Query Optimization for Semistructured Data using Path Constraints in a Deterministic Data Model

Query Optimization for Semistructured Data using Path Constraints in a Deterministic Data Model

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

Wenfei Fan[†]
Temple University
fan@joda.cis.temple.edu

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

Abstract

Path constraints have been studied in [4, 11, 12, 13] for semistructured data modeled as a rooted edge-labeled directed graph. They have proven useful in the optimization of path queries. However, in this graph model, the implication problems associated with many natural path constraints are undecidable [11, 13]. A variant of the graph model, called *the deterministic data model*, was recently proposed in [10]. In this model, data is represented as a graph with deterministic edge relations, i.e., the edges emanating from any node in the graph have distinct labels. The deterministic graph model is more appropriate for representing, for example, ACeDB [27] databases and Web sites.

This paper investigates path constraints for the deterministic data model. It demonstrates the application of path constraints to, among other things, query optimization. Three classes of path constraints are considered: the language P_c introduced in [11], an extension of P_c, denoted by P_c^w, by including wildcards in path expressions, and a generalization of P_c^w, denoted by P_c^*, by representing paths as regular expressions. The implication problems for these constraint languages are studied in the context of the deterministic data model. It shows that in contrast to the undecidability result of [11], the implication and finite implication problems for P_c are decidable in cubic-time and are finitely axiomatizable. Moreover, the implication problems are decidable for P_c^w. However, the implication problems for P_c^* are undecidable.

1 Introduction

Semistructured data is usually modeled as an edge-labeled rooted directed graph [1, 8]. Let us refer to this graph model as the *semistructured data model* (SM). For data found in many applications, the graph is *deterministic*, i.e., the edges emanating from each node in the graph have distinct labels. For example, when modeling Web pages as a graph, a node stands for an HTML document and an edge represents a link with an HTML label from one document (source) to another (target). It is reasonable to assume that the HTML label uniquely identifies the target document. Even if this is not literally the case, one can achieve this by including the URL (Universal Resource Locator) of the target document in the edge label. This yields a deterministic graph. As another example, consider ACeDB [27], which is a database management system popular with biologists. A graph representing an ACeDB database is also deterministic. In general, any database with "exportable" data identities can be modeled as a deterministic graph by including the identities in the edge labels. Here by exportable identities we mean directly observable identities such as keys. Some relational and object-oriented database management systems support exportable identities. In particular, in the OEM model (see, e.g., [3]), there are exportable object identities. To capture this, we consider a data model for semistructured data which is a variant of SM, referred to as *the deterministic data model* (DM). In DM, data is represented as a deterministic, rooted, edge-labeled, directed graph. An important feature of DM is that in this model, each component of a database is uniquely identified by a path.

A number of query languages (e.g., [3, 9, 24]) have been developed for semistructured data. The study of semistructured data has also generated the design of query languages for XML (eXtensible Markup Language [7]) documents (e.g., [17]). In these languages, queries are described in terms of navigation paths. To optimize path queries, it often appears necessary to use structural information about the data described by path constraints. Path constraints are capable of expressing natural integrity constraints that are a fundamental part of the semantics of the data, such as inclusion dependencies and inverse relationships. In traditional structured databases such as object-oriented databases, this semantic information is described in schemas. Unlike structured databases, semistructured data does not have a schema, and path constraints are used to convey the semantics of the data. The approach to querying semistructured data with path constraints was proposed in [4] and later studied in [11, 12, 13]. Several proposals (e.g., [6, 19, 21, 22]) for adding structure or type systems to XML data also advocate the need for integrity constraints that can be expressed as path constraints.

To use path constraints in query optimization, it is important to be able to reason about them. That is,

[*]This work was partly supported by the Army Research Office (DAAH04-95-1-0169) and NSF Grant CCR92-16122.
[†]Supported in part by Temple University.
[‡]Supported by NSF Grant CCR-9403447.

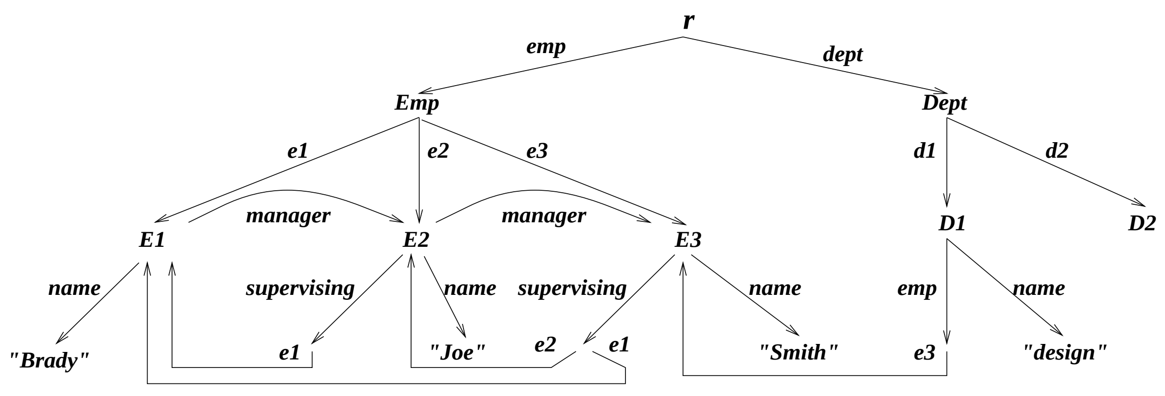

Figure 1: An example semistructured database in DM

we need to settle the question of constraint implication: given that certain constraints are known to hold, does it follow that some other constraint is necessarily satisfied? In the context of databases, only finite instances (graphs) are considered, and constraint implication is referred to as *finite implication.* In the traditional logic framework, both infinite and finite instances (graphs) are permitted, and constraint implication is called *unrestricted implication* or simply *implication.* For the graph model SM, it has been shown that the implication problems associated with many natural integrity constraints are undecidable. For example, the implication problem for the simple constraint language P_c studied in [11, 12, 13] is r.e. complete, and the finite implication problem for P_c is co-r.e. complete [11, 13]. In addition, we have already studied the connection between object-oriented databases and semistructured databases in SM with P_c constraints in [12]. The results of [12] show that the connection is not simple.

In this paper, we investigate path constraints for the deterministic data model DM. We demonstrate applications of path constraints to semantic specification and query optimization, and study the implication problems associated with path constraints. We show that in contrast to the undecidability result of [11, 13], the implication and finite implication problems for P_c are decidable in cubic-time and are finitely axiomatizable in the context of DM. That is, there is a finite set of inference rules that is sound and complete for implication and finite implication of P_c constraints, and in addition, there is an algorithm for testing P_c constraint implication in time $O(n^3)$, where n is the length of constraints. This demonstrates that the determinism condition of DM simplifies the analysis of path constraint implication. We also introduce and investigate two generalizations of P_c. One generalization, denoted by P_c^w, is defined by including wildcards in path expressions. The other, denoted by P_c^*, represents paths by regular expressions. We show that in the context of DM, the implication and finite implication problems for P_c^w are also decidable. However, the implication and finite implication problems for P_c^* are undecidable in the context of DM. This undecidability result shows that the determinism condition of DM does not reduce the analysis of path constraint implication to a trivial problem.

The rest of the paper is organized as follows. Section 2 uses an example to illustrate how path constraints can be used in query optimization. Section 3 reviews the definition of P_c constraints proposed in [11], and introduces two extensions of P_c, namely, P_c^w and P_c^*. Section 4 studies the implication and finite implication problems for P_c, P_c^w and P_c^* for the deterministic data model. Finally, Section 5 identifies open problems and directions for further work. A cubic-time algorithm for testing implication and finite implication of P_c constraints is given in an Appendix.

2 An example

To demonstrate applications of path constraints, let us consider Figure 1, which collects information on employees and departments. It is an example of semistructured data represented in the deterministic data model. In Figure 1, there are two edges emanating from root node r, which are labeled `emp` and `dept` and connected to nodes `Emp` and `Dept`, respectively. Edges emanating from `Emp` are labeled with employee ID's and connected to vertices representing employees. An employee node may have three edges emanating from it: an edge labeled `manager` and connected to his/her manager, an edge labeled `supervising` that connects to a node from which there are outgoing edges connected to employees under his/her supervision, and an edge labeled `name`. Similarly, there are vertices representing departments that may have edges connected to employees. Observe that Figure 1 is deterministic.

Path constraints. Typical path constraints on Figure 1 include:

$$\forall x\,(emp \cdot \_ \cdot manager(r, x) \to emp \cdot \_\,(r, x)) \qquad (\phi_1)$$

$$\forall x\,(emp \cdot \_ \cdot supervising \cdot \_\,(r, x) \to emp \cdot \_\,(r, x)) \qquad (\phi_2)$$

$$\forall x\,(emp \cdot \_\,(r, x) \to \forall y\,(manager(x, y) \to supervising \cdot \_\,(y, x))) \qquad (\phi_3)$$

Here r is a constant denoting the root of the graph, variables x and y range over vertices, and "_" is a "wildcard" symbol, which matches any edge label. A path in the graph is a sequence of edge labels, which can be expressed as a logic formula $\alpha(x, y)$ that holds in the graph if α is a sequence of edge labels from vertex x to y. For example, $\texttt{emp} \cdot \texttt{e1} \cdot \texttt{manager}$ can be expressed as a logic formula, which holds in Figure 1. Path formulas can be naturally generalized to include wildcards. The path constraints above describe inclusion relations. More specifically, ϕ_1 states that if a node is reached from the root r by following $\texttt{emp} \cdot \_ \cdot \texttt{manager}$, then it is also reachable from r by following $\texttt{emp} \cdot \_$. It asserts that the manager of any employee is also an employee that occurs in the database. Similarly, ϕ_2 states that if a node is reached from r by following $\texttt{emp} \cdot \_ \cdot \texttt{supervising} \cdot \_$, then it is also reachable from r by following $\texttt{emp} \cdot \_$. Constraint ϕ_3 states that for any employee x and for any y, if x is connected to y by a `manager` edge, then x is reachable from y by following $\texttt{supervising} \cdot \_$. These are constraints of P_c^w, one of the path constraint languages studied in this paper.

We generalize P_c^w by representing paths as regular expressions. This generalization is denoted by P_c^*. For example, the following are constraints of P_c^*:

$$\begin{aligned} \forall x\,(emp \cdot \_\,(r, x) \to \forall y\,(manager \cdot manager^*(x, y) \\ \to supervising \cdot \_\,(y, x))) \qquad (\psi_1) \\ \forall x\,(emp \cdot \_\,(r, x) \to \forall y\,(supervising \cdot \_\,(x, y) \\ \to manager \cdot manager^*(y, x))) \qquad (\psi_2) \end{aligned}$$

Here $*$ is the Kleene star. These constraints describe an inverse relationship between $\texttt{manager} \cdot \texttt{manager}^*$ and $\texttt{supervising} \cdot \_$. More specifically, ψ_1 asserts that for any employee x and for any y, if y is reachable from x by following one or more `manager` edges, then x is reachable from y by following path $\texttt{supervising} \cdot \_$. Similarly, ψ_2 asserts that if y is reachable from x by following $\texttt{supervising} \cdot \_$, then x is reachable from y by following one or more `manager` edges.

A subclass of P_c^*, P_c, has been investigated in [11, 12, 13] for the graph model *SM* for semistructured data. As opposed to P_c^* constraints, path constraints of P_c contain neither wildcards nor the Kleene star. In the deterministic data model, P_c constraints express path equalities. For example, the following can be described by P_c constraints:

$$\begin{aligned} emp \cdot e1 \cdot manager &= emp \cdot e2 \qquad (\varphi_1) \\ dept \cdot d1 \cdot emp \cdot e1 &= emp \cdot e1 \qquad (\varphi_2) \end{aligned}$$

Observe that the paths in the constraints above contain neither wildcards nor the Kleene closure.

Semantic specification with path constraints. The path constraints above describe certain typing information about the data. For example, abusing object-oriented database terms, ϕ_1 asserts that a manager of an employee has an "`employee` type", and in addition, is in the "extent" of "class" `employee`. By using ϕ_1, it can be shown that for any employee x and any y, if y is reachable from x by following zero or more `manager` edges, then y also has an "`employee` type" and is in the "extent" of `employee`. A preliminary type system was proposed in [10] for the deterministic data model, in which the types of paths are defined by means of path constraints. This is a step towards unifying the (programming language) notion of a type with the (database) notion of a schema.

Query optimization with path constraints. To illustrate how path constraints can be used in query optimization, consider again the database represented in Figure 1. Suppose, for example, we want to find the name of the employee with ID $e1$ in department $d1$. One may write the query as Q_1 (in Lorel syntax [3]):

```
Q1:     select X.name
        from   r.dept.d1.emp.e1 X
```

Given path constraint φ_2, the query Q_1 can be rewritten as Q_1':

```
Q1':    select X.name
        from   r.emp.e1 X
```

One can easily verify that Q_1 and Q_1' are equivalent.

As another example, suppose we want to find the names of the employees connected to Smith by one or more `manager` edges. Without path constraints, one would write the query as Q_2 (in Lorel syntax):

```
Q2:     select X.name
        from   r.emp.% X, X(.manager)+ Y
        where  Y.name = "Smith"
```

In Lorel, `%` denotes wildcard and `(.manager)+` means one or more occurrences of `.manager`. Given constraints ψ_1, ψ_2, ϕ_1 and ϕ_2, we can rewrite Q_2 as Q_2', which finds the names of the employees under the supervision of Smith:

```
Q2':    select X.name
        from   r.emp.% Y, Y.supervising.% X
        where  Y.name = "Smith"
```

It can be verified that given those path constraints, Q_2 and Q_2' are equivalent. In addition, Q_2' is more efficient than Q_2 because it does not require the traversal of sequences of `manager` edges. It should be mentioned that to show Q_2 and Q_2' are equivalent, we need to verify that certain constraints necessarily hold given that ψ_1, ψ_2, ϕ_1 and ϕ_2 hold. That is, they are implied by ψ_1, ψ_2, ϕ_1 and ϕ_2. In particular, we need to show that ψ_3 below is implied by ψ_1, ψ_2, ϕ_1 and ϕ_2:

$$\forall x\,(emp \cdot \_ \cdot manager^*(r, x) \to emp \cdot \_\,(r, x)) \qquad (\psi_3)$$

Related work. A more general deterministic data model, *DDM*, was proposed in [10]. In *DDM*, edge labels also have structure, and a number of database operations may be obtained by manipulation of this structure. In particular, annotations can be described in this structure for the purpose of data provenance, i.e., to keep track by what process some piece of data got into the database. To simplify the discussion we do not consider this general model here.

Path constraints have been studied in [4, 11, 12, 13]. The constraints of [4] have either the form $p \subseteq q$ or $p = q$, where p and q are regular expressions representing paths. These constraints were investigated for the graph model SM for semistructured data. The decidability of the implication problems for this form of constraints was established in [4] in the context of SM. Another path constraint language, P_c, was introduced and studied in [11] for SM. It was shown there that despite the simple syntax of P_c, its associated implication and finite implication problems are undecidable in the context of SM. The details of the proofs of these undecidability results can be found in [13]. The interaction between P_c constraints and type systems was investigated in [12]. However, none of these papers has considered the deterministic data model. In addition, path constraint languages P_c^w and P_c^* were not studied in these papers.

Recently, the application of integrity constraints to query optimization was also studied in [25]. Among other things, [25] developed an equational theory for query rewriting by using a certain form of constraints.

The connection between semistructured databases in SM with P_c constraints and object-oriented databases has been studied in [12]. Object-oriented databases are constrained by types, e.g., class types with single-valued and set-valued attributes, whereas databases in SM are in general free of these type constraints. These types cannot be expressed as path constraints and *vice versa*. As an example, it has been shown in [12] that there is a P_c constraint implication problem that is decidable in PTIME in the context of SM, but that becomes undecidable when an object-oriented type system is added. On the other hand, there is a P_c constraint implication problem that is undecidable in the context of SM, but becomes decidable in PTIME when an object-oriented type system is imposed.

There is a natural analogy between the work on path constraints and inclusion dependency theory developed for relational databases. Path constraints specify inclusions among certain sets of objects, and can be viewed as a generalization of inclusion dependencies. Inclusion dependencies have proven useful in semantic specification and query optimization for relational databases. In the same way, path constraints are important in a variety of database contexts, ranging from semistructured data to object-oriented databases. It should be mentioned that the path constraints considered in this paper are not expressible in any class of dependencies studied for relational databases, including inclusion and tuple-generating dependencies [5]. See [2] for in-depth presentations of dependency theories.

The results established on path constraint implication in this paper may find applications to other fields. Indeed, if we view vertices in a graph as states and labeled edges as actions, then the deterministic graphs considered here are in fact Kripke models studied in deterministic propositional dynamic logic (DPDL. See, e.g., [20, 28]), which is a powerful language for reasoning about programs. These deterministic graphs may also be viewed as feature structures studied in feature logics [26]. It should be mentioned that DPDL and feature logics are modal logics, in which our path constraints are not expressible.

Description logics (see, e.g., [16]) reason about concept subsumption, which can be expressed as inclusion assertions similar to path constraints. There has been work on specifying constraints on semistructured data by means of description logics [15]. One of the most expressive description logics used in the database context is $\mathcal{ALCQI}_{reg}$ [16], which allows negation, conjunction, disjunction, qualified universal and existential quantification, qualified number restriction, and in addition, provides constructs to form regular expressions such as role union, role concatenation, transitive closure and role identity. It is known that $\mathcal{ALCQI}_{reg}$ corresponds to propositional dynamic logic (PDL) with converse and graded modalities [16, 20]. We should remark here that our path constraints are not expressible in $\mathcal{ALCQI}_{reg}$.

3 Deterministic graphs and path constraints

In this section, we first give an abstraction of semistructured databases in DM in terms of first-order logic, and then present three path constraint languages: P_c, P_c^w and P_c^*.

3.1 The deterministic data model

In the graph model SM, a database is represented as an edge-labeled rooted directed graph [1, 8]. An abstraction of databases in SM has been given in [11] as (finite) first-order logic structures of a relational signature

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

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

In the deterministic data model DM, a database is represented as an edge-labeled rooted directed graph with deterministic edge relations. That is, for any edge label K and node a in the graph, there exists at most one edge labeled K going out of a. Along the same lines of the abstraction of databases in SM, we represent a database in DM as a (finite) σ-structure satisfying the *determinism condition*:

$$\bigwedge_{K \in E} \forall\, x\, y\, z\, (K(x,\ y) \wedge K(x,\ z) \rightarrow y = z).$$

Such structures are called *deterministic structures*. A deterministic structure G is specified by $(|G|,\ r^G,\ E^G)$, where $|G|$ is the set of nodes in G, r^G is the root node, and E^G is the set of binary relations on $|G|$, each of which is named by a relation symbol of E.

3.2 Path constraint language P_c

Next, we review the definition of P_c constraints introduced in [11]. To do this, we first present the notion of paths.

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

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

Here ϵ is the empty path, $K \in E$, and $\cdot$ denotes path concatenation. Paths defined above are the simplest form of path expressions. We shall present more general forms of path expressions shortly in this section.

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

We have seen many examples of paths in Section 2. Among them are:

$$\begin{array}{l} \mathit{emp} \cdot e1 \cdot \mathit{manager} \\ \mathit{dept} \cdot d1 \cdot \mathit{emp} \cdot e1 \end{array}$$

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. For example, the paths above can be described by the following formulas:

$$\begin{array}{l} \exists\, z\, (\mathit{emp}(x, z) \land \exists\, w\, (e1(z, w) \land \mathit{manager}(w, y))) \\ \exists\, z\, (\mathit{dept}(x, z) \land \exists\, w\, (d1(z, w) \land \exists\, u\, (\mathit{emp}(w, u) \land \\ \quad e1(u, y)))) \end{array}$$

We write $\rho(x, y)$ as ρ when the parameters x and y are clear from the context.

By treating paths as logic formulas, we are able to borrow the standard notion of models from first-order logic [18]. Let G be a deterministic 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 $\rho(a, b)$ holds in G, i.e., there is a path ρ from a to b in G.

The *length* of path ρ, $|\rho|$, is defined by:

$$|\rho| \ = \ \begin{cases} 0 & \text{if } \rho = \epsilon \\ 1 & \text{if } \rho = K \\ 1 + |\varrho| & \text{if } \rho = K \cdot \varrho \end{cases}$$

For example, $|\mathit{emp} \cdot e1| = 2$ and $|\mathit{dept} \cdot d1 \cdot \mathit{emp} \cdot e1| = 4$.

By a straightforward induction on the lengths of paths, it can be verified that deterministic graphs have the following property.

Lemma 3.1: Let G be a deterministic structure. Then for any path ρ and node $a \in |G|$, there is at most one node b such that $G \models \rho(a, b)$. ■

This lemma shows that in DM, any component of a database can be uniquely identified by a path.

Path constraints of P_c introduced in [11] are defined in terms of path formulas.

Definition 3.1 [11]: A *path constraint* φ of P_c is an expression of either the *forward* form

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

or the *backward* form

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

Here α, β, γ are path formulas. Path α is called the *prefix* of φ, denoted by $pf(\varphi)$. Paths β and γ are denoted by $lt(\varphi)$ and $rt(\varphi)$, respectively. ■

For example, φ_1 and φ_2 given in Section 2 can be described by P_c constraints.

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

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

Definition 3.2 [4]: A *word constraint* is an expression of the form

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

where β and γ are path formulas. ■

In other words, a word constraint is a forward constraint of P_c with its prefix being the empty path ϵ. It has been shown in [11] that many P_c constraints cannot be expressed as word constraints or even by the more general constraints given in [4].

Next, we describe implication and finite implication of P_c constraints in the context of the deterministic data model. We assume the standard notion of model from first-order logic [18]. Let G be a deterministic structure and φ be a P_c constraint. We use $G \models \varphi$ to denote that G *satisfies* φ (i.e., G *is a model of* φ). Let Σ be a finite set of P_c constraints. We use $G \models \Sigma$ to denote that G *satisfies* Σ (i.e., G *is a model of* Σ). That is, for every $\phi \in \Sigma$, $G \models \phi$.

Let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c. We use $\Sigma \models \varphi$ to denote that Σ *implies* φ in the context of DM. That is, for every deterministic structure G, if $G \models \Sigma$, then $G \models \varphi$. Similarly, we use $\Sigma \models_f \varphi$ to denote that Σ *finitely implies* φ. That is, for every finite deterministic structure G, if $G \models \Sigma$, then $G \models \varphi$.

In the context of DM, the *implication problem for* P_c is the problem to determine, given any finite subset $\Sigma \cup \{\varphi\}$ of P_c, whether $\Sigma \models \varphi$. Similarly, the *finite implication problem for* P_c is the problem of determining whether $\Sigma \models_f \varphi$.

In the context of the graph model SM, the structures considered in the implication problems for P_c are σ-structures, which are not necessarily deterministic. It was shown in [11, 13] that in SM, the implication and finite implication problems for P_c are undecidable.

Theorem 3.2 [11]: In the context of SM, the implication problem for P_c is r.e. complete, and the finite implication problem for P_c is co-r.e. complete. ■

In the next section, we shall show that this undecidability result no longer holds in the context of DM.

3.3 Path constraint language P_c^w

Let us generalize the syntax of path expressions by including the union operator $+$ as follows:

$$w \ ::= \ \epsilon \mid K \mid w \cdot w \mid w + w$$

That is, we define path expressions to be regular expressions which do not contain the Kleene closure. Let us refer to such expressions as $*$*-free regular expressions*.

Let p be a $*$-free regular expression and ρ be a path. We use $\rho \in p$ to denote that ρ is in the regular language generated by p.

We also treat a $*$-free regular expression p as a logic formula $p(x, y)$, where x and y are free variables. We say that a deterministic structure G *satisfies* $p(x, y)$, denoted by $G \models p(x, y)$, if there exist path $\rho \in p$ and nodes $a, b \in |G|$ such that $G \models \rho(a, b)$.

The following should be noted about $*$-free regular expressions.

- The regular language generated by a $*$-free regular expression is finite.
- Recall that the wildcard symbol "_" matches any edge label. We can express "_" as a $*$-free regular expression. More specifically, let E, the finite set of binary relation symbols in signature σ, be enumerated as K_1, K_2, ..., K_n. Then "_" can be defined as $*$-free regular expression:

$$K_1 + K_2 + \ldots + K_n.$$

For example, we have seen in Section 2 the following path expressions that can be represented as $*$-free regular expressions:

$$\begin{array}{l} emp \cdot \_ \cdot manager \\ emp \cdot \_ \cdot supervising \cdot \_ \end{array}$$

Using $*$-free regular expressions, we define P_c^w as follows.

Definition 3.3: A constraint ϕ of P_c^w is an expression of either the *forward form*:

$$\forall x\, (p(r, x) \to \forall y\, (q(x, y) \to s(x, y))),$$

or the *backward form:*

$$\forall x\, (p(r, x) \to \forall y\, (q(x, y) \to s(y, x))),$$

where p, q and s are $*$-free regular expressions, denoted by $pf(\phi)$, $lt(\phi)$ and $rt(\phi)$, respectively. ■

For example, path constraints ϕ_1, ϕ_2 and ϕ_3 given in Section 2 are P_c^w constraints, but they are not in P_c.

A deterministic structure G *satisfies* a constraint ϕ of P_c^w, denoted by $G \models \phi$, if the following condition is satisfied:

- when ϕ is a forward constraint: for all $a, b \in |G|$, if there exist paths $\alpha \in p$ and $\beta \in q$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b)$, then there exists a path $\gamma \in s$ such that $G \models \gamma(a, b)$;
- when ϕ is a backward constraint: for all $a, b \in |G|$, if there exist paths $\alpha \in p$ and $\beta \in q$ such that $G \models \alpha(r^G, a) \wedge \beta(a, b)$, then there exists $\gamma \in s$ such that $G \models \gamma(b, a)$.

The implication and finite implication problems for P_c^w are formalized in the same way as for P_c, as described in Section 3.2.

Obviously, P_c is properly contained in P_c^w. Thus the corollary below follows immediately from Theorem 3.2.

Corollary 3.3: In the context of SM, the implication and finite implication problems for P_c^w are undecidable. ■

In the next section, we shall show that this undecidability result also breaks down in the context of DM.

3.4 Path constraint language P_c^*

We next further generalize the syntax of path expressions by including the Kleene closure $*$ as follows:

$$e \ ::= \ \epsilon \mid K \mid e \cdot e \mid e + e \mid e^*$$

That is, we define path expressions to be general regular expressions. Recall that the wildcard symbol can be expressed as a ($*$-free) regular expression. In Section 2, we have seen the following path expressions that can be represented as regular expressions:

$$\begin{array}{l} manager \cdot manager^* \\ emp \cdot \_ \cdot manager^* \end{array}$$

Let p be a regular expression and ρ be a path. As in Section 3.3, we use $\rho \in p$ to denote that ρ is in the regular language generated by p. Similarly, we also treat p as a logic formula $p(x, y)$, and define the notion of $G \models p(x, y)$ for deterministic structure G.

Using regular expressions, we define P_c^* as follows.

Definition 3.4: A constraint ψ of P_c^* is an expression of either the *forward form*:

$$\forall x\, (p(r, x) \to \forall y\, (q(x, y) \to s(x, y))),$$

or the *backward form:*

$$\forall x\, (p(r, x) \to \forall y\, (q(x, y) \to s(y, x))),$$

where p, q and s are regular expressions, denoted by $pf(\psi)$, $lt(\psi)$ and $rt(\psi)$, respectively. ■

For example, ψ_1, ψ_2 and ψ_3 given in Section 2 are P_c^* constraints, but they are in neither P_c nor P_c^w.

As in Section 3.3, for a deterministic structure G and a P_c^* constraint ψ, we can define the notion of $G \models \psi$. Similarly, we can formalize the implication and finite implication problems for P_c^*.

For example, let $\Sigma = \{\psi_1, \psi_2, \phi_1, \phi_2\}$. Then the question whether $\Sigma \models \psi_3$ ($\Sigma \models_f \psi_3$) is an instance of the (finite) implication problem for P_c^*. In Section 2, this implication is used in the proof of the equivalence of the queries Q_2 and Q'_2.

Clearly, P_c^w is a proper subset of P_c^*. Therefore, by Corollary 3.3, we have the following.

Corollary 3.4: In the context of SM, the implication and finite implication problems for P_c^* are undecidable. ■

In the next section, we shall show that this undecidability result still holds in the context of DM.

4 Path constraint implication

In this section, we study the implication problems associated with P_c, P_c^w and P_c^* for the deterministic data model DM. More specifically, we show the following.

Theorem 4.1: In the context of DM, the implication and finite implication problems for P_c are finitely axiomatizable and are decidable in cubic-time. ■

Proposition 4.2: In the context of DM, the implication and finite implication problems for P_c^w are decidable. ■

Theorem 4.3: In the context of DM, the implication and finite implication problems for P_c^* are undecidable. ■

In contrast to Theorem 3.2 and Corollary 3.3, Theorem 4.1 and Proposition 4.2 show that in the context of DM, the implication problems for P_c and P_c^w are decidable. This demonstrates that the determinism condition of DM may simplify reasoning about path constraints. However, Theorem 4.3 shows that this determinism condition does not trivialize the problem of path constraint implication.

4.1 Decidability of P_c

We prove Theorem 4.1 in two steps. We first present a finite axiomatization for P_c constraint implication in the context of DM. That is, we give a finite set of inference rules that is sound and complete for implication and finite implication of P_c constraints. We then show that in the context of DM, there is a cubic-time algorithm for testing implication and finite implication of P_c constraints.

4.1.1 A finite axiomatization

It is desirable to develop a finite set of inference rules for path constraints. Inference rules can be used not only for generating symbolic proofs of implication, but also for studying the essential properties of the constraints. In general, the existence of a finite set of inference rules is a stronger property than the existence of an algorithm for testing implication.

Before we present a finite axiomatization for P_c, we first study basic properties of P_c constraints in the context of DM. While Lemma 4.6 given below holds in the context of both SM and DM, Lemmas 4.4 and 4.5 hold in the context of DM but not in SM. Their proofs require Lemma 3.1. We omit the proofs of these lemmas due to the lack of space, but we encourage the reader to consult [14].

Lemma 4.4: Let φ be a forward constraint of P_c:

$$\varphi = \forall x\, (\alpha(r, x) \to \forall y\, (\beta(x, y) \to \gamma(x, y))),$$

and ψ be a word constraint:

$$\psi = \forall x\, (\alpha \cdot \beta(r, x) \to \alpha \cdot \gamma(r, x)).$$

Then for every deterministic structure G, $G \models \varphi$ iff $G \models \psi$. ■

Word constraints are described in Definition 3.2.

Lemma 4.5: Let φ be a backward constraint of P_c:

$$\varphi = \forall x\, (\alpha(r, x) \to \forall y\, (\beta(x, y) \to \gamma(y, x))),$$

and ψ be a word constraint:

$$\psi = \forall x\, (\alpha(r, x) \to \alpha \cdot \beta \cdot \gamma(r, x)).$$

Then for every deterministic structure G, if it is given that $G \models \exists x\, (\alpha \cdot \beta(r, x))$, then $G \models \varphi$ iff $G \models \psi$. ■

Lemma 4.6: For every finite subset $\Sigma \cup \{\varphi\}$ of P_c,

$$\Sigma \models \varphi \quad \text{iff} \quad \Sigma \cup \{\exists x\, (pf(\varphi) \cdot lt(\varphi)(r, x))\} \models \varphi,$$
$$\Sigma \models_f \varphi \quad \text{iff} \quad \Sigma \cup \{\exists x\, (pf(\varphi) \cdot lt(\varphi)(r, x))\} \models_f \varphi,$$

where $pf(\varphi)$ and $lt(\varphi)$ are described in Definition 3.1. ■

Based on Lemma 4.6, we extend P_c by including constraints of the *existential form* as follows:

$$P_c^e = P_c \cup \{\exists x\, \rho(r, x) \mid \rho \text{ is a path}\}.$$

Constraints of the existential form enable us to assert the existence of paths. As pointed out by [23], this ability is important for specifying Web link characteristics.

For P_c^e, we consider a set of inference rules, $\mathcal{I}_c$, given below. Note that the last four inference rules in $\mathcal{I}_c$ are sound in DM because of Lemmas 4.4 and 4.5.

- Reflexivity:

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

- Transitivity:

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

- Right-congruence:

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

- Empty-path:

$$\frac{}{\exists x\ \epsilon(r,x)}$$

- Prefix:

$$\frac{\exists x\ (\alpha \cdot \beta(r,x))}{\exists x\ \alpha(r,x)}$$

- Entail:

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

- Symmetry:

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

- Forward-to-word:

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

- Word-to-forward:

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

- Backward-to-word:

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

- Word-to-backward:

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

Let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c^e. We use $\Sigma \vdash_{\mathcal{I}_c} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}_c$. That is, there is an $\mathcal{I}_c$-proof of φ from Σ.

The following theorem shows that in the context of DM, $\mathcal{I}_c$ is indeed a finite axiomatization of P_c.

Theorem 4.7: In the context of DM, for every finite subset $\Sigma \cup \{\varphi\}$ of P_c,

$$\begin{array}{lll} \Sigma \models \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi, \\ \Sigma \models_f \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi. \end{array}$$

■

Proof sketch: By Lemma 4.6, we only need to show that $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \models \varphi$ if and only if $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi$.

Soundness of $\mathcal{I}_c$ can be verified by induction on the lengths of $\mathcal{I}_c$-proofs. For the proof of completeness, it suffices to show the following:

Claim: There is a finite deterministic structure G such that $G \models \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r^G,\, x))\}$. In addition, if $G \models \varphi$, then $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x)))\} \vdash_{\mathcal{I}_c} \varphi$.

To see why this claim suffices, suppose it is given that $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x)))\} \models \varphi$. By the claim, $G \models \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x)))\}$. Therefore, we have $G \models \varphi$. In addition, since G is finite, if it is the case where $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \models_f \varphi$, then we also have $G \models \varphi$. Thus again by the claim, we have that $\Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi$. Space limitations do not allow us to include the lengthy definition of G. The interested reader should consult [14]. ■

As an immediate corollary of Theorem 4.7, in the context of DM, the implication and finite implication problems for P_c coincide and are decidable.

In addition, it can be shown that $\mathcal{I}_c$ is also a finite axiomatization of P_c^e, by using a proof similar to that of Theorem 4.7.

Theorem 4.8: In the context of DM, for every finite subset $\Sigma \cup \{\varphi\}$ of P_c^e, if $\varphi \in P_c$, then

$$\begin{array}{lll} \Sigma \models \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi, \\ \Sigma \models_f \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (pf(\varphi) \cdot lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_c} \varphi. \end{array}$$

Otherwise, i.e., when φ is an existential constraints,

$$\begin{array}{lll} \Sigma \models \varphi & \text{iff} & \Sigma \vdash_{\mathcal{I}_c} \varphi, \\ \Sigma \models_f \varphi & \text{iff} & \Sigma \vdash_{\mathcal{I}_c} \varphi. \end{array}$$

■

In the context of SM, [4] has shown that the first three rules of $\mathcal{I}_c$, i.e., Reflexivity, Transitivity and Right-congruence, are sound and complete for word constraint implication. In the context of DM, however, these rules are no longer complete. To illustrate this, let α be a path and consider the following word constraints:

$$\begin{array}{lll} \varphi & = & \forall\, x\, (\epsilon(r,\, x) \to \alpha(r,\, x)) \\ \phi & = & \forall\, x\, (\alpha(r,\, x) \to \epsilon(r,\, x)) \end{array}$$

By Lemma 3.1, it can be verified that $\varphi \models \phi$. However, this implication cannot be derived by using these three rules.

In the context of DM, the first seven rules of $\mathcal{I}_c$ are sound and complete for word constraint implication. More specifically, let $\mathcal{I}_w$ be the set consisting of these seven rules. Then we can show the following by using a proof similar to that of Theorem 4.7.

Theorem 4.9: In the context of DM, for every finite set $\Sigma \cup \{\varphi\}$ of word constraints,

$$\begin{array}{lll} \Sigma \models \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_w} \varphi, \\ \Sigma \models_f \varphi & \text{iff} & \Sigma \cup \{\exists\, x\, (lt(\varphi)(r,\, x))\} \vdash_{\mathcal{I}_w} \varphi. \end{array}$$

■

4.1.2 A cubic-time algorithm

Based on Theorem 4.7, we can show the following:

Proposition 4.10: There exists an algorithm that, given a finite subset Σ of P_c and paths α, β, computes a finite deterministic structure G in time $O(n^3)$, where n is the length of Σ and $\alpha \cdot \beta$. The structure G has the following property: there are nodes $a, b \in |G|$ such that $G \models \alpha(r^G,\, a) \wedge \beta(a,\, b)$, and moreover, for any path γ,

$$\begin{array}{lll} G \models \gamma(a,\, b) & \text{iff} & \Sigma \cup \{\exists\, x\, (\alpha \cdot \beta(r,\, x))\} \vdash_{\mathcal{I}_c} \forall\, x\, (\alpha(r,\, x) \\ & & \quad \to \forall\, y\, (\beta(x,\, y) \to \gamma(x,\, y))), \\ G \models \gamma(b,\, a) & \text{iff} & \Sigma \cup \{\exists\, x\, (\alpha \cdot \beta(r,\, x))\} \vdash_{\mathcal{I}_c} \forall\, x\, (\alpha(r,\, x) \\ & & \quad \to \forall\, y\, (\beta(x,\, y) \to \gamma(y,\, x))). \end{array}$$

■

An algorithm having the properties described in the proposition is given in the Appendix. The algorithm constructs the structure G. Each step of the construction corresponds to an application of some inference rule in $\mathcal{I}_c$. The algorithm has low complexity because, by Lemma 3.1, every constraint in Σ is used at most once by the algorithm. We do not include the proof of this proposition due to the lack of space. The interested reader should see [14] for a detailed proof.

By Theorem 4.7, we can use this algorithm for testing implication and finite implication of P_c constraints in the context of DM.

4.2 Decidability of P_c^w

We next prove Proposition 4.2. To establish the decidability of the implication and finite implication problems for P_c^w, it suffices to give a finite model argument. That is, it suffices to show the following claim.

Claim: Let $\Sigma \cup \{\varphi\}$ be a finite subset of P_c^w, and let $\phi = \bigwedge \Sigma \wedge \neg\varphi$. If there is a deterministic structure G such that $G \models \phi$, then there is a finite deterministic structure H such that $H \models \phi$.

For if the claim holds, then the implication and finite implication problems for P_c^w coincide and are decidable.

To show the claim, assume that there is a deterministic structure G satisfying ϕ. Recall that a constraint ψ of P_c^w is of either the form

- $\forall x\, (pf(\psi)(r,\, x) \rightarrow \forall y\, (lt(\psi)(x,\, y) \rightarrow rt(\psi)(x,\, y)))$ (i.e., the forward form), or the form
- $\forall x\, (pf(\psi)(r,\, x) \rightarrow \forall y\, (lt(\psi)(x,\, y) \rightarrow rt(\psi)(y,\, x)))$ (i.e., the backward form),

where $pf(\psi)$, $lt(\psi)$ and $rt(\psi)$ are $*$-free regular expressions, as described in Definition 3.3. Let

$$\begin{aligned} PEs(\phi) &= \{pf(\psi)\cdot lt(\psi),\ pf(\psi)\cdot rt(\psi) \mid \psi \in \Sigma\cup\{\varphi\},\\ &\qquad \psi \text{ is of the forward form}\}\\ &\cup \{pf(\psi)\cdot lt(\psi)\cdot rt(\psi) \mid \psi \in \Sigma\cup\{\varphi\},\\ &\qquad \psi \text{ is of the backward form}\},\\ Pts(\phi) &= \{\varrho \mid \varrho \text{ is a path},\ p \in PEs(\phi),\ \varrho \in p\},\\ CloPts(\phi) &= \{\rho \mid \varrho \in Pts(\phi),\ \rho \preceq \varrho\}. \end{aligned}$$

Here $\varrho \in p$ means that path ϱ is in the regular language generated by $*$-free regular expression p, and $\rho \preceq \varrho$ stands for that path ρ is a prefix of path ϱ. Let E_ϕ be the set of edge labels appearing in some path in $Pts(\phi)$. Then we define H to be $(|H|,\, r^H,\, E^H)$ such that

- $|H| = \{a \mid a \in |G|,\ \rho \in CloPts(\phi),\ G \models \rho(r^G,\, a)\}$,
- $r^H = r^G$,
- for all $a, b \in |H|$ and $K \in E$, $H \models K(a,\, b)$ iff $K \in E_\phi$ and $G \models K(a,\, b)$.

It is easy to verify that $H \models \phi$ and H is deterministic, since G has these properties. By Lemma 3.1, the size of $|H|$ is at most the cardinality of $CloPts(\phi)$, which is finite because the regular language generated by a $*$-free regular expression is finite. This proves the claim. It should be noted that E_ϕ and $CloPts(\phi)$ are determined by ϕ only.

4.3 Undecidability of P_c^*

Next, we prove Theorem 4.3. We establish the undecidability of the implication and finite implication problems for P_c^* by reduction from the word problem for (finite) monoids. Before we give the proof, we first review the word problem for (finite) monoids.

4.3.1 The word problem for (finite) monoids

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

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

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

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

Theorem 4.11: Both the word problem for monoids and the word problem for finite monoids are undecidable. ■

4.3.2 Reduction from the word problem

We next present an encoding of the word problem for (finite) monoids in terms of the (finite) implication problem for P_c^* in the context of DM.

Let $?_0$ be a finite alphabet and Θ_0 be a finite set of equations over $?_0$. Without loss of generality, assume $?_0 \subseteq E$, where E is the set of binary relation symbols in signature σ described in Section 3. Assume

$$\begin{aligned} ?_0 &= \{K_j \mid j \in [1, m],\ K_i \neq K_j \text{ if } i \neq j\},\\ \Theta_0 &= \{(\alpha_i, \beta_i) \mid \alpha_i, \beta_i \in ?_0^*,\ i \in [1, n]\}. \end{aligned}$$

Note here that each symbol in $?_0$ is a binary relation symbol in E. Therefore, every α in $?_0^*$ can be represented as a path, also denoted by α. We use $\cdot$ to denote the concatenation operator for both paths and strings.

Let e_0 be the regular expression defined by:

$$e_0 = (K_1 + K_2 + \ldots + K_m)^*$$

We encode Θ_0 in terms of a subset Σ of P_c^*, which includes the following: for each $i \in [1, n]$,

$$\begin{gathered} \forall x\, (e_0(r,\, x) \rightarrow \forall y\, (\alpha_i(x,\, y) \rightarrow \beta_i(x,\, y))),\\ \forall x\, (e_0(r,\, x) \rightarrow \forall y\, (\beta_i(x,\, y) \rightarrow \alpha_i(x,\, y))). \end{gathered}$$

Let (α, β) be a test equation, where α and β are arbitrary strings in $?_0^*$. We encode this test equation as

$$\varphi = \forall x\, (e_0(r, x) \to \forall y\, (\alpha(x, y) \to \beta(x, y))).$$

It should be noted that in the encoding above, only forward constraints of P_c^* are used. In addition, for each $\psi \in \Sigma \cup \{\varphi\}$, $lt(\psi)$ and $rt(\psi)$ are simply paths rather than complex regular expressions, where $lt(\psi)$ and $rt(\psi)$ are described in Definition 3.4.

The lemma below shows that the encoding above is indeed a reduction from the word problem for (finite) monoids. From this lemma and Theorem 4.11, Theorem 4.3 follows immediately.

Lemma 4.12: In the context of DM,

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

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

■

Proof sketch: We give a proof sketch of (b). The proof of (a) is similar and simpler. Owing to the space limit, we omit the details of the lengthy proof, but we encourage the interested reader to consult [14].

(*if*) Suppose that $\Theta_0 \not\models_f (\alpha, \beta)$. Then there exist a finite monoid M and a homomorphism $h : ?_0^* \to M$ such that $h(\alpha_i) = h(\beta_i)$ for $i \in [1, n]$, but $h(\alpha) \neq h(\beta)$. We show that there exists a finite deterministic structure G, such that $G \models \Sigma$ and $G \not\models \varphi$.

To do this, we define an equivalence relation on $?_0^*$:

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

For every string $\rho \in ?_0^*$, let $\widehat{\rho}$ be the equivalence class of ρ with respect to $\approx$, and let $o(\widehat{\rho})$ be a distinct node. Then we define a structure $G = (|G|, r^G, E^G)$, such that $|G| = \{o(\widehat{\rho}) \mid \rho \in ?_0^*\}$ and the root $r^G = o(\widehat{\epsilon})$. The binary relations are populated in G such that for each $K \in E$ and $o(\widehat{\rho}), o(\widehat{\varrho}) \in |G|$, $G \models K(o(\widehat{\rho}), o(\widehat{\varrho}))$ iff $\rho \cdot K \in \widehat{\varrho}$. It can be verified that G is indeed a finite deterministic structure. In addition, $G \models \Sigma$ and $G \not\models \varphi$. A property of e_0 used in the proof is that $\epsilon \in e_0$. That is, the empty path ϵ is in the language generated by the regular expression e_0.

(*only if*) Suppose that there is a finite deterministic structure G such that $G \models \Sigma$ and $G \models \neg\varphi$. Then we define a finite monoid $(M, \circ, id)$ and a homomorphism $h : ?_0^* \to M$ such that for every $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$.

To do this, we define another equivalence relation $\sim$ on $?_0^*$ as follows:

$$\begin{aligned}\rho \sim \varrho \text{ iff } G \models {} & \forall x (e_0(r, x) \to \forall y\, (\rho(x, y) \to \varrho(x, y))) \wedge \\ & \forall x\, (e_0(r, x) \to \forall y\, (\varrho(x, y) \to \rho(x, y))).\end{aligned}$$

For every $\rho \in ?_0^*$, let $[\rho]$ denote the equivalence class of ρ with respect to $\sim$. Then we define $M = \{[\rho] \mid \rho \in ?_0^*\}$, operator $\circ$ by $[\rho] \circ [\varrho] = [\rho \cdot \varrho]$, identity $id = [\epsilon]$, and $h : ?_0^* \to M$ by $h : \rho \mapsto [\rho]$. It can be verified that $(M, \circ, [\epsilon])$ is a finite monoid, h is a homomorphism, and in addition, for every $i \in [1, n]$, $h(\alpha_i) = h(\beta_i)$, but $h(\alpha) \neq h(\beta)$. In the proof, we use the following property of e_0: for any $\rho \in ?_0^*$, $e_0 \cdot \rho \subseteq e_0$. That is, the language generated by the regular expression $e_0 \cdot \rho$ is contained in the language generated by e_0. ■

5 Conclusion

We have investigated path constraints for the deterministic data model DM. Three path constraint languages have been considered: P_c, P_c^w and P_c^*. While P_c was studied for the graph model SM for semistructured data [11, 13], P_c^w and P_c^* have not appeared in any literature. We have demonstrated how constraints of these languages might be used for, among other things, query optimization. We have also studied implication problems associated with these constraint languages in the context of DM. More specifically, we have shown that in contrast to the undecidability result of [11, 13] established for SM, the implication and finite implication problems for P_c and P_c^w are decidable in the context of DM. In particular, the implication problems associated with P_c are decidable in cubic-time and are finitely axiomatizable. These results show that the determinism condition of DM may simplify the analysis of path constraint implication. However, we have also shown that the implication and finite implication problems for P_c^* remain undecidable in the context of DM. This shows that the determinism condition does not trivialize the problem of path constraint implication.

A number of important questions are open.

First, a more general deterministic data model for semistructured data, DDM, was proposed in [10], in which edge labels may also have structure. A type system for DDM is currently under development, in which certain path constraints are embedded. A natural question here is: do the decidability and undecidability results established here hold in DDM? This question becomes more intriguing when types are considered. As shown in [12], adding a type to the data in some cases simplifies reasoning about path constraints, and in other cases makes it harder.

Second, to define a richer data model for semistructured data, one may want to replace the set of edge labels with a set of logic formulas, which possesses a decidable satisfiability problem. A question here is: in this new setting, do the decidability results of this paper still hold?

Third, can path constraints help in reasoning about the equivalence of data representations?

Finally, how should path constraints be used in reasoning about the containment and equivalence of path queries? What kind of automatic tools should be developed to achieve this?

Acknowledgements. We thank Victor Vianu for comments and discussions.

References

[1] S. Abiteboul. “Querying semi-structured data”. In *Proc. 6th Int'l. Conf. on Database Theory (ICDT'97)*, 1997.

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

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Weiner. “The lorel query language for semistructured data”. *J. Digital Libraries*, 1(1), 1997.

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

[5] C. Beeri and M. Y. Vardi. “Formal systems for tuple and equality generating dependencies”. *SIAM J. Comput.*, 13(1): 76 - 98, 1984.

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

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

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

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. “A query language and optimization techniques for unstructured data”. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, 1996.

[10] P. Buneman, A. Deutsch, and W. Tan. “A deterministic model for semi-structured data”. In *Proc. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.

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

[12] P. Buneman, W. Fan, and S. Weinstein. “Interaction between path and type constraints”. In *Proc. 18th ACM Symp. on Principles of Database Systems (PODS'99)*, 1999.

[13] P. Buneman, W. Fan, and S. Weinstein. “Path constraints in semistructured databases”. To appear in *J. Comput. System Sci. (JCSS)*.

[14] P. Buneman, W. Fan, and S. Weinstein. “Path constraints on deterministic graphs”. Technical report MS-CIS-98-33, Department of Computer and Information Science, University of Pennsylvania, 1998. Available as `ftp://ftp.cis.upenn.edu/pub/papers/db-research/tr9833.ps.gz`.

[15] D. Calvanese, G. De Giacomo, and M. Lenzerini. “What can knowledge representation do for semi-structured data?” In *Proc. 15th National Conf. on Artificial Intelligence (AAAI/IAAI'98)*, 1998.

[16] D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. “Reasoning in expressive description logics”. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 1999.

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

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

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

[20] D. Harel. “Dynamic logic”. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic. II: Extensions of Classical Logic*. D. Reidel Publ. Co., 1984.

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

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

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

[24] A. O. Mendelzon, G. A. Mihaila, and T. Milo. “Querying the World Wide Web”. *J. Digital Libraries*, 1(1), 1997.

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

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

[27] J. Thierry-Mieg and R. Durbin. “Syntactic definitions for the ACEDB data base manager”. Technical Report MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge, CB2 2QH, UK, 1992.

[28] M. Y. Vardi and P. Wolper. “Automata-theoretic techniques for modal logic of programs”. *J. Comput. System Sci. (JCSS)*, 32(2), 1986.

Appendix

The algorithm for testing implication and finite implication of P_c constraints is shown in Table 1. The procedure *merge* used in the algorithm is given in Table 2.

```
Algorithm

Input: a finite subset $\Sigma$ of $P_c$ and paths $\alpha$, $\beta$
Output: the structure $G$ described in Proposition 4.10

1. $E_\phi$ := the set of edge labels appearing in either $\alpha \cdot \beta$ or some path in constraints of $\Sigma$;
2. $Rules := \Sigma$;
3. $G := (|G|, r^G, E_\phi^G)$, where
   • $|G| = \{o(\rho) \mid \rho \preceq \alpha \cdot \beta,\ o(\rho)$ is a distinct node$\}$,
   • $r^G = o(\epsilon)$,
   • $E_\phi^G$ is populated such that $G \models K(o(\rho), o(\varrho))$ iff $\varrho = \rho \cdot K$;
4. repeat until no further change:
     (1) if $\forall x\, (\rho(r, x) \to \forall y\, (\varrho(x, y) \to \zeta(x, y))) \in \Sigma$ and there are $o_\rho, o_{\rho\cdot\varrho} \in |G|$ such that
           $G \models \rho(r^G, o_\rho) \wedge \varrho(o_\rho, o_{\rho\cdot\varrho})$ then
           (i) $Rules := Rules \setminus \{\forall x\, (\rho(r, x) \to \forall y\, (\varrho(x, y) \to \zeta(x, y)))\}$;
           (ii) for each $\xi \cdot K \preceq \zeta$ do
                  if there is no $o \in |G|$ such that $G \models \xi \cdot K(o_\rho, o)$ then
                      (a) add to $|G|$ a distinct node $o_{\rho\cdot\xi\cdot K}$;
                      (b) add to $E_\phi^G$ an edge labeled $K$ from $o_{\rho\cdot\xi}$ to $o_{\rho\cdot\xi\cdot K}$,
                            where $o_{\rho\cdot\xi} \in |G|$ such that $G \models \xi(o_\rho, o_{\rho\cdot\xi})$;
           (iii) $merge(o_{\rho\cdot\varrho}, o_{\rho\cdot\zeta})$;
     (2) if $\forall x\, (\rho(r, x) \to \forall y\, (\varrho(x, y) \to \zeta(y, x))) \in \Sigma$ and there are $o_\rho, o_{\rho\cdot\varrho} \in |G|$ such that
           $G \models \rho(r^G, o_\rho) \wedge \varrho(o_\rho, o_{\rho\cdot\varrho})$ then
           (i) $Rules := Rules \setminus \{\forall x\, (\rho(r, x) \to \forall y\, (\varrho(x, y) \to \zeta(y, x)))\}$;
           (ii) for each $\xi \cdot K \preceq \zeta$ do
                  if there is no $o \in |G|$ such that $G \models \xi \cdot K(o_{\rho\cdot\varrho}, o)$ then
                      (a) add to $|G|$ a distinct node $o_{\rho\cdot\varrho\cdot\xi\cdot K}$;
                      (b) add to $E_\phi^G$ an edge labeled $K$ from $o_{\rho\cdot\varrho\cdot\xi}$ to $o_{\rho\cdot\varrho\cdot\xi\cdot K}$,
                            where $o_{\rho\cdot\varrho\cdot\xi} \in |G|$ such that $G \models \xi(o_{\rho\cdot\varrho}, o_{\rho\cdot\varrho\cdot\xi})$;
           (iii) $merge(o_\rho, o_{\rho\cdot\varrho\cdot\zeta})$;
5. output $G$.
```

Table 1: An algorithm for testing P_c constraint implication in DM

```
procedure merge(a, b)

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

Table 2: Procedure *merge*