg\beta(r^G, o)$, we have $G \not\models \forall x\,(K(r,x) \rightarrow \forall y\,(\alpha(x,y) \rightarrow \beta(x,y)))$. Therefore, $\alpha \not\sim \beta$. For every $\rho \in \Gamma_0^*$, let $[\rho]$ denote the equivalence class of ρ with respect to $\sim$. Then clearly, for any $i \in [1,n]$, $[\alpha_i] = [\beta_i]$, but $[\alpha] \neq [\beta]$.

Using the notion of $\sim$, we define $M = \{[\rho] \mid \rho \in \Gamma_0^*\}$. We have the following about M.
Claim 4: M is finite.

To show this, we define $S_\rho = \{(a,b) \mid a,b \in |G|,\ G \models K(r^G,\, a) \wedge \rho(a,\, b)\}$ for every $\rho \in \Gamma_0^*$. In addition, let $S_G = \{S_\rho \mid \rho \in \Gamma_0^*\}$. Since $S_\rho \subseteq |G| \times |G|$ and $|G|$ is finite, S_G is finite. Moreover, it is easy to verify:
Fact: For all $\rho, \varrho \in \Gamma_0^*$, $\rho \sim \varrho$ iff $S_\rho = S_\varrho$.

To show the fact, first assume $\rho \sim \varrho$. Then for each $(a,\, b) \in S_\rho$, by the definition of S_ρ, we have $G \models K(r^G,\, a) \wedge \rho(a,\, b)$. By the definition of $\sim$ and the assumption that $\rho \sim \varrho$, we have $G \models K(r^G,\, a) \wedge \varrho(a,\, b)$. Hence $(a,\, b) \in S_\varrho$. Therefore, $S_\rho \subseteq S_\varrho$. Similarly, it can be shown that $S_\varrho \subseteq S_\rho$. Hence $S_\rho = S_\varrho$. Conversely, assume that $S_\rho = S_\varrho$. Suppose, for *reductio*, that $\rho \not\sim \varrho$. Without loss of generality, assume that $G \not\models \forall x(K(r,x) \rightarrow \forall y\,(\rho(x,y) \rightarrow \varrho(x,y)))$. Then there exist $a, b \in |G|$ such that $G \models K(r^G,\, a) \wedge \rho(a,\, b) \wedge \neg\varrho(a,\, b)$. That is, $(a,\, b) \in S_\rho$ but $(a,\, b) \notin S_\varrho$. Hence $S_\rho \neq S_\varrho$. This contradicts the assumption. Therefore, the fact holds.

Next, consider function $g : M \rightarrow S_G$ defined by $g :\ [\rho] \mapsto S_\rho$. Using the fact above, it is easy to see that g is well-defined, total and injective. Therefore, because S_G is finite, so is M.

We next define a binary operation $\circ$ on M by $[\rho] \circ [\varrho] = [\rho \cdot \varrho]$. It is easy to verify the following claims.
Claim 5: $\circ$ is well-defined.

To prove this, for all $\rho_1, \rho_2, \varrho_1, \varrho_2 \in \Gamma_0^*$ such that $\rho_1 \sim \rho_2$ and $\varrho_1 \sim \varrho_2$, we show $\rho_1 \cdot \varrho_1 \sim \rho_2 \cdot \varrho_2$. Consider all $o, o_1 \in |G|$ such that $G \models K(r^G, o) \wedge \rho_1 \cdot \varrho_1(o, o_1)$. Clearly, there is $o' \in |G|$ such that $G \models \rho_1(o, o') \wedge \varrho_1(o', o_1)$. By $\rho_1 \sim \rho_2$, we have $G \models \rho_2(o, o')$. By $G \models \forall x\,(K \cdot l_j(r, x) \rightarrow K(r, x))$ and $G \models K(r^G, o) \wedge \rho_1(o, o')$, we have $G \models K(r^G, o')$. Thus by $\varrho_1 \sim \varrho_2$, we also have $G \models \varrho_2(o', o_1)$. Hence $G \models \rho_2 \cdot \varrho_2(o, o_1)$. Therefore, $G \models \forall x\,(K(r, x) \rightarrow \forall y\,(\rho_1 \cdot \varrho_1(x, y) \rightarrow \rho_2 \cdot \varrho_2(x, y)))$. Similarly, we can show $G \models \forall x\,(K(r, x) \rightarrow \forall y\,(\rho_2 \cdot \varrho_2(x, y) \rightarrow \rho_1 \cdot \varrho_1(x, y)))$. Therefore, $\rho_1 \cdot \varrho_1 \sim \rho_2 \cdot \varrho_2$.

Claim 6: $\circ$ is associative.

This is because for all $[\rho], [\varrho], [\lambda] \in M$, $([\rho] \circ [\varrho]) \circ [\lambda] = [\rho \cdot \varrho] \circ [\lambda] = [\rho \cdot \varrho \cdot \lambda] = [\rho] \circ ([\varrho \cdot \lambda]) = [\rho] \circ ([\varrho] \circ [\lambda])$.

Claim 7: $[\epsilon]$ is the identity for $\circ$.

This is because for any $[\rho] \in M$, $[\epsilon] \circ [\rho] = [\rho] = [\rho] \circ [\epsilon]$.

By these claims, $(M, \circ, [\epsilon])$ is a finite monoid.

Finally, we define $h : \Gamma_0^* \rightarrow M$ by $h : \rho \mapsto [\rho]$. Clearly, h is a homomorphism since $h(\rho \cdot \varrho) = [\rho \cdot \varrho] = [\rho] \circ [\varrho] = h(\rho) \circ h(\varrho)$. In addition, for any $i \in [1, n]$, by $[\alpha_i] = [\beta_i]$, $h(\alpha_i) = h(\beta_i)$. Moreover, by $[\alpha] \neq [\beta]$, we have $h(\alpha) \neq h(\beta)$. Therefore, $\Theta_0 \not\models_f (\alpha, \beta)$.

This completes the proof of Lemma 5.5. □

5.2 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. Indeed, in the context of $\mathcal{M}$ the complement of the (unrestricted) implication problem for P_c has the small model property. Specifically, for any schema Δ in $\mathcal{M}$, and any finite set Σ and φ of P_c constraints, if there exists a structure in $\mathcal{U}(\Delta)$ satisfying $\Psi = \bigwedge \Sigma \wedge \neg\varphi$, then Ψ has a finite model in $\mathcal{U}_f(\Delta)$, the size of which may be effectively computed from Δ and Ψ. This is because the definition of an $\mathcal{M}$ schema Δ bounds the out-degree of vertices (in the graph view) of a structure in $\mathcal{U}(\Delta)$ by the size of Δ, and the definition of P_c constraints allows us to inspect only vertices of a bounded distance (linear in the size of the constraints) from the root when checking these constraints. As an immediate result, if Ψ has a model in $\mathcal{U}(\Delta)$ then it has a "small" model in $\mathcal{U}_f(\Delta)$, which is at most exponentially large in the size of Δ and Ψ; this shows that the complement of the (unrestricted) implication problem for P_c in the context of $\mathcal{M}$ is in NEXPTIME.

Theorem 5.2 tells us that one can do much better than NEXPTIME: in the context of $\mathcal{M}$, path constraint implication is not only decidable in cubic-time, but is also finitely axiomatizable. We prove Theorem 5.2 by first presenting a finite axiomatization for implication and finite implication of P_c constraints, and then providing a cubic-time algorithm for testing path constraint implication.

A finite axiomatization Let $\mathcal{I}_r$ be the set consisting of the following inference rules:

—Reflexivity:

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

—Transitivity:

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

—Right-congruence:

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

—Commutativity:

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

—Forward-to-word:

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

—Word-to-forward:

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

—Backward-to-word:

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

—Word-to-backward:

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

Let Δ be a schema in $\mathcal{M}$ and $\Sigma \cup \{\varphi\}$ be a finite set of $P_c(\Delta)$ constraints. We use $\Sigma \vdash_{\mathcal{I}_r} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}_r$.

The lemma below shows that $\mathcal{I}_r$ is indeed a finite axiomatization of path constraints.

LEMMA 5.6. *Let Δ be any schema in $\mathcal{M}$. For every finite set Σ and φ of $P_c(\Delta)$ constraints,*

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

As an immediate result, we have the following:

COROLLARY 5.7. *Over any schema Δ in $\mathcal{M}$, the unrestricted implication and finite implication problems for $P_c(\Delta)$ coincide and are decidable.*

To see this, consider any finite set Σ and φ of $P_c(\Delta)$ constraints. If $\Sigma \models_{\Delta} \varphi$, then obviously $\Sigma \models_{(f,\Delta)} \varphi$. Conversely, if $\Sigma \models_{(f,\Delta)} \varphi$, then $\Sigma \vdash_{\mathcal{I}_r} \varphi$. By the soundness of $\mathcal{I}_r$ for unrestricted implication, we have $\Sigma \models_{\Delta} \varphi$. Thus these two problems coincide. Since $\mathcal{U}(\Delta)$ is definable in first-order logic, the equivalence of these problems leads to the decidability of both problems.

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

LEMMA 5.8. *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 $G \models \alpha(r^G, o)$.*

Lemma 5.8 shows, among other things, that the Commutativity rule in $\mathcal{I}_r$ is sound. Because of this lemma, the following lemmas hold in the context of $\mathcal{M}$, which establish the soundness of Forward-to-word, Word-to-forward, Backward-to-word, Word-to-backward in $\mathcal{I}_r$.

LEMMA 5.9. *For any schema Δ in the model $\mathcal{M}$, any forward $P_c(\Delta)$ constraint $\varphi = \forall\, x\,(\alpha(r,\, x) \rightarrow \forall\, y\,(\beta(x,\, y) \rightarrow \gamma(x,\, y)))$, a word constraint ψ of the form $\forall\, x\,(\alpha \cdot \beta(r,\, x) \rightarrow \alpha \cdot \gamma(r,\, x))$ in $P_c(\Delta)$, and for any $G \in \mathcal{U}(\Delta)$, $G \models \varphi$ iff $G \models \psi$.*

Proof: If $G \models \neg\psi$, then there is $b \in |G|$ such that $G \models \alpha \cdot \beta(r^G,\, b) \wedge \neg\alpha \cdot \gamma(r^G,\, b)$. Thus there exists $a \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b)$. In addition, we have $G \models \neg\gamma(a,\, b)$ since otherwise $G \models \alpha \cdot \gamma(r^G,\, b)$. Hence there are $a, b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b) \wedge \neg\gamma(a,\, b)$. Thus $G \models \neg\varphi$.

Conversely, if $G \models \neg\varphi$, then there must exist two nodes $a, b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b) \wedge \neg\gamma(a,\, b)$. By Lemma 5.8, a is the unique node such that $G \models \alpha(r^G,\, a)$. Thus $G \models \alpha \cdot \beta(r^G,\, b) \wedge \neg\alpha \cdot \gamma(r^G,\, b)$. That is, $G \models \neg\psi$. □

LEMMA 5.10. *Let Δ be a schema in $\mathcal{M}$, a backward constraint φ of $P_c(\Delta)$ be $\forall\, x\,(\alpha(r,\, x) \rightarrow \forall\, y\,(\beta(x,\, y) \rightarrow \gamma(y,\, x)))$, and $\psi = \forall\, x\,(\alpha(r,\, x) \rightarrow \alpha \cdot \beta \cdot \gamma(r,\, x))$ be a word constraint in $P_c(\Delta)$. Then for any $G \in \mathcal{U}(\Delta)$, $G \models \varphi$ iff $G \models \psi$.*

Proof: If $G \models \neg\psi$, then there is $a \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \neg\alpha \cdot \beta \cdot \gamma(r^G,\, a)$. By Lemma 5.8, there exists $b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b)$. Clearly, $G \models \neg\gamma(b,\, a)$ since otherwise we would have had $G \models \alpha \cdot \beta \cdot \gamma(r^G,\, a)$. Hence there are $a, b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b) \wedge \neg\gamma(b,\, a)$. Thus $G \models \neg\varphi$.

Conversely, if $G \models \neg\varphi$, then there must exist two nodes $a, b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b) \wedge \neg\gamma(b,\, a)$. By Lemma 5.8, a is the unique node such that $G \models \alpha(r^G,\, a)$, and b is the unique node such that $G \models \beta(a,\, b)$. Therefore, we have $G \models \neg\alpha \cdot \beta \cdot \gamma(r^G,\, a)$ since otherwise $G \models \gamma(b,\, a)$. Thus $G \models \neg\psi$. □

Using these lemmas, we show Lemma 5.6 as follows.

Proof Lemma 5.6: 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 following:
Claim 1: Let k with $k \geq max\{\, |pf(\psi)| + |lt(\psi)| + |rt(\psi)| \mid \psi \in \Sigma \cup \{\varphi\}\}$ be a natural number, where $pf(\psi)$, $lt(\psi)$ and $rt(\psi)$ are described in Definition 2.1. Then there exists $G \in \mathcal{U}(\Delta)$ such that $G \models \Sigma$ and for any path ρ with $|\rho| \leq k - |pf(\varphi) \cdot lt(\varphi)|$,
(1) if $G \models \forall\, x\,(pf(\varphi)(r,\, x) \rightarrow \forall\, y\,(lt(\varphi)(x,\, y) \rightarrow \rho(x,\, y)))$, then

$$\Sigma \vdash_{\mathcal{I}_r} \forall\, x\,(pf(\varphi)(r,\, x) \rightarrow \forall\, y\,(lt(\varphi)(x,\, y) \rightarrow \rho(x,\, y)));$$

(2) if $G \models \forall\, x\,(pf(\varphi)(r,\, x) \rightarrow \forall\, y\,(lt(\varphi)(x,\, y) \rightarrow \rho(y,\, x)))$, then

$$\Sigma \vdash_{\mathcal{I}_r} \forall\, x\,(pf(\varphi)(r,\, x) \rightarrow \forall\, y\,(lt(\varphi)(x,\, y) \rightarrow \rho(y,\, x))).$$

For if Claim 1 holds and $\Sigma \models_\Delta \varphi$, then by $G \models \Sigma$ we have $G \models \varphi$. Thus again by Claim 1, $\Sigma \vdash_{\mathcal{I}_r} \varphi$.

We next show Claim 1. Let $\Delta = (\mathcal{C},\, \nu,\, DBtype)$. We define the structure G described in Claim 1 in two steps: we first define the k-neighborhood of G, denoted by G_k, and then construct G from G_k. Here *the k-neighborhood of G* is

the substructure G_k of G with its universe

$$|G_k| = \{o \mid o \in |G|, G \models \alpha(r^G, o) \text{ for some } \alpha \in Paths(\Delta) \text{ with } |\alpha| \leq k\}.$$

We construct G_k by means of paths in $\mathcal{U}(\Delta)$. To do so, we define the following:

—$Paths^k(\Delta) = \{\alpha \mid \alpha \in Paths(\Delta),\ |\alpha| \leq k\}$.

—An equivalence relation $\approx$ on $Paths^k(\Delta)$: $\alpha \approx \beta$ iff $\Sigma \vdash_{\mathcal{I}_r} \forall x\ (\alpha(r, x) \rightarrow \beta(r, x))$. It should be noted that by Commutativity in $\mathcal{I}_r$, $\Sigma \vdash_{\mathcal{I}_r} \forall x\ (\alpha(r, x) \rightarrow \beta(r, x))$ iff $\Sigma \vdash_{\mathcal{I}_r} \forall x\ (\beta(r, x) \rightarrow \alpha(r, x))$.

—We use $\widehat{\alpha}$ to denote the equivalence class of α with respect to $\approx$, and define $\mathcal{A} = \{\widehat{\alpha} \mid \alpha \in Paths^k(\Delta)\}$.

—$type(\widehat{\alpha}) = type(\alpha)$, where $type(\alpha)$ is the type of path α determined by Δ. This is well-defined since if $\alpha \approx \beta$ then by the definition of $P_w(\Delta)$ they have the same type.

Using these notions, we define $G_k = (|G_k|, r^{G_k}, E^{G_k}, T^{G_k})$ as follows: (1) for each $\widehat{\alpha} \in \mathcal{A}$, let $o(\widehat{\alpha})$ be a distinct node and let $|G_k| = \{o(\widehat{\alpha}) \mid \widehat{\alpha} \in \mathcal{A}\}$; (2) let $r^{G_k} = o(\widehat{\epsilon})$; (3) for each $\tau \in T(\Delta)$, let $\tau^{G_k} = \{o(\widehat{\alpha}) \mid \widehat{\alpha} \in \mathcal{A},\ type(\widehat{\alpha}) = \tau\}$; (4) for each $o(\widehat{\alpha})$, if $type(\widehat{\alpha}) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or $type(\widehat{\alpha})$ is some class $C \in \mathcal{C}$ and $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$) and there is $\beta \in \widehat{\alpha}$ with $|\beta| < k$, then for each $i \in [1, n]$, let $G_k \models l_i(o(\widehat{\alpha}), o(\widehat{\beta \cdot l_i}))$. Note that this is well-defined by Transitivity and Right-congruence in $\mathcal{I}_r$.

Based on G_k, we define the structure G such that G satisfies the type constraint determined by Δ, among other things, as follows. For each $\tau \in T(\Delta)$, let $o(\tau)$ be a distinct node. Let $G = (|G|, r^G, E^G, T^G)$, where $|G| = |G_k| \cup \{o(\tau) \mid \tau \in T(\Delta)\}$; $r^G = r^{G_k}$; for each $\tau \in T(\Delta)$, $\tau^G = \tau^{G_k} \cup \{o(\tau)\}$; and for each $l \in E(\Delta)$, if $G_k \models l(o, o')$ then $G \models l(o, o')$. Moreover,

—for each $o(\widehat{\alpha}) \in |G_k|$, if $type(\widehat{\alpha}) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or $type(\widehat{\alpha})$ is some class $C \in \mathcal{C}$ and $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$) and for some $i \in [1, n]$, $o(\widehat{\alpha})$ does not have any outgoing edge labeled with l_i, then let $G \models l_i(o(\widehat{\alpha}), o(\tau_i))$;

—for each type $\tau \in T(\Delta)$, if τ is a record $[l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or τ is some class $C \in \mathcal{C}$ and $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), then for each $i \in [1, n]$, let $G \models l_i(o(\tau), o(\tau_i))$.

We now show that G is indeed the structure described in Claim 1.

1. $G \in \mathcal{U}_f(\Delta)$. It is easy to verify that $|G|$ is finite and $G \models \Phi(\Delta)$, where $\Phi(\Delta)$ is the type constraint determined by Δ.

2. $G \models \Sigma$. We first show the following claim.

Claim 2: For every $\alpha \in Paths^k(\Delta)$, $G \models \alpha(r^G, o(\widehat{\alpha}))$.

As a result of Claim 2 and Lemma 5.8, $o(\widehat{\alpha})$ is the unique node in G such that $G \models \alpha(r^G, o(\widehat{\alpha}))$.

We show Claim 2 by induction on $|\alpha|$.

Base case: $\alpha = \epsilon$. Recall that $r^G = o(\widehat{\epsilon})$. Obviously, $G \models \epsilon(r^G, o(\widehat{\epsilon}))$.

Inductive step: Assume Claim 2 for α. We next show that Claim 2 also holds for $\alpha \cdot l$, where $\alpha \cdot l \in Paths^k(\Delta)$. By induction hypothesis, $G \models \alpha(r^G, o(\widehat{\alpha}))$. Since $\alpha \cdot l \in Paths^k(\Delta)$, $|\alpha \cdot l| \leq k$. Hence $|\alpha| < k$. By $\alpha \in \widehat{\alpha}$ and the definition of G,

we have $G \models \alpha(r^G, o(\widehat{\alpha})) \wedge l(o(\widehat{\alpha}), o(\widehat{\alpha \cdot l}))$. That is, $G \models \alpha \cdot l(r^G, o(\widehat{\alpha \cdot l}))$. Hence Claim 2 holds.

Using Claim 2, we show $G \models \Sigma$. Suppose, for *reductio*, that there is $\psi \in \Sigma$ such that $G \models \neg\psi$. Consider the following cases. If ψ is a forward constraint $\forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y)))$, then there must be $a, b \in |G|$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b) \wedge \neg\gamma(a, b)$. Thus by Lemma 5.8 and Claim 2, we have $a = o(\widehat{\alpha})$ and $b = o(\widehat{\alpha \cdot \beta})$. By Forward-to-word in $\mathcal{I}_r$, we have $\alpha \cdot \beta \approx \alpha \cdot \gamma$. Therefore, again by Claim 2 and Lemma 5.8, we have $G \models \alpha \cdot \gamma(r^G, o(\widehat{\alpha \cdot \beta}))$ and $G \models \gamma(o(\widehat{\alpha}), o(\widehat{\alpha \cdot \beta}))$. This contradicts the assumption. If ψ is a backward constraint $\forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x)))$, then there are $a, b \in |G|$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b) \wedge \neg\gamma(b, a)$. Again by Lemma 5.8 and Claim 2, we have $a = o(\widehat{\alpha})$ and $b = o(\widehat{\alpha \cdot \beta})$. By Backward-to-word in $\mathcal{I}_r$, we have $\alpha \approx \alpha \cdot \beta \cdot \gamma$. Therefore, again by Claim 2 and Lemma 5.8, we have $G \models \alpha \cdot \beta \cdot \gamma(r^G, o(\widehat{\alpha}))$ and thus $G \models \gamma(o(\widehat{\alpha \cdot \beta}), o(\widehat{\alpha}))$. This again contradicts the assumption. Thus $G \models \psi$. Hence $G \models \Sigma$.

3. G has the property described by Claim 1. To show this, let ρ be a path such that $|\rho| \leq k - |pf(\varphi) \cdot lt(\varphi)|$.

First, suppose $G \models \forall\ x\ (pf(\varphi)(r,\ x) \rightarrow \forall\ y\ (lt(\varphi)(x, y) \rightarrow \rho(x, y)))$, then by Lemma 5.9, $G \models \forall\ x\ (pf(\varphi) \cdot lt(\varphi)(r, x) \rightarrow pf(\varphi) \cdot \rho(r, x))$. By Claim 2 and Lemma 5.8, we have $G \models pf(\varphi) \cdot \rho(r^G, o(\widehat{pf(\varphi) \cdot lt(\varphi)}))$ and moreover, $pf(\varphi) \cdot lt(\varphi) \approx pf(\varphi) \cdot \rho$. Thus by the definition of $\approx$ and Commutativity in $\mathcal{I}_r$, we have $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(pf(\varphi)(r, x) \cdot lt(\varphi)(r, x) \rightarrow pf(\varphi) \cdot \rho(r, x))$. By Word-to-forward in $\mathcal{I}_r$, we have $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(pf(\varphi)(r, x) \rightarrow \forall y\,(lt(\varphi)(x, y) \rightarrow \rho(x, y)))$.

Now suppose $G \models \forall x\,(pf(\varphi)(r, x) \rightarrow \forall y\,(lt(\varphi)(x, y) \rightarrow \rho(y, x)))$, then by Lemma 5.10, we have $G \models \forall x\,(pf(\varphi)(r, x) \rightarrow pf(\varphi) \cdot lt(\varphi) \cdot \rho(r, x))$. By Claim 2 and Lemma 5.8, we have $G \models pf(\varphi) \cdot lt(\varphi) \cdot \rho(r^G, o(\widehat{pf(\varphi)}))$, and moreover, $pf(\varphi) \approx pf(\varphi) \cdot lt(\varphi) \cdot \rho$. Thus by the definition of $\approx$ and the Commutativity rule in $\mathcal{I}_r$, we have $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(pf(\varphi)(r, x) \rightarrow pf(\varphi) \cdot lt(\varphi) \cdot \rho(r, x))$. By Word-to-backward in $\mathcal{I}_r$, we have $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(pf(\varphi)(r, x) \rightarrow \forall y\,(lt(\varphi)(x, y) \rightarrow \rho(y, x)))$.

This completes the proof of Claim 1, and therefore, the proof of Lemma 5.6. □

An algorithm. Next, we present an algorithm for testing path constraint implication in the context of $\mathcal{M}$. Let Δ be a schema in $\mathcal{M}$. This algorithm takes as input a finite set Σ of $P_c(\Delta)$ constraints and paths α, β such that $\alpha \cdot \beta$ in $Paths(\Delta)$. It computes a structure G having the following properties: $G \models \Sigma$ and there are $a, b \in |G|$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b)$. In addition, for any path γ,

$$G \models \gamma(a, b) \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y))),$$
$$G \models \gamma(b, a) \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x))).$$

By Lemma 5.6, this algorithm can be used for testing both implication and finite implication of $P_c(\Delta)$ constraints in the context of $\mathcal{M}$.

The algorithm needs the following notations. Let Δ be a schema in $\mathcal{M}$, Σ be a finite set of $P_c(\Delta)$ constraints and paths α, β such that $\alpha \cdot \beta \in Paths(\Delta)$. We

Algorithm 5.1:

Input: a finite sub Σ of $P_c(\Delta)$ constraints and paths α, β such that $\alpha \cdot \beta \in Paths(\Delta)$
Output: the structure G described in Section 5.2

1. E_ϕ := the set of edge labels appearing in either $\alpha \cdot \beta$ or some path in constraints of Σ;
2. $Rules := \Sigma$;
3. $G := (|G|, r^G, E_\phi^G)$, where
   - $|G| = \{o(\rho) \mid \rho \in CloPts(\Sigma, \alpha \cdot \beta),\ o(\rho)$ is a distinct node$\}$,
   - $r^G = o(\epsilon)$,
   - E_ϕ^G is populated in such a way that $G \models l(o(\rho), o(\varrho))$ iff $\varrho = \rho \cdot l$;
4. **for** each $\psi \in \Sigma$ **do**
   (1) **if** $\psi = \forall x\,(\rho(r, x) \rightarrow \forall y\,(\varrho(x, y) \rightarrow \zeta(x, y)))$ **then**
   (i) $Rules := Rules \setminus \{\forall x\,(\rho(r, x) \rightarrow \forall y\,(\varrho(x, y) \rightarrow \zeta(x, y)))\}$;
   (ii) $merge(o_{\rho\cdot\varrho}, o_{\rho\cdot\zeta})$,
   where $o_{\rho\cdot\varrho}, o_{\rho\cdot\zeta} \in |G|$ such that $G \models \rho \cdot \varrho(r^G, o_{\rho\cdot\varrho}) \wedge \rho \cdot \zeta(r^G, o_{\rho\cdot\zeta})$;
   (2) **if** $\psi = \forall x\,(\rho(r, x) \rightarrow \forall y\,(\varrho(x, y) \rightarrow \zeta(y, x)))$ **then**
   (i) $Rules := Rules \setminus \{\forall x\,(\rho(r, x) \rightarrow \forall y\,(\varrho(x, y) \rightarrow \zeta(y, x)))\}$;
   (ii) $merge(o_\rho, o_{\rho\cdot\varrho\cdot\zeta})$,
   where $o_\rho, o_{\rho\cdot\varrho\cdot\zeta} \in |G|$ such that $G \models \rho(r^G, o_\rho) \wedge \rho \cdot \varrho \cdot \zeta(r^G, o_{\rho\cdot\varrho\cdot\zeta})$;
5. output G.

procedure $merge(a, b)$

1. **for** each $o \in |G|$ **do**
   if there is $l \in E_\phi^G$ such that $G \models l(o, b)$ **then**
   (1) delete from E_ϕ^G the edge labeled l from o to b;
   (2) add to E_ϕ^G an edge labeled l from o to a;
2. **for** each $l \in E_\phi$ **do**
   if there are $o_a, o_b \in |G|$ such that $G \models l(b, o_b) \wedge l(a, o_a)$ and $o_a \neq o_b$ **then**
   (1) delete from E_ϕ^G the edge labeled l from b to o_b;
   (2) add to E_ϕ^G an edge labeled l from a to o_b;
   (3) $merge(o_a, o_b)$;
3. $|G| := |G| \setminus \{b\}$;

Fig. 6. An algorithm for testing path constraint implication in $\mathcal{M}$

define

$$\begin{aligned} Pts(\Sigma, \alpha \cdot \beta) &= \{\alpha \cdot \beta\} \\ &\cup \{pf(\psi) \cdot lt(\psi),\ pf(\psi) \cdot rt(\psi) \mid \psi \in \Sigma,\ \psi \text{ is forward}\} \\ &\cup \{pf(\psi) \cdot lt(\psi) \cdot rt(\psi) \mid \psi \in \Sigma,\ \psi \text{ is backward}\}, \\ CloPts(\Sigma, \alpha \cdot \beta) &= \{\rho \mid \varrho \in Pts(\Sigma, \alpha \cdot \beta),\ \rho \preceq_p \varrho\}. \end{aligned}$$

Here $\rho \preceq_p \varrho$ denotes that ρ is a prefix of ϱ, as described in Section 2.

Using these notions, we give Algorithm 5.1 in Fig. 6. The following should be noted about the algorithm.

Remark 1: Algorithm 5.1 is independent of any particular schema. Although it is required that input constraints and path are defined over some schema in $\mathcal{M}$, no particular information about the schema is used by the algorithm. As a result, this

algorithm can be used in the context of any schema in $\mathcal{M}$.

Remark 2: Although structure G computed by the algorithm may not be in $\mathcal{U}(\Delta)$, G can be naturally extended to a structure $H \in \mathcal{U}_f(\Delta)$ as follows: $H = (|H|, r^H, E^H, T^H)$, where $|H| = |G| \cup \{o(\tau) \mid \tau \in T(\Delta),\ o(\tau)$ is a distinct node$\}$; the root r^H is the same as r^G; for each type $\tau \in T(\Delta)$, we populate τ^H to be $\{o(\tau)\} \cup \{o \mid o \in |G|,\ \rho \in Paths(\Delta),\ type(\rho) = \tau,\ G \models \rho(r^G, o)\}$; and for each $l \in E(\Delta)$, if $G \models l(o, o')$, then $H \models l(o, o')$. Moreover,

—for any $o \in |G|$, if there is $\rho \in Paths(\Delta)$ such that $G \models \rho(r^G, o)$ and $type(\rho) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or $type(\rho)$ is a class $C \in \mathcal{C}$ and $\nu(C) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), then let $H \models l_i(o, o(\tau_i))$ for any $i \in [1, n]$ such that o does not have an outgoing edge labeled l_i;

—for each $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ in $T(\Delta)$ (and for each class τ in $\mathcal{C}$ with $\nu(\tau) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$) let $H \models l_i(o(\tau), o(\tau_i))$ for each $i \in [1, n]$.

It is easy to verify that $H \in \mathcal{U}(\Delta)$ as long as $\Sigma \subseteq P_c(\Delta)$ and $\alpha \cdot \beta \in Paths(\Delta)$. In addition, H is finite and thus $H \in \mathcal{U}_f(\Delta)$. We refer to the structure H defined above as *the extension of G with respect to Δ*.

Remark 3: The rationale behind the procedure *merge* is Commutativity, Transitivity and Right-congruence in $\mathcal{I}_r$. The rationale behind step 4 (1) (i) and 4 (2) (i) of Algorithm 5.1 is Lemma 5.8. Let Δ be a schema in $\mathcal{M}$ and $G \in \mathcal{U}(\Delta)$. For any path $\rho \in Paths(\Delta)$, there exists a unique $o \in |G|$ such that $G \models \rho(r^G, o)$. As a result, every constraint in Σ can be applied at most once by the algorithm. It is because of this property that Algorithm 5.1 has low complexity.

Next, we analyze the complexity of the algorithm. Let n_E be the cardinality of E_ϕ, n_C the cardinality of $CloPts(\Sigma, \alpha \cdot \beta)$, n_G the size of $|G|$, n the length of Σ and $\alpha \cdot \beta$, and n_Σ the cardinality of Σ. Note the following. (1) $n_E \leq n$, $n_C \leq n$, $n_G \leq n$ and $n_\Sigma \leq n$. (2) Step 4 is executed n_Σ times. (3) Testing whether $G \models \rho(r^G, o_\rho)$ in step 4 can be done in at most $O(n_G\,|\rho|)$ time. Therefore, it can be done in $O(n^2)$ time. (4) The procedure *merge* is executed at most n_G times. Each step takes $O(n_E\ n_G)$ time. Hence the total cost of executing *merge* is $O(n_G^2\ n_E)$, i.e., $O(n^3)$. Therefore, Algorithm 5.1 runs in $O(n^3)$ time.

The lemma below shows that Algorithm 5.1 is correct.

LEMMA 5.11. *Let Δ be a schema in $\mathcal{M}$. Given a finite set Σ of $P_c(\Delta)$ constraints and paths α, β such that $\alpha \cdot \beta \in Paths(\Delta)$, Algorithm 5.1 computes a structure G having the following property: $G \models \Sigma$, and there exist $a, b \in |G|$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b)$. Moreover, for any path γ,*

$$G \models \gamma(a, b) \quad \textit{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y))),$$
$$G \models \gamma(b, a) \quad \textit{iff} \quad \Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x))).$$

Proof: (*if*) Step 4 of Algorithm 5.1 ensures $G \models \Sigma$ by Lemma 5.8. Step 3 ensures that there are $a, b \in |G|$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b)$. Let H denote the extension of G with respect to Δ. Then it is easy to verify $H \models \alpha(r^G, a) \wedge \beta(a, b)$ and $H \models \Sigma$. Thus if $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y)))$, then by Lemma 5.6, $H \models \gamma(a, b)$. From the definition of H follows $G \models \gamma(a, b)$. Similarly, if $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x)))$, then $G \models \gamma(b, a)$.

(*only if*) Conversely, by an induction on the number of steps in the construction of G by the algorithm, we can show that for all paths ρ and ϱ, if there exists node $o \in |G|$ such that $G \models \rho(r^G, o) \wedge \varrho(r^G, o)$, then $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(\rho(r, x) \rightarrow \varrho(r, x))$. Indeed, each step of the construction in fact corresponds to applications of some rules in $\mathcal{I}_r$. For example, step 4 (1) corresponds to an application of Forward-to-word, step 4 (2) corresponds to an application of Backward-to-word, and *merge* corresponds to applications of Transitivity, Right-congruence and Commutativity in $\mathcal{I}_r$. As a result, if $G \models \gamma(a, b)$, then by Word-to-forward in $\mathcal{I}_r$, it can be verified that $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y)))$. Similarly, if $G \models \gamma(b, a)$, then $\Sigma \vdash_{\mathcal{I}_r} \forall x\,(\alpha(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x)))$ by Word-to-backward. □

From Lemmas 5.11 and 5.6, Theorem 5.2 follows immediately.

6. COMPLICATION OF PATH CONSTRAINT IMPLICATION WITH TYPES

In light of Theorems 5.1 and 5.2, one might be 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 6.1. *In the context of semistructured data, the implication and finite implication problems for local inclusion constraints are decidable in PTIME.*

THEOREM 6.2. *In the context of the object-oriented data model $\mathcal{M}^+$, the implication and finite implication problems for local inclusion 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 counter-intuitive 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 the implication problems in a different way to path constraints. More specifically, let $\Sigma \cup \{\varphi\}$ be a finite set of local inclusion constraints 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 $G \notin \mathcal{U}(\Delta)$. 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 $H \models \bigwedge \Sigma \wedge \neg\varphi$.

We show Theorems 6.1 and 6.2 in Sections 6.1 and 6.2, respectively.

6.1 Decidability on untyped data

We first show Theorem 6.1. The idea of the proof is by reduction to word constraint implication. The following has been shown in [Abiteboul and Vianu 1999].

LEMMA 6.3 [Abiteboul and Vianu 1999]. *In the context of semistructured data, the implication and finite implication problems for P_w are decidable in PTIME.*

Recall Definition 4.1: the (finite) implication problem for local inclusion constraints is to determine, given any finite sets Σ_K, Σ_r and φ of P_c constraints,

whether $\Sigma_K \cup \Sigma_r \models \varphi$ ($\Sigma_K \cup \Sigma_r \models_f \varphi$), where $\Sigma_K \cup \{\varphi\}$ consists of constraints bounded by a path α and a label K, and Σ_r is excluded by α and K. This means that for each $\phi \in \Sigma_K \cup \{\varphi\}$, ϕ is a forward constraint and the prefix of ϕ, $pf(\phi)$, is $\alpha \cdot K$. Moreover, 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'. In particular, when $\alpha' = \epsilon$, $rt(\phi)$ is a path of zero or more K labels. Let $\Sigma = \Sigma_K \cup \Sigma_r$.

To reduce this problem to the (finite) implication problem for word constraints, we first define a function f that is used in the construction of the reduction. Let α be a path and φ be a P_c constraint. Then $f(\varphi, \alpha)$ 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 a forward constraint $\forall x\,(\rho(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(x, y)))$;
— $\forall x\,(\alpha \cdot \rho(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x)))$, if φ is a backward constraint $\forall x\,(\rho(r, x) \rightarrow \forall y\,(\beta(x, y) \rightarrow \gamma(y, x)))$.

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

$$\varphi^1 = g_1(\varphi), \qquad \Sigma_K^1 = \{g_1(\phi) \mid \phi \in \Sigma_K\}, \qquad \Sigma_r^1 = \{g_1(\psi) \mid \psi \in \Sigma_r\}.$$

Second, we define another function g_2 such that $\phi = f(g_2(\phi), K)$ for all constraint $\phi \in \Sigma_K^1 \cup \{\varphi^1\}$. That is, g_2 further removes K from the prefix of ϕ. Now let

$$\varphi^2 = g_2(\varphi^1), \qquad \Sigma_K^2 = \{g_2(\phi) \mid \phi \in \Sigma_K^1\}.$$

Clearly, $\Sigma_K^2 \subseteq P_w$ and $\varphi^2 \in P_w$. The functions g_1 and g_2 establish a reduction:

LEMMA 6.4. *In the context of semistructured data,*

$$\Sigma \models \varphi \quad \textit{iff} \quad \Sigma_K^1 \cup \Sigma_r^1 \models \varphi^1 \quad \textit{iff} \quad \Sigma_K^2 \models \varphi^2,$$
$$\Sigma \models_f \varphi \quad \textit{iff} \quad \Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1 \quad \textit{iff} \quad \Sigma_K^2 \models_f \varphi^2.$$

This lemma suffices to show Theorem 6.1. For if it holds, then the (finite) implication problem for local inclusion 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 inclusion constraints follows from Lemma 6.3.

Proof Lemma 6.4: We show the lemma for finite implication only. The proof for implication is similar and simpler. The proof consists of two parts.
(*part I:*) We first show that $\Sigma \models_f \varphi$ iff $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$. It suffices to prove that for each path α and each finite set Σ and φ of P_c constraints, $\bigwedge \Sigma \wedge \neg\varphi$ has a finite model iff $\bigwedge_{\phi \in \Sigma} f(\phi, \alpha) \wedge \neg f(\varphi, \alpha)$ has a finite model. To simplify the discussion, assume that all the constraints in $\Sigma \cup \{\varphi\}$ are of the forward form. The proof for the general case is similar.
(*if*) Suppose that $\bigwedge_{\phi \in \Sigma} f(\phi, \alpha) \wedge \neg f(\varphi, \alpha)$ has a finite model $G = (|G|, r^G, E^G)$. Then we construct a finite model of $\bigwedge \Sigma \wedge \neg\varphi$. That is, we construct a finite σ-structure H such that $H \models \bigwedge \Sigma \wedge \neg\varphi$. Here σ is the signature defined in Section 2.

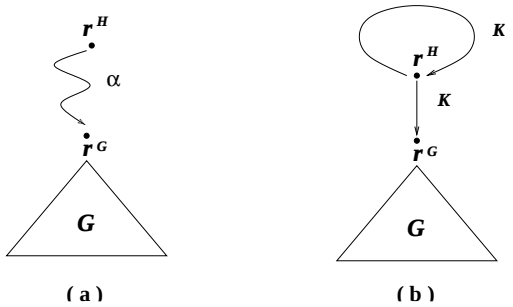

Fig. 7. The structure H in the proof of Lemma 6.4

Recall the notations *pf*, *lt* and *rt* from Definition 2.1. Since $G \models \neg f(\varphi, \alpha)$, there exist $a, b, c \in |G|$ such that $G \models \alpha(r^G, a) \wedge pf(\varphi)(a, b) \wedge lt(\varphi)(b, c) \wedge \neg rt(\varphi)(b, c)$. We define H to be the same as G except that its root is a. More specifically, $H = (|G|, a, E^G)$.

It is easy to verify the following.

(1) H is finite. This is because $|G|$ is finite and $|H| = |G|$.

(2) $H \models \neg\varphi$. Since $b \in |H|$ and $c \in |H|$, from the definition of H follows that $H \models pf(\varphi)(a, b) \wedge lt(\varphi)(b, c) \wedge \neg rt(\varphi)(b, c)$. That is, $H \models \neg\varphi$.

(3) $H \models \Sigma$. We show this by *reductio*.

Suppose that there exists $\phi \in \Sigma$ such that $H \models \neg\phi$. Then there must be $d, e \in |H|$ such that $H \models pf(\phi)(a, d) \wedge lt(\phi)(d, e) \wedge \neg rt(\phi)(d, e)$. Then by the definition of H, it must be the case that $G \models \alpha(r^G, a) \wedge pf(\phi)(a, d) \wedge lt(\phi)(d, e) \wedge \neg rt(\phi)(d, e)$. That is, $G \models \neg f(\phi, \alpha)$. This contradicts the assumption that $G \models \{f(\phi, \alpha) \mid \phi \in \Sigma\}$.

(*only if*) Suppose that $\bigwedge \Sigma \wedge \neg\varphi$ has a finite model $G = (|G|, r^G, E^G)$. Then we show that $\bigwedge_{\phi \in \Sigma} f(\phi, \alpha) \wedge \neg f(\varphi, \alpha)$ also has a finite model $H = (|H|, r^H, E^H)$ as shown in Fig. 7 (a), which extends G with a new root r^H and a path α from r^H to G. Specifically, let $L_\alpha = \{\rho \mid \rho \text{ is a path}, \rho \preceq_p \alpha, \rho \neq \alpha\}$. where $\rho \preceq_p \alpha$ denotes that ρ is a prefix of α. For each $\rho \in L_\alpha$, let c_ρ be a distinguished node not in $|G|$. Let $|H| = |G| \cup \{c_\rho \mid \rho \in L_\alpha\}$, $r^H = c_\epsilon$, and for all $a, b \in |H|$ and each $l \in E$, let $H \models l(a, b)$ iff one of the following conditions is satisfied: either (1) there exists $\rho \in L_\alpha$ such that $a = c_\rho$ and $b = c_{\rho \cdot l}$ and $\rho \cdot l \in L_\alpha$; or (2) there exists $\rho \in L_\alpha$ such that $\alpha = \rho \cdot l$ and $a = c_\rho$ and $b = r^G$; or (3) $a, b \in |G|$ and $G \models l(a, b)$.

It is easy to verify the following.

(1) H is finite. This is because both $|G|$ and L_α are finite.

(2) $H \models \neg f(\varphi, \alpha)$. To show this, we first observe the following simple facts, which are immediate from the construction of H.

Fact 1: r^G is the unique node in $|H|$ such that $H \models \alpha(c_\epsilon, r^G)$. In addition, for all $\rho, \varrho \in L_\alpha$ and $l \in E$, $H \models l(c_\rho, c_\varrho)$ iff $\varrho = \rho \cdot l$.

Fact 2: For each $\rho \in L_\alpha$, $l \in E$, (i) for each $a \in |G| \backslash \{r^G\}$, $H \models \neg l(c_\rho, a) \wedge \neg l(a, c_\rho)$; (ii) if $\alpha \neq \rho \cdot l$, then $H \models \neg l(c_\rho, r^G) \wedge \neg l(r^G, c_\rho)$; and finally, (iii) if $\alpha = \rho \cdot l$, then $H \models l(c_\rho, r^G) \wedge \neg l(r^G, c_\rho)$.

Using these facts, we show $H \models \neg f(\varphi, \alpha)$. By $G \models \neg\varphi$, there must be nodes $b, c \in |G|$ such that $G \models pf(\varphi)(r^G, b) \wedge lt(\varphi)(b, c) \wedge \neg rt(\varphi)(b, c)$. From Facts 1 and 2 above follows that $H \models \alpha(c_\epsilon, r^G) \wedge pf(\varphi)(r^G, b) \wedge lt(\varphi)(b, c) \wedge \neg rt(\varphi)(b, c)$. That is, $H \models \neg f(\varphi, \alpha)$.

(3) $H \models \{f(\phi, \alpha) \mid \phi \in \Sigma\}$. We prove this by *reductio*.

Suppose that there exists $\phi \in \Sigma$ such that $H \models \neg f(\phi, \alpha)$. Then there exist $a, b, c \in |H|$ such that $H \models \alpha(c_\epsilon, a) \wedge pf(\phi)(a, b) \wedge lt(\phi)(b, c) \wedge \neg rt(\phi)(b, c)$. By Fact 1, $a = r^G$. By Fact 2, $b, c \in |G|$, and moreover, by the construction of H, we have $G \models pf(\phi)(a, b) \wedge lt(\phi)(b, c) \wedge \neg rt(\phi)(b, c)$. That is, $G \models \neg\phi$. This contradicts the assumption that $G \models \Sigma$.

(*part II*) We next show that $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$ iff $\Sigma_K^2 \models_f \varphi^2$.

The argument of Part I suffices to show that if $\Sigma_K^2 \models_f \varphi^2$ then $\Sigma_K^1 \models_f \varphi^1$ and thus $\Sigma_K^1 \cup \Sigma_r^1 \models_f \varphi^1$.

Conversely, suppose that $\bigwedge \Sigma_K^2 \wedge \neg\varphi^2$ has a finite model $G = (|G|, r^G, E^G)$. Then we construct from G a finite model H of $\bigwedge \Sigma_K^1 \wedge \bigwedge \Sigma_r^1 \wedge \neg\varphi^1$. Let $H = (|H|, r^H, E^H)$, which is a mild extension of G as shown in Fig. 7 (b). More specifically, $|H| = |G| \cup \{r^H\}$ with $r^H \notin |G|$, and E^H is E^G augmented with two edges labeled K, by letting $H \models K(r^H, r^G) \wedge K(r^H, r^H)$. In other words, $E^H = E^G \cup \{K(r^H, r^G),\ K(r^H, r^H)\}$. Clearly, H is finite since G is finite. Below we show that H is a model of $\bigwedge \Sigma_K^1 \wedge \bigwedge \Sigma_r^1 \wedge \neg\varphi^1$.

(1) $H \models \Sigma_r^1$. For each $\phi \in \Sigma_r^1$, since Σ_r is excluded by α and K, K is not a prefix of $pf(\phi)$. Consider the following cases. If $pf(\phi) \neq \epsilon$, then by the construction of H, there are no $o, o' \in |H|$ such that $H \models pf(\phi)(r^H, o) \wedge lt(\phi)(o, o')$. Hence $H \models \phi$. If $pf(\phi) = \epsilon$, then by the definition of constraints excluded by α and K, ϕ is of the form $\forall\, x\, (\epsilon(r, x) \rightarrow K^n(r, x))$, where K^n denotes a path of length n (possibly zero), consisting of K labels. Clearly, $H \models \phi$ since $(r^H, r^H) \in K^H$. Thus $H \models \Sigma_r^1$.

(2) $H \models \Sigma_K^1$. Suppose, for *reductio*, that there is $\phi \in \Sigma_K^1$ such that $H \models \neg\phi$. Since Σ_K is bounded by α and K, by the definition of bounded constraints given in Section 4, ϕ must be of the form $\forall\, x\, (K(r, x) \rightarrow \forall\, y\, (\beta(x, y) \rightarrow \gamma(x, y)))$, where $K \not\preceq_p \beta$ and $\beta \neq \epsilon$. Thus by the definition of H, if $H \models \neg\phi$, then there exists node $o \in |G|$ such that $H \models K(r^H, r^G) \wedge \beta(r^G, o) \wedge \neg\gamma(r^G, o)$. Thus we have $G \models \beta(r^G, o) \wedge \neg\gamma(r^G, o)$. That is, $G \models \neg g_2(\phi)$. This contradicts the assumption that G is a model of $\bigwedge \Sigma_K^2 \wedge \neg\varphi^2$. Therefore, $H \models \phi$. Hence $H \models \Sigma_K^1$.

(3) $H \models \neg\varphi^1$. Observe that by the definition of implication of local inclusion constraints, φ is bounded by α and K. As a result, φ^1 must be of the form $\forall\, x\, (K(r, x) \rightarrow \forall\, y\, (\beta(x, y) \rightarrow \gamma(x, y)))$, where $K \not\preceq_p \beta$ and $\beta \neq \epsilon$. Therefore, φ^2 is of the form $\forall\, x\, (\beta(r, x) \rightarrow \gamma(r, x))$. By $G \models \neg\varphi^2$, there is $o \in |G|$ such that $G \models \beta(r^G, o) \wedge \neg\gamma(r^G, o)$. By the definition of H, $H \models K(r^H, r^G) \wedge \beta(r^G, o) \wedge \neg\gamma(r^G, o)$. That is, $H \models \neg\varphi^1$.

This completes the proof of Lemma 6.4. □

It is worth mentioning the following.

(1) This proof is not applicable in the context of $\mathcal{M}$ or $\mathcal{M}^+$. For any schema Δ in these models, the structure H shown in Fig. 7 (b) is not in $\mathcal{U}(\Delta)$, because of the type constraint $\Phi(\Delta)$.

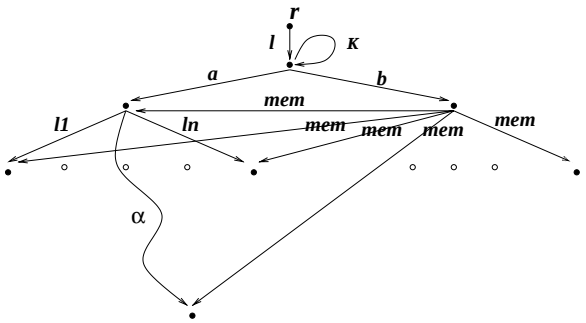

Fig. 8. The structure G in the proof of Lemma 6.5

(2) Theorem 6.1 does not conflict with the proof of Theorem 5.1. Recall Σ_1, Σ_2, $\varphi_{(\alpha,\beta)}$ and $\varphi_{(\beta,\alpha)}$ defined in Section 5.1. The implication $\Sigma_1 \cup \Sigma_2 \models \varphi_{(\alpha,\beta)} \wedge \varphi_{(\beta,\alpha)}$ is not an instance of the implication problem for local inclusion constraints.

6.2 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 6.2 by reduction from the word problem for (finite) monoids.

Recall the alphabet Γ_0 and the equations Θ_0 given in Section 5.1. Using Γ_0, we define a schema $\Delta_1 = (\mathcal{C}, \nu, DBtype)$ in $\mathcal{M}^+$ such that $\mathcal{C} = \{C, C_s, C_l\}$, ν is defined by:

$$C \mapsto [l_1 : C, \ldots, l_m : C], \qquad C_s \mapsto \{C\}, \qquad C_l \mapsto [a : C, b : C_s, K : C_l],$$

where $a, b, l, K \notin \Gamma_0$, and $DBtype = [l : C_l]$. 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 α.

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

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

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

Observe that Σ can be partitioned into Σ_K and Σ_r, where Σ_K consists of the constraints defined in (1) and (2) and Σ_r consists of those specified in (3) and (4). Note that $\Sigma_K \cup \{\varphi_{(\alpha,\beta)}\}$ is bounded by l and K whereas Σ_r is excluded by l and K.

A structure G satisfying Σ and $\Phi(\Delta_1)$ is depicted in Fig. 8.

The lemma below shows that the encoding is indeed a reduction from the word problem for (finite) monoids to the (finite) implication problem for local inclusion constraints. From this and Lemma 5.4 follows immediately Theorem 6.2.

LEMMA 6.5. *In the context of $\mathcal{M}^+$, for all α and β in Γ_0^*,*

$$\Theta_0 \models (\alpha,\ \beta) \quad \textit{iff} \quad \Sigma \models_{\Delta_1} \varphi_{(\alpha,\beta)}, \qquad \Theta_0 \models_f (\alpha,\ \beta) \quad \textit{iff} \quad \Sigma \models_{(f,\,\Delta_1)} \varphi_{(\alpha,\beta)}.$$

To prove this lemma, first observe the following. These lemmas hold because of the type constraint $\Phi(\Delta_1)$. In the context of semistructured data, however, they no longer hold in general.

LEMMA 6.6. *For each $G \in \mathcal{U}(\Delta_1)$, G has the following properties.*

(1) *There is a unique node $o \in |G|$ such that $G \models l(r^G,\ o)$. This node is denoted by o_l.*
(2) *There exists a unique node $o_K \in |G|$ such that $G \models l \cdot K(r^G,\ o_K)$. In addition, $o_K = o_l$ if $G \models \forall\, x\,(l(r,x) \rightarrow \forall\, y\,(\epsilon(x,y) \rightarrow K(x,y)))$.*
(3) *For every $\alpha \in \Gamma_0^*$, the following statements hold.*
   (a) *There is a unique $o \in |G|$ such that $G \models a \cdot \alpha(o_l, o)$. This node is denoted by o_α.*
   (b) *For every $o \in |G|$, if $G \models C^G(o)$ then there is a unique $o' \in |G|$ such that $G \models \alpha(o, o')$.*

Proof: This is immediate from the type constraint $\Phi(\Delta_1)$. □

LEMMA 6.7. *For any $G \in \mathcal{U}(\Delta_1)$ and $\alpha, \beta \in \Gamma_0^*$, if $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha(x,y) \rightarrow \beta(x,y)))$ then $G \models \forall\ x\ (l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\beta(x,y) \rightarrow \alpha(x,y)))$. Similarly, if $G \models \forall\ x\ (l \cdot K(r,x) \rightarrow \forall\ y\ (a \cdot \alpha(x,y) \rightarrow a \cdot \beta(x,y)))$ then it must follow that $G \models \forall\ x\ (l \cdot K(r,x) \rightarrow \forall\ y\ (a \cdot \beta(x,y) \rightarrow a \cdot \alpha(x,y)))$.*

Proof: We show that if $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha(x,y) \rightarrow \beta(x,y)))$ then $G \models \forall\ x\ (l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\beta(x,y) \rightarrow \alpha(x,y)))$. The proof for the other statement is similar.

By Lemma 6.6 and $\Phi(\Delta_1)$, for each $o \in |G|$ such that $G \models l \cdot b \cdot mem(r^G, o)$, there is a unique node $o_1 \in |G|$ such that $G \models \alpha(o, o_1)$. Similarly, there is a unique $o_2 \in |G|$ such that $G \models \beta(o, o_2)$. If $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha(x,y) \rightarrow \beta(x,y)))$, then $o_1 = o_2$. Thus $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\beta(x,y) \rightarrow \alpha(x,y)))$. □

LEMMA 6.8. *For any structure $G \in \mathcal{U}(\Delta_1)$, if $G \models \Sigma$ then it must follow that for any $\alpha \in \Gamma_0^*$, $G \models l \cdot b \cdot mem(r^G,\ o_\alpha)$, where o_α is the unique node in $|G|$ such that $G \models l \cdot a \cdot \alpha(r^G,\ o_\alpha)$, as specified in Lemma 6.6. In addition, for every $o, o' \in |G|$ such that $G \models l \cdot b \cdot mem(r^G,\ o') \wedge \alpha(o',\ o)$, $G \models l \cdot b \cdot mem(r^G,\ o)$.*

This can be verified by a straightforward induction on $|\alpha|$.

LEMMA 6.9. *For any structure $G \in \mathcal{U}(\Delta_1)$ and any $\alpha, \beta \in \Gamma_0^*$, if $G \models \Sigma$ and $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha(x,y) \rightarrow \beta(x,y)))$, then it must be the case that $G \models \forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a \cdot \alpha(x,y) \rightarrow a \cdot \beta(x,y)))$.*

Proof: By Lemma 6.6 and $\Phi(\Delta_1)$, if $G \models \Sigma$ then there are unique nodes $o, o' \in |G|$ such that $G \models l \cdot K(r^G,\ o)$ and $G \models a(o, o')$. Denote these nodes by o_l and o_a, respectively. Again by Lemma 6.6, there is a unique node $o_\alpha \in |G|$ such that $G \models a \cdot \alpha(o_l, o_\alpha)$, and there is a unique node $o_\beta \in |G|$ such that $G \models a \cdot \beta(o_l, o_\beta)$.

Thus $G \models a(o_l,\ o_a) \wedge \alpha(o_a,\ o_\alpha) \wedge \beta(o_a,\ o_\beta)$. By Lemma 6.8, $G \models l \cdot b \cdot mem(r^G,\ o_a)$. As a result, if $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha(x,y) \rightarrow \beta(x,y)))$, then we must have $o_\alpha = o_\beta$. Hence $G \models \forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a \cdot \alpha(x,y) \rightarrow a \cdot \beta(x,y)))$. □

Now we are ready to prove Lemma 6.5.

Proof Lemma 6.5: We show the lemma for finite implication. The proof for implication is similar.

(*if*) Suppose $\Theta_0 \not\models_f (\alpha,\ \beta)$. We construct a structure $G \in \mathcal{U}_f(\Delta_1)$ such that $G \models \Sigma$ but $G \not\models \varphi_{(\alpha,\beta)}$.

By $\Theta_0 \not\models_f (\alpha,\ \beta)$, there exist a finite monoid $(M,\ \circ,\ 1)$ and a homomorphism $h : \Gamma_0^* \rightarrow M$ such that for any $i \in [1,n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. Using M and h, we define an equivalence relation $\approx$ in the same way as in the proof of Lemma 5.5. In addition, for each $\rho \in \Gamma_0^*$, let $\widehat{\rho}$ be defined as in the proof of Lemma 5.5. Similarly, we define C_{Θ_0}.

Using C_{Θ_0}, we define $G = (|G|,\ r^G,\ E^G,\ T^G)$ as shown in Fig. 8, as follows. (1) For each $\rho \in \Gamma_0^*$, let $o(\widehat{\rho})$ be a distinct node. In addition, let o_r, o_l and o_b be distinct nodes. Then we define the universe $|G| = \{o_r, o_b, o_l\} \cup \{o(\widehat{\rho}) \mid \widehat{\rho} \in C_{\Theta_0}\}$. (2) $r^G = o_r$. (3) The unary relations C^G, ${C_s}^G$ and $DBtype^G$ are defined as follows: $C^G = \{o(\widehat{\rho}) \mid \widehat{\rho} \in C_{\Theta_0}\}$, ${C_s}^G = \{o_b\}$, ${C_l}^G = \{o_l\}$ and $DBtype^G = \{o_r\}$. (4) The binary relations are populated as follows: let $G \models l(o_r,\ o_l)$, $G \models K(o_l,\ o_l)$, $G \models a(o_l, o(\widehat{\epsilon}))$, $G \models b(o_l, o_b)$, and for each $\widehat{\rho} \in C_{\Theta_0}$, let $G \models mem(o_b,\ o(\widehat{\rho}))$. In addition, for each $j \in [1,m]$, let $G \models l_j(o(\widehat{\rho}),\ o(\widehat{\rho \cdot l_j}))$. It should be noted that since $o_l \in {C_l}^G$, it is valid to have $G \models K(o_l,\ o_l)$. Moreover, since $\nu(C_l)$ is a record type, there is no $o \in |G|$ such that $o \neq o_l$ and $G \models K(o_l,\ o_l) \wedge K(o_l,\ o)$.

By the definition of G, it is easy to verify the following claims.

Claim 1: $G \in \mathcal{U}_f(\Delta_1)$.

Obviously $G \models \Phi(\Delta_1)$. As in the proof of Lemma 5.5, it can be shown that G is finite. Thus $G \in \mathcal{U}_f(\Delta_1)$.

Claim 2: $G \models \Sigma$.

From the construction of G immediately follows $G \models \Sigma_K$. To show $G \models \Sigma_r$, observe the following. First, by assumption, for every $i \in [1,n]$, $\alpha_i \approx \beta_i$. In addition, for every $\rho \in \Gamma_0^*$, $h(\rho \cdot \alpha_i) = h(\rho \cdot \beta_i)$. This is because h is a homomorphism. Therefore, $\rho \cdot \alpha_i \approx \rho \cdot \beta_i$. That is, $\widehat{\rho \cdot \alpha_i} = \widehat{\rho \cdot \beta_i}$. Second, by the construction of G, for any $o \in |G|$, if $G \models l \cdot b \cdot mem(o_r,\ o)$, then $o = o(\widehat{\rho})$ for some $\rho \in \Gamma_0^*$. Moreover, by Lemma 6.6, for each $\varrho \in \Gamma_0^*$, there is a unique $o' \in |G|$ such that $G \models \varrho(o(\widehat{\rho}),\ o')$. By a straightforward induction on $|\varrho|$, it can be shown that $o' = o(\widehat{\rho \cdot \varrho})$. Therefore, for each $o(\widehat{\rho})$ such that $G \models l \cdot b \cdot mem(o_r,\ o(\widehat{\rho}))$, $o(\widehat{\rho \cdot \alpha_i})$ is the unique node in $|G|$ such that $G \models \alpha_i(o(\widehat{\rho}),\ o(\widehat{\rho \cdot \alpha_i}))$. Similarly, we have $G \models \beta_i(o(\widehat{\rho}),\ o(\widehat{\rho \cdot \beta_i}))$. By $o(\widehat{\rho \cdot \alpha_i}) = o(\widehat{\rho \cdot \beta_i})$, we have $G \models \beta_i(o(\widehat{\rho}),\ o(\widehat{\rho \cdot \alpha_i}))$. Therefore, for each $i \in [1,n]$, $G \models \forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(\alpha_i(x,y) \rightarrow \beta_i(x,y)))$. Moreover, again from the construction of G follows $G \models \forall\, x\,(l(r,x) \rightarrow \forall\, y\,(\epsilon(x,y) \rightarrow K(x,y)))$. That is, $G \models \Sigma_r$.

Claim 3: $G \not\models \forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a \cdot \alpha(x,y) \rightarrow a \cdot \beta(x,y)))$.

As in Claim 2, we can show $G \models l \cdot K(o_r,\ o_l) \wedge a \cdot \alpha(o_l,\ o(\widehat{\alpha})) \wedge a \cdot \beta(o_l,\ o(\widehat{\beta}))$. In addition, $o(\widehat{\beta})$ is the unique node in $|G|$ such that $G \models a \cdot \beta(o_l,\ o(\widehat{\beta}))$. By

assumption, we have $\alpha \not\approx \beta$. Thus $\widehat{\alpha} \neq \widehat{\beta}$. That is, $o(\widehat{\alpha}) \neq o(\widehat{\beta})$. Therefore, we have $G \models a \cdot \alpha(o_l, o(\widehat{\alpha})) \wedge \neg a \cdot \beta(o_l, o(\widehat{\alpha}))$. As an immediate result, we have $G \not\models \forall\, x\,(l \cdot K(r, x) \rightarrow \forall\, y\,(a \cdot \alpha(x, y) \rightarrow a \cdot \beta(x, y)))$.

(*only if*) Suppose that there is $G \in \mathcal{U}_f(\Delta_1)$ such that $G \models \Sigma$ but $G \not\models \varphi_{(\alpha,\beta)}$. We show that there exist a finite monoid $(M, \circ, 1)$ and a homomorphism $h : \Gamma_0^* \rightarrow M$ such that for any $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$.

To do this, we define another equivalence relation on Γ_0^* as follows:

$$\rho \sim \varrho \quad \text{iff} \quad G \models \forall\, x\,(l \cdot b \cdot mem(r, x) \rightarrow \forall\, y\,(\rho(x, y) \rightarrow \varrho(x, y))).$$

By Lemma 6.7, it can be easily verified that $\sim$ is indeed an equivalence relation. In addition, for any $i \in [1, n]$, $\alpha_i \sim \beta_i$, but $\alpha \not\sim \beta$. More specifically, by $G \models \Sigma_r$ and Lemma 6.7, for any $i \in [1, n]$, we have $\alpha_i \sim \beta_i$. In addition, by $G \not\models \varphi_{(\alpha,\beta)}$, Lemma 6.7 and Lemma 6.9, $G \not\models \forall\, x\,(l \cdot b \cdot mem(r, x) \rightarrow \forall\, y\,(\alpha(x, y) \rightarrow \beta(x, y)))$. Therefore, $\alpha \not\sim \beta$. For every $\rho \in \Gamma_0^*$, let $[\rho]$ be the equivalence class of ρ with respect to $\sim$. Then clearly, for any $i \in [1, n]$, $[\alpha_i] = [\beta_i]$. But $[\alpha] \neq [\beta]$.

Using the notion of $\sim$, we define $M = \{[\rho] \mid \rho \in \Gamma_0^*\}$. We verify the following claim.

Claim 4: M is finite.

To show this, let $S_\rho = \{(x, y) \mid x, y \in |G|,\ G \models l \cdot b \cdot mem(r^G, x),\ G \models \rho(x, y)\}$ for every $\rho \in \Gamma_0^*$. In addition, let $S_G = \{S_\rho \mid \rho \in \Gamma_0^*\}$. By Lemma 6.6, S_ρ is a finite function from $|G|$ to $|G|$. Since $|G|$ is finite, there are finitely many such functions. Therefore, S_G is finite. Moreover, it is easy to verify that for all $\rho, \varrho \in \Gamma_0^*$, $\rho \sim \varrho$ iff $S_\rho = S_\varrho$. Consider a function $g : M \rightarrow S_G$ defined by $g : [\rho] \mapsto S_\rho$. Clearly, g is well-defined, total and injective. Therefore, because S_G is finite, M is also finite.

Next, we define a binary operation $\circ$ on M by $[\rho] \circ [\varrho] = [\rho \cdot \varrho]$. It is easy to verify the following.

Claim 5: $\circ$ is well-defined.

For all $\rho_1, \rho_2, \varrho_1, \varrho_2 \in \Gamma_0^*$ such that $\rho_1 \sim \rho_2$ and $\varrho_1 \sim \varrho_2$, we show $\rho_1 \cdot \varrho_1 \sim \rho_2 \cdot \varrho_2$. By Lemmas 6.6 and 6.8, for every $o \in |G|$ such that $G \models l \cdot b \cdot mem(r^G, o)$, there exists a unique $o_1 \in |G|$ such that $G \models \rho_1 \cdot \varrho_1(o, o_1)$. In addition, there is a unique $o' \in |G|$ such that $G \models \rho_1(o, o') \wedge \varrho_1(o', o_1)$. By $\rho_1 \sim \rho_2$, we have $G \models \rho_2(o, o')$. By Lemma 6.8 and $G \models l \cdot b \cdot mem(r^G, o) \wedge \rho_1(o, o')$, we have $G \models l \cdot b \cdot mem(r^G, o')$. Thus by $\varrho_1 \sim \varrho_2$, we also have $G \models \varrho_2(o', o_1)$. Hence $G \models \rho_2 \cdot \varrho_2(o, o_1)$. Therefore, $G \models \forall\, x\,(l \cdot b \cdot mem(r, x) \rightarrow \forall\, y\,(\rho_1 \cdot \varrho_1(x, y) \rightarrow \rho_2 \cdot \varrho_2(x, y)))$. That is, $\rho_1 \cdot \varrho_1 \sim \rho_2 \cdot \varrho_2$.

Claim 6: $\circ$ is associative, and $[\epsilon]$ is the identity for $\circ$. The proof is the same as the one found in the proof of Lemma 5.5.

By these claims, $(M, \circ, [\epsilon])$ is a finite monoid. We define $h : \Gamma_0^* \rightarrow M$ by $h : \rho \mapsto [\rho]$. As in the proof of Lemma 5.5, we can show that h is a homomorphism, and moreover, for any $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. Therefore, $\Theta_0 \not\models_f (\alpha, \beta)$.

This completes the proof of Lemma 6.5. □

7. EXTENSIONS

Other interesting results can be established by generalizing the proofs of the previous sections. In this section we present three of these extensions. First, we investi-

gate the (finite) implication problem for a mild generalization of word constraints in the context of $\mathcal{M}^+$. Second, we revisit the (finite) implication problems associated with P_c constraints in a mild variant of the object-oriented model $\mathcal{M}^+$, denoted by $\mathcal{M}^*$. Like $\mathcal{M}^+$, $\mathcal{M}^*$ also supports records, classes and recursive structures. However, it supports finite sets instead of sets. Finally, we provide a complete picture of unrestricted implication versus finite implication for path constraints in different settings.

7.1 More on path constraint implication in the context of $\mathcal{M}^+$

Observe that Lemma 5.3 in fact establishes the undecidability of the (finite) implication problem for a mild generalization of the class P_w of word constraints in the untyped context. The fragment, denoted by $P_w(\alpha)$, is defined to be $P_w \cup \{\delta(\psi, \alpha) \mid \psi \in P_w\}$, where α is a nonempty path, and δ is a function that given α and any word constraint $\psi = \forall\, x\, (\beta(r,\, x) \rightarrow \gamma(r,\, x))$, defines

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

In other words, $P_w(\alpha)$ consists of word constraints and word constraints extended with a common prefix α.

Given Theorem 5.2, one might wonder whether the undecidability result of Lemma 5.3 breaks down in the context of $\mathcal{M}^+$. Below we show that it is not the case.

THEOREM 7.1. *In the context of $\mathcal{M}^+$, the implication and finite implication problems for $P_w(\alpha)$ and therefore, for P_c, are undecidable.*

Proof: We give a reduction from the word problem for (finite) monoids to this problem. Recall the alphabet Γ_0 and functions Θ_0 given in Section 5.1. Using Γ_0, we define an $\mathcal{M}^+$ schema $\Delta_0 = (\mathcal{C},\, \nu,\, DBtype)$, where $\mathcal{C} = \{C, C_s\}$, ν is defined by:

$$C \mapsto [l_1 : C, \ldots, l_m : C], \qquad C_s \mapsto \{C\},$$

and $DBtype = [a : C,\, b : C_s]$, where $a, b \notin \Gamma_0$.

We encode Θ_0 in terms of a finite set Σ consisting of the following P_c constraints

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

We encode a test equation $(\alpha,\, \beta)$ with $\varphi_{(\alpha,\beta)} = \forall\, x(a \cdot \alpha(r, x) \rightarrow a \cdot \beta(r, x))$. Obviously, constraints of Σ and $\varphi_{(\alpha,\beta)}$ are in $P_w(b \cdot mem)$. A structure that satisfies $\Phi(\Delta_0)$ and Σ is shown in Fig. 9.

We reduce the word problem for monoids and the word problem for finite monoids to the problem of determining whether $\Sigma \models_{\Delta_0} \varphi_{(\alpha,\beta)}$ and $\Sigma \models_{(f,\Delta_0)} \varphi_{(\alpha,\beta)}$, respectively. The rest of the proof is similar to that of Theorem 6.2. □

7.2 Path constraint implication revisited in an object-oriented model M^*

We next investigate path constraint implication in the context of another object-oriented model, denoted by M^*. Schemata and database instances in $\mathcal{M}^*$ are

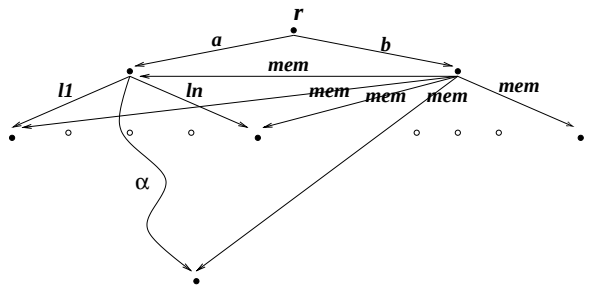

Fig. 9. A structure used in the proof of Theorem 7.1

defined in the same way as in $\mathcal{M}^+$, except the semantics of sets. More specifically, in a database instance (π, μ, d), the domain of a set $\{\tau\}$ is defined to be

$$[\![\{\tau\}]\!]_\pi \quad = \quad \{V \mid V \subseteq [\![\tau]\!]_\pi, \, V \text{ is finite}\},$$

where π is an oid assignment, μ maps oids to values, and d is the entry point into the database instance. Contrast this with the database instances defined in Section 3.1 for the model $\mathcal{M}^+$. We may describe the difference between $\mathcal{M}^+$ and $\mathcal{M}^*$ as follows. First, $\mathcal{M}^+$ and $\mathcal{M}^*$ support exactly the same collection of schemata Δ and, indeed, provide exactly the same collection of finite database instances for each such schema, since over finite databases, the distinction between finite and arbitrary subsets of the values of a given type lapses. The difference between $\mathcal{M}^+$ and $\mathcal{M}^*$ arises when we consider the $\mathcal{M}^+$ versus the $\mathcal{M}^*$ interpretation of the schema Δ over arbitrary structures. For the $\mathcal{M}^+$ interpretation of Δ over arbitrary structures we have chosen the collection $\mathcal{U}(\Delta)$, the set of all $\sigma(\Delta)$-structures satisfying the first-order condition $\Phi(\Delta)$ (type constraint, see Section 3.1). In analogy, as the $\mathcal{M}^*$ interpretation of Δ over arbitrary structures, we specify $\mathcal{U}^*(\Delta)$ to be the sub-collection of $\mathcal{U}(\Delta)$ which contains exactly those $\sigma(\Delta)$-structures all of whose elements have "finite *mem* out-degree", i.e., for any node of a set type in such a structure, there are *finitely* many *mem* edges going out of the node. In contrast to $\mathcal{U}(\Delta)$, a simple application of the Compactness Theorem for first-order logic [Enderton 1972] shows that $\mathcal{U}^*(\Delta)$ is not the collection of models of any set of first-order sentences. Now, on the basis of the first-order definability of $\mathcal{U}(\Delta)$, we could conclude immediately from the Completeness Theorem for first-order logic that the unrestricted implication problem for path constraints in $\mathcal{M}^+$ is recursively enumerable. It follows at once, that if the unrestricted implication and finite implication problems in $\mathcal{M}^+$ for a given collection of path constraints are equivalent, then both problems are decidable, since the latter problem is patently co-r.e. In the case of $\mathcal{M}^*$, no such conclusion can be drawn immediately, since we are not in a position to conclude, without further argument, that the unrestricted implication problem for a given collection of path constraints in $\mathcal{M}^*$ is recursively enumerable.

It can be shown that the results developed for $\mathcal{M}^+$ also hold for $\mathcal{M}^*$:

THEOREM 7.2. *In the context of $\mathcal{M}^*$, the implication and finite implication problems for $P_w(\alpha)$, for P_c, and for local inclusion constraints are all undecidable.*

The proofs of some of these results, however, are quite different from the anal-

ogous proofs for $\mathcal{M}^+$ for the reason mentioned above. More specifically, with the same proofs of Theorems 6.2 and 7.1 we can establish the undecidability of the finite implication problem for local inclusion constraints and the finite implication problem for $P_w(\alpha)$ in the context of $\mathcal{M}^*$. However, for unrestricted implication problems the technique no longer applies because $\mathcal{M}^*$ does not allow infinite sets. To establish the undecidability of the unrestricted implication problems, it suffices to show the lemma below, that is, in $\mathcal{M}^*$, the finite implication problem and the unrestricted implication problem for any P_c constraints coincide. From the observation above, we can see that the equivalence of the unrestricted implication problem and the finite implication problem for path constraints in $\mathcal{M}^*$ does not lead to the decidability of these problems. From this and the proofs of Theorems 6.2 and 7.1 immediately follows Theorem 7.2. It is worth mentioning that the lemma does not hold in the context of $\mathcal{M}^+$.

LEMMA 7.3. *For any schema Δ_0 in $\mathcal{M}^*$, any finite set Σ and φ of P_c constraints over Δ_0, if $\bigwedge \Sigma \wedge \neg\varphi$ has a model in $\mathcal{U}^*(\Delta_0)$, then it has a model in $\mathcal{U}_f^*(\Delta_0)$, where $\mathcal{U}^*(\Delta_0)$ (resp. $\mathcal{U}_f^*(\Delta_0)$) is the set of all (resp. finite) $\sigma(\Delta_0)$-structures that satisfy the type constraint $\Phi(\Delta_0)$ determined by Δ_0.*

Proof: Given a model G of $\bigwedge \Sigma \wedge \neg\varphi$ in $\mathcal{U}^*(\Delta_0)$, we construct a finite structure G' such that $G' \in \mathcal{U}_f^*(\Delta_0)$ and $G' \models \bigwedge \Sigma \wedge \neg\varphi$.

Recall the notion of k-neighborhood defined in the proof of Lemma 5.6. Also recall the notations $pf(\varphi)$, $lt(\varphi)$ and $rt(\varphi)$ from Definition 2.1. Let

$$k = max\{|pf(\psi)| + |lt(\psi)| + |rt(\psi)| \mid \psi \in \Sigma \cup \{\varphi\}\} \quad + \quad 1,$$

and let G_k be the k-neighborhood of G. We construct G' as follows. For each $\tau \in T(\Delta_0)$, let $o(\tau)$ be a distinct node, and let $G' = (|G'|, r^{G'}, E^{G'}, T^{G'})$, where the universe $|G'| = |G_k| \cup \{o(\tau) \mid \tau \in T(\Delta_0)\}$; the root $r^{G'} = r^{G_k}$; for each type $\tau \in T(\Delta_0)$, $\tau^{G'} = (\tau^G \cap |G_k|) \cup \{o(\tau)\}$; and the edge relation $E^{G'}$ is E^{G_k} augmented with the following:

—for each $o \in \tau^G \cap |G_k|$, if $\tau = [l_1 : \tau_1, \ldots l_n : \tau_n]$ (or for some class A in D_0, $\tau = A$ and $\nu(A) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), then for any $i \in [1, n]$ and any $o' \in |G_k|$ such that $G_k \not\models l_i(o, o')$, let $G' \models l_i(o, o(\tau_i))$;

—for each type $\tau \in T(\Delta_0)$, if $\tau = [l_1 : \tau_1, \ldots, l_n : \tau_n]$ (or for some class A in D_0, $\tau = A$ and $\nu(A) = [l_1 : \tau_1, \ldots, l_n : \tau_n]$), then $G' \models l_i(o(\tau), o(\tau_i))$ for each $i \in [1, n]$.

We now show that G' is indeed the structure desired.

(1) $G' \in \mathcal{U}_f^*(\Delta_0)$. Since $G \in \mathcal{U}^*(\Delta_0)$, each node in $|G|$ has finitely many outgoing edges. Hence by the definition of G_k, $|G_k|$ is finite. In addition, $T(\Delta_0)$ is finite. Therefore, by the construction of G', $|G'|$ is finite. In addition, it can be easily verified that $G' \models \Phi(\Delta_0)$.

(2) $G' \models \bigwedge \Sigma \wedge \neg\varphi$. The following can be easily verified by *reductio*:

Claim: For any $\sigma(\Delta_0)$-structure G'', if G_k is the k-neighborhood of G'', then $G'' \models \bigwedge \Sigma \wedge \neg\varphi$ iff $G_k \models \bigwedge \Sigma \wedge \neg\varphi$.

By the claim, it suffices to show that G_k is also the k-neighborhood of G'. To do this, assume for *reductio* that there exists $o(\tau) \in |G'|$ and $\alpha \in Paths(\Delta_0)$

such that $|\alpha| \leq k$ and $G' \models \alpha(r, o(\tau))$. Observe that $o(\tau) \notin |G_k|$. Without loss of generality, assume that α has the shortest length among all such paths. Then by the construction of G', there is $o \in |G_k|$ such that (i) $\alpha = \alpha' \cdot l$ and $G' \models \alpha'(r^{G'}, o) \wedge l(o, o(\tau))$; (ii) there is $\tau \in T(\Delta_0)$ such that $\tau = [l : \tau, ...]$ (or for some class A, $\tau = A$ and $\nu(A) = [l : \tau, ...]$), $o \in \tau^G$, and for any $o'' \in |G_k|$, $G_k \not\models l(o, o'')$; and (iii) $G_k \models \alpha'(r^{G'}, o)$. Note that for each $\tau \in T(\Delta_0)$, $o(\tau)$ does not have any outgoing edge to any node of $|G_k|$. By $G \in \mathcal{U}^*(\Delta_0)$, there is $o' \in |G|$ such that $G \models l(o, o')$. By the argument above, $o' \notin |G_k|$. Hence by the definition of k-neighborhood, there is no path $\beta \in Paths(\Delta_0)$ such that $|\beta| < k$ and $G \models \beta(r, o) \wedge l(o, o')$. Therefore, α' must have a length of at least k. That is, $|\alpha| > k$. This contradicts the assumption. Hence G_k is indeed the k-neighborhood of G'. Thus by the claim, $G' \models \bigwedge \Sigma \wedge \neg\varphi$.

Therefore, G' is indeed the structure desired. This proves Lemma 7.3. □

7.3 Finite vs. unrestricted implications

Given Lemma 7.3, one may wonder whether unrestricted implication and finite implication are equivalent or not for path constraints in other settings. Corollary 5.7 has shown that in the context of $\mathcal{M}$, these problems coincide for P_c (thus also for $P_w(\alpha)$ and local inclusion constraints). The corollary below tells us that it is also the case for local inclusion constraints in the untyped setting.

COROLLARY 7.4. *In the context of semistructured data, the unrestricted implication and finite implication problems for local inclusion constraints coincide.*

Proof: By Lemma 6.4, given any finite set Σ and φ of local inclusion constraints, there exist a finite set Σ' and ϕ of word constraints such that $\Sigma \models \varphi$ iff $\Sigma' \models \phi$, and $\Sigma \models_f \varphi$ iff $\Sigma' \models_f \phi$. It is known that unrestricted implication and finite implication for word constraints are equivalent in the untyped context [Abiteboul and Vianu 1999]: $\Sigma' \models \phi$ iff $\Sigma' \models_f \phi$. It follows at once that the same is true for local inclusion constraints. Indeed, clearly if $\Sigma \models \varphi$ then $\Sigma \models_f \varphi$. Conversely, if $\Sigma \models_f \varphi$, then $\Sigma' \models_f \phi$ and thus $\Sigma' \models \phi$. This yields $\Sigma \models \varphi$. □

For $P_w(\alpha)$ and P_c, however, these problems are not equivalent in the untyped setting.

COROLLARY 7.5. *In the context of semistructured data, unrestricted implication and finite implication of $P_w(\alpha)$ (and thus P_c) constraints differ from each other.*

Proof: By the Completeness Theorem for first-order logic, the unrestricted implication problem is recursively enumerable, while the finite implication problem is easily seen to be co-r.e. Therefore, the equivalence of these problems would imply the decidability of each, contradicting Lemma 5.3. □

Below we provide a concrete example to witness Corollary 7.5: we give a set Σ and φ of $P_w(\alpha)$ constraints such that $\Sigma \models_f \varphi$ but $\Sigma \not\models \varphi$.

Let Σ consist of the following:

(1) $\forall x\,(\epsilon(r, x) \rightarrow K(r, x))$,
(2) $\forall x\,(K \cdot S(r, x) \rightarrow K(r, x))$,

(3) $\forall x\,(K(r,x) \rightarrow \forall y\,(\epsilon(x,y) \rightarrow S \cdot S^-(x,y)))$,
(4) $\forall x\,(K(r,x) \rightarrow \forall y\,(S \cdot S^-(x,y) \rightarrow \epsilon(x,y)))$,
(5) $\forall x\,(K(r,x) \rightarrow \forall y\,(S^- \cdot S(x,y) \rightarrow \epsilon(x,y)))$.

Let φ be $\forall x\,(\epsilon(r,x) \rightarrow S^- \cdot S(r,x))$. Observe that these constraints are in $P_w(K)$: φ and constraints (1) and (2) are word constraints, whereas constraints (3), (4) and (5) are word constraints extended with a common prefix K. Intuitively, these constraints encode natural numbers: the root r for 0, S for the successor function, and S^- for the predecessor function. Indeed, in a structure satisfying Σ, constraints (1), (2) and (3) assert that every node has a successor and S^- is the inverse of S, and constraints (3), (4) and (5) ensure that S is an injective function. The negation of φ asserts that the root (0) does not have a predecessor.

To show $\Sigma \models_f \varphi$ but $\Sigma \not\models \varphi$, it suffices to prove that $\Psi = \bigwedge \Sigma \wedge \neg\varphi$ has a model but it does not have a finite model, i.e., Ψ is satisfiable but not finitely satisfiable. That is, the complement of the unrestricted implication problem for $P_w(\alpha)$ does not have the finite model property. Indeed, Ψ has an infinite model as depicted in Fig. 10 (a). We next show that it does not have a finite model. Assume for *reductio* that it has a finite model G. Then it is easy to verify that G must have a cycle consisting of S edges, which is reachable from the root by following (zero or more) S edges. Thus the cycle must have one of the following two cases: either the root r is on the cycle (and thus it has an incoming S edge), or there exists a node x on the cycle having two incoming S edges, one on the cycle and the other on the path from the root to the cycle. However, the root r cannot be on the cycle: if there exists a node x such that $G \models S(x,r)$, then by Σ, $G \models S^-(r,x)$ and thus $G \models S^- \cdot S(r,r)$. This violates $\neg\varphi$. Furthermore, if there is a node x on the cycle such that it has two incoming S edges, i.e., there exist distinct nodes y, z such that $G \models S(y,x) \wedge S(z,x)$, then by Σ, we have $G \models S^-(x,y) \wedge S^-(x,z)$. Thus $G \models S \cdot S^-(y,z)$, which violates constraint (4). This contradicts the assumption that G is a model of Ψ.

In contrast to Lemma 7.3, below we show that a slight change to the semantics of sets makes unrestricted and finite implications different problems.

COROLLARY 7.6. *In the context of the object-oriented data model $\mathcal{M}^+$, the unrestricted implication and finite implication problems for local inclusion constraints (resp. for $P_w(\alpha)$ and for P_c) do not coincide.*

Proof: Since any schema in $\mathcal{M}^+$ can be characterized with a first-order logic sentence (type constraint), the argument of Corollary 7.5 also applies here: Theorems 7.1 and 6.2 imply that unrestricted and finite implications are not equivalent in the context of $\mathcal{M}^+$ for $P_w(\alpha)$ (thus P_c) constraints and for local inclusion constraints. □

Below we give a concrete example to witness Corollary 7.6 for local inclusion constraints. Examples for $P_w(\alpha)$ (thus P_c) constraints can be constructed similarly.

We use a schema in $\mathcal{M}^+$ similar to the one constructed in the proof of Theorem 6.2: $\Delta = (\mathcal{C}, \nu, DBtype)$, where $DBtype = [l : C_l]$, $\mathcal{C} = \{C, C_s, C_l\}$, and ν is defined by $\mathcal{C} \mapsto [S : C,\ S^- : C]$, $C_s \mapsto \{C\}$, $C_l \mapsto [a : C, b : C_s, K : C_l]$. Let Σ be the set consisting of:

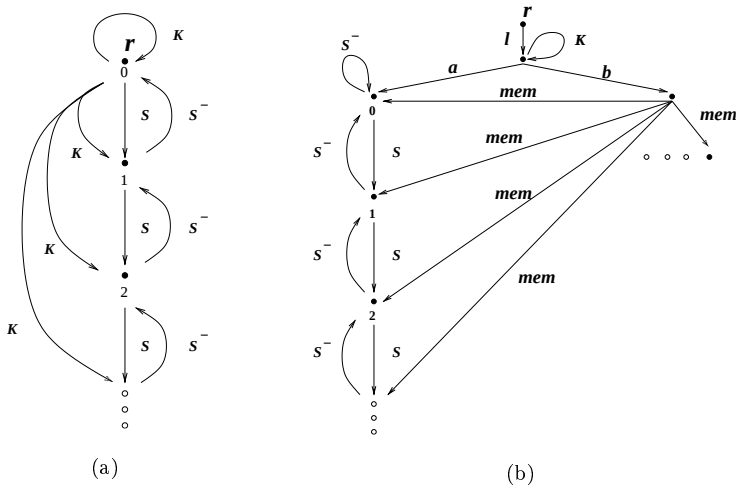

Fig. 10. Structures used in the proofs of Corollaries 7.5 and 7.6

(1) $\forall\, x\,(l(r,x) \rightarrow \forall\, y\,(\epsilon(x,y) \rightarrow K(x,y)))$,
(2) $\forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a(x,y) \rightarrow b \cdot mem(x,y)))$;
(3) $\forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(b \cdot mem \cdot S(x,y) \rightarrow b \cdot mem(x,y)))$;
(4) $\forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a(x,y) \rightarrow a \cdot S^-(x,y)))$,
(5) $\forall\, x\,(l \cdot b \cdot mem(r,x) \rightarrow \forall\, y\,(S \cdot S^-(x,y) \rightarrow \epsilon(x,y)))$,

Let φ be $\forall\, x\,(l \cdot K(r,x) \rightarrow \forall\, y\,(a(x,y) \rightarrow a \cdot S(x,y)))$. Observe that Σ and φ can be partitioned into Σ_K and Σ_r, where Σ_r consists of constraints (1) and (5), and Σ_K consists of all the others. Note that Σ_K is bounded by l and K whereas Σ_r is excluded by l and K.

As in the proof of Theorem 6.2, it is easy to show the following for an instance of Δ satisfying Σ. (1) There exist a unique C_s object and multiple C objects, each connected to the C_s object by a *mem* edge. In particular, there is a unique C object reachable from the root by following $l \cdot K \cdot a$, denoted by 0. The C_s object is reachable from the root by following $l \cdot b$. (2) By the definition of class C, each C object has a unique S (successor) edge and a unique S^- (predecessor) edge emanating from it; thus S and S^- can be viewed as total functions on C objects. (3) By Σ constraints, S^- is the inverse of S except at 0. (4) The negation of φ asserts that the successor of 0 is not 0 itself, and constraint (4) says that the S^- edge from 0 goes back to 0.

We show that $\Psi = \bigwedge \Sigma \wedge \neg\varphi$ is satisfiable but is not finitely satisfiable, i.e., $\Sigma \models_f \varphi$ but $\Sigma \not\models \varphi$. Indeed, Fig. 10 (b) gives an infinite model of Ψ. Assume for *reductio* that Ψ has a finite model G. Now, the same argument as given in the example witnessing Corollary 7.5 shows that G has a cycle consisting of S edges, which is reachable from the node 0 by following (zero or more) S edges. Again we must have one of the following two cases: either the node 0 is on the cycle (and thus it has an incoming S edge), or there exists a C object x on the cycle having

two incoming S edges, one on the cycle and the other on the path from the node 0 to the cycle. However, there cannot be a node x on the cycle having two distinct C objects y and z such that $G \models S(y,x) \wedge S(z,x)$. This is because x has a unique S^- outgoing edge and thus we cannot have both $G \models S^-(x,y)$ and $G \models S^-(x,z)$. Thus either $G \not\models S \cdot S^-(y,y)$ or $G \not\models S \cdot S^-(z,z)$, which violates constraint (5). Furthermore, the node 0 cannot be on the cycle either: if there exists a node x such that $G \models S(x,0)$, then by constraint (4), we have $G \models S \cdot S^-(x,0)$, which violates either $\neg\varphi$ (when $x = 0$) or constraint (5) (when $x \neq 0$). Thus G cannot be a finite model of Ψ.

Observe that the structure in Fig. 10 (b) is not a model of Ψ in the context of $\mathcal{M}^*$, because its C_s object has an infinite set as its value, which is not allowed by $\mathcal{M}^*$.

8. CONCLUSION

We have investigated the interaction between two forms of constraints that are important in specifying the semantics of data, namely, type constraints and path constraints. 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 [Buneman et al. 2000] interact with two type systems. One of the type systems, $\mathcal{M}^+$, is an object-oriented model similar to those studied in [Abiteboul et al. 1995; Abiteboul and Kanellakis 1989; Cattell 2000]. It supports classes, records, sets and recursive types. The other, $\mathcal{M}$, is a restriction of $\mathcal{M}^+$. On the one hand, we have shown that the unrestricted 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 unrestricted implication and finite implication problems for local inclusion 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. We establish some extensions of these results both for different constraints and for a different model. In particular, we show the undecidability of the foregoing (finite) implication problems in the context of an object-oriented model $\mathcal{M}^*$, a variant of $\mathcal{M}^+$ that supports finite sets instead of (unrestricted) sets. In addition, we establish the undecidability of (finite) implication in the contexts of $\mathcal{M}^+$ and $\mathcal{M}^*$, and in the untyped setting, for $P_w(\alpha)$, a mild generalization of the word constraints for which the PTIME decidability was established by Abiteboul and Vianu [1999] for semistructured data. We have also determined for all the implication problems we study, whether or not their finite and unrestricted versions coincide. The main results of the paper are summarized in Table I, which provides a complete picture for the complexity and comparison of unrestricted implication and finite implication of path constraints in different settings, along with references to the theorems where the results were proved.

Acknowledgments. We thank Leonid Libkin and Victor Vianu for comments and discussions. We are grateful to the referees for valuable suggestions.

	finite/unrestricted implication: $P_w(\alpha)$ constraints	*finite/unrestricted implication: local inclusion constraints*	*finite/unrestricted implication: P_c constraints*
semistructured data model	undecidable Lemma 5.3	decidable (PTIME) Theorem 6.1	undecidable Theorem 5.1
finite vs. restricted implications	differ Corollary 7.5	coincide Corollary 7.4	differ Corollary 7.5
object-oriented model $\mathcal{M}$	decidable, $O(n^3)$ Theorem 5.2	decidable, $O(n^3)$ Theorem 5.2	decidable, $O(n^3)$ Theorem 5.2
finite vs. restricted implications	coincide Corollary 5.7	coincide Corollary 5.7	coincide Corollary 5.7
object-oriented model $\mathcal{M}^+$	undecidable Theorem 7.1	undecidable Theorem 6.2	undecidable Theorem 7.1
finite vs. restricted implications	differ Corollary 7.6	differ Corollary 7.6	differ Corollary 7.6
object-oriented model $\mathcal{M}^*$	undecidable Theorem 7.2	undecidable Theorem 7.2	undecidable Theorem 7.2
finite vs. restricted implications	coincide Lemma 7.3	coincide Lemma 7.3	coincide Lemma 7.3

Table I. The main results of the paper

REFERENCES

Abiteboul, S., Buneman, P., and Suciu, D. 2000. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufman.

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Abiteboul, S. and Kanellakis, P. C. 1989. Object identity as a query language primitive. In *Proceedings of ACM SIGMOD Conference on Management of Data.* 159–173.

Abiteboul, S. and Vianu, V. 1999. Regular path queries with constraints. *Journal of Computer and System Sciences (JCSS) 53,* 3, 428–452.

Alon, N., Milo, T., Neven, F., Suciu, D., and Vianu, V. 2001. XML with data values: Typechecking revisited. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS).* 138–149.

Arenas, M., Fan, W., and Libkin, L. 2002. On verifying consistency of XML specifications. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS).* 259–270.

Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., and Chu, V. 1999. XML-based information mediation with MIX. In *Proceedings of ACM SIGMOD Conference on Management of Data.* 597–599.

Börger, E., Grädel, E., and Gurevich, Y. 1997. *The Classical Decision Problem.* Springer.

Bray, T., Paoli, J., and Sperberg-McQueen, C. M. 1998. Extensible Markup Language (XML) 1.0. W3C Recommendation. `http://www.w3.org/TR/REC-xml/`.

Buneman, P., Davidson, S., Fan, W., Hara, C., and Tan, W. 2001. Keys for XML. In *Proceedings of International World Wide Web Conference.* 201–210.

Buneman, P., Fan, W., and Weinstein, S. 1999. Query optimization for semistructured data

using path constraints in a deterministic data model. In *Proceedings of International Workshop on Database Programming Languages (DBPL).* 208–223.

Buneman, P., Fan, W., and Weinstein, S. 2000. Path constraints in semistructured databases. *Journal of Computer and System Sciences (JCSS) 61,* 2, 146–193.

Calvanese, D., De Giacomo, G., and Lenzerini, M. 1998. What can knowledge representation do for semi-structured data? In *Proceedings of National Conference on Artificial Intelligence (AAAI/IAAI).*

Calvanese, D., De Giacomo, G., Lenzerini, M., and Nardi, D. 2001. Reasoning in expressive description logics. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Elsevier Science Publishers, Chapter 23, 1581–1634.

Cattell, R. G. 2000. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann.

Davidson, A., Fuchs, M., Hedin, M., Jain, M., Koistinen, J., Lloyd, C., Maloney, M., and K. Schwarzhof. 1999. Schema for object-oriented XML 2.0. W3C Note. `http://www.w3.org/TR/NOTE-SOX`.

Deutsch, A., Fernandez, M. F., and Suciu, D. 1999. Storing semistructured data with STORED. In *Proceedings of ACM SIGMOD Conference on Management of Data.* 431–442.

Deutsch, A., Popa, L., and Tannen, V. 1999. Physical data independence, constraints, and optimization with universal plans. In *Proceedings of International Conference on Very Large Databases (VLDB).*

Enderton, H. B. 1972. *A Mathematical Introduction to Logic.* Academic Press.

Fan, W. and Libkin, L. 2001. On XML integrity constraints in the presence of DTDs. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS).* 114–125.

Florescu, D. and Kossmann, D. 1999. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin 22,* 3, 27–34.

Florescu, D., Raschid, L., and Valduriez, P. 1996. A methodology for query reformulation in CIS using semantic knowledge. *Int'l J. Cooperative Information Systems (IJCIS) 5,* 4, 431–468.

Ito, M. and Weddell, G. E. 1995. Implication problems for functional constraints on databases supporting complex objects. *Journal of Computer and System Sciences (JCSS) 50,* 1, 165–187.

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

Lee, D. and Chu, W. W. 2000. Constraints-preserving transformation from XML document type definition to relational schema. In *Proceedings of International Conference on Conceptual Modeling (ER).* 112–125.

Manolescu, I., Florescu, D., and Kossmann, D. 2001. Answering XML queries on heterogeneous data sources. In *Proceedings of International Conference on Very Large Databases (VLDB).*

Ramakrishnan, R. and Gehrke, J. 2000. *Database Management Systems.* McGraw-Hill Higher Education.

Rounds, W. C. 1997. Feature logics. In *Handbook of Logic and Language.* Elsevier.

Schmidt-Schauss, M. 1989. Subsumption in KL-ONE is undecidable. In *Proceedings of the 1st International Conference on the Principles of Knowledge Representation and Reasoning (KR'89).* 421–431.

Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D. J., and Naughton, J. F. 1999. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of International Conference on Very Large Databases (VLDB).*

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. 2001. XML Schema Part 1: Structures. W3C Recommendation. `http://www.w3.org/TR/xmlschema-1/`.

van Bommel, M. F. and Weddell, G. E. 1994. Reasoning about equations and functional dependencies on complex objects. *IEEE Transactions on Data and Knowledge Engineering 6,* 3, 455–469.

Received August 2001; revised May 2002; accepted May 2002.