Reasoning About Equations and Functional Dependencies on Complex Objects[*]

M. F. van Bommel and G. E. Weddell

Department of Computer Science

University of Waterloo

Waterloo, Canada N2L 3G1

Abstract

Virtually all semantic or object-oriented data models assume objects have an identity separate from any of their parts, and allow users to define complex object types in which part values may be any other objects. This often results in a choice of query language in which a user can express navigating from one object to another by following a property value path. In this paper, we consider a constraint language in which one may express equations and functional dependencies over complex object types. The language is novel in the sense that component attributes of individual constraints may correspond to property paths. The kind of equations we consider are also important since they are a natural abstraction of the class of *conjunctive queries* for query languages which support property value navigation. In our introductory comments, we give an example of such a query, and outline two applications of the constraint theory to problems relating to a choice of access plan for the query.

We present a sound and complete axiomatization of the constraint language for the case in which interpretations are permitted to be infinite, where interpretations themselves correspond to a form of directed labeled graph. Although the implication problem for our form of equational constraint alone over arbitrary schema is undecidable, we present decision procedures for the implication problem for both kinds of constraints when the problem schema satisfies a stratification condition, and when all input functional dependencies are *keys*.

[*]This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, and by Bell-Northern Research Ltd.

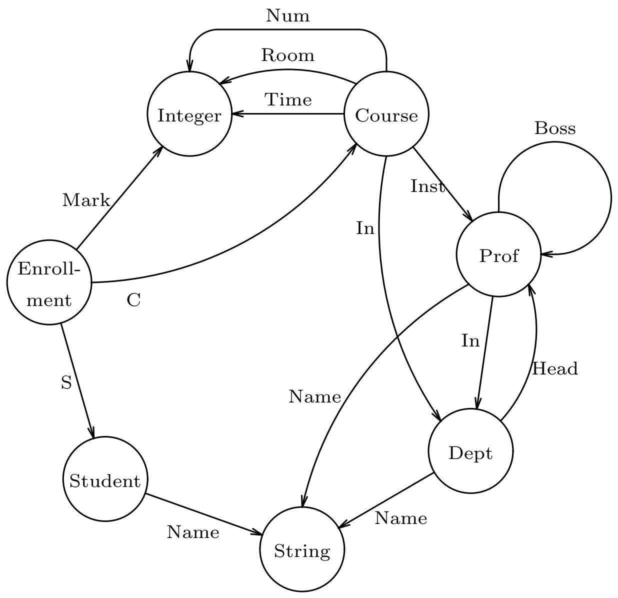


Figure 1: A UNIVERSITY schema graph.

1. Introduction

We consider the problem of reasoning about two kinds of constraints for data models which support the definition and manipulation of complex objects [1, 2, 11, 16, 26, 31, 39]. The two kinds of constraints resemble equations and functional dependencies, and are referred to as *path equations* (PEs) and *path functional dependencies* (PFDs). The work "path" is motivated by their form; component attributes can correspond to descriptions of property value paths in a database. We refer to these descriptions as *path functions*.

In contrast to the relational model, complex object models are pointer-based instead of value-based. Essentially, this means that objects exist independently of their property values and also that property values may in turn point to any other objects. To focus on the essential ideas, we define a simple pointer-based model in the next section. An example *schema graph* characterizing information about student course enrollment at a hypothetical university in terms of this data model is depicted in Figure 1. Each *class* in the schema graph is represented by a labeled vertex, and each *class property* by a labeled arc. In our model, the arcs represent functions that are total on their "from" class, and single-valued on their "to" class.

An example of a path function over the UNIVERSITY schema is the "department of the instructor" function from course objects to department objects, denoted "`Inst.In`". A number of examples of PE and PFD constraints over the UNIVERSITY schema are listed in Table 1. Informally, the two PE constraints are satisfied by a database only if professors teach courses offered by their own departments, and only if a department head is responsible for all professors in the department, including herself.

Table 1: PE and PFD constraints over the UNIVERSITY schema.

path equations	*path functional dependencies*
`Course( Inst.In = In )`	`Dept( Name` $\rightarrow$ `Id )`
`Prof( Boss = In.Head )`	`Dept( Head` $\rightarrow$ `Id )`
	`Course( Num In` $\rightarrow$ `Id )`
	`Course( Room Time` $\rightarrow$ `Id )`
	`Enrollment( S C` $\rightarrow$ `Id )`
	`Enrollment( S C.Time` $\rightarrow$ `C )`

The first five PFD constraints mention an *identity* path function, "`Id`", on their right-hand-side. The identity path function is our means of referring to property value paths of zero length—of referencing object identity directly. Thus, they express the following respective *key* constraints:

- no two departments have the same name (similar constraints might be given for students and professors),
- no professor is the head of more than one department,
- no two courses in the same department have the same number,
- no two courses can be given in the same room at the same time, and
- a student can enroll at most once in a given course.

The only non-key PFD constraint, occurring as the last entry in the table, is justified by virtue of a physical limitation; it is satisfied by a database if no student is enrolled in two different courses at the same time.

PE constraints are also a natural abstraction for the join and selection conditions occurring in conjunctive queries for object-oriented query languages. To illustrate, consider the following SQL-like query on the UNIVERSITY schema graph (except for the "distinct" keyword, the query is an instance of the RELOOP query language supported by the O_2 database system [15]):

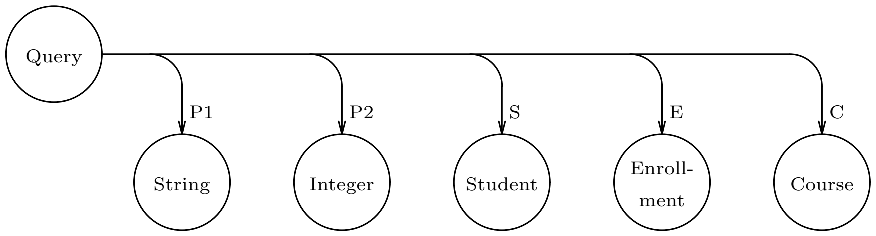


Figure 2: A QUERY CLASS OVER THE UNIVERSITY SCHEMA.

```
select   distinct S
from     S in Student, E in Enrollment, C in Course
where    S = E.S and
         C.Time = E.C.Time and
         C.Inst.In.Name = P1 and
         C.Num = P2.
```

The query is parameterized by a string variable `P1` and integer variable `P2`. Our intention is that, for a particular binding of `P1` and `P2`, the query returns a sequence of all distinct student objects `S` for which

> `S` *is enrolled in some course taught at the same time as some other course numbered* `P2` *with an instructor in the department named* `P1`.

The query includes two join conditions and two selection conditions mentioning a number of path descriptions, which in turn represent a number of functional joins [39].

Part of the intention of the query can be captured by augmenting the UNIVERSITY schema graph with the additional `Query` class illustrated in Figure 2. If we think of an object in this new class as representing a solution to the query, then the join and selection conditions can be abstracted as the collection of four PE constraints on `Query` listed in Table 2. This is useful for a number of reasons relating to query optimization. To illustrate two of these reasons, consider an access plan for the query based on the following nested-loops strategy.

Table 2: PE constraints on class `Query`.

`Query( S = E.S ),`
`Query( C.Time = E.C.Time ),`
`Query( C.Inst.In.Name = P1 ),` and
`Query( C.Num = P2 ).`

```
for each Course object C satisfying the last two conditions do
    for each Enrollment object E satisfying the second condition do
        for each Student object S satisfying the first condition do
            add S to the output if not already in the output.
        end
    end
end
```

The first case concerns the detection of search conditions for complex object indices. In particular, assume an index of all `Course` objects exists which is sorted according to the value of the path function "`In.Name`."[1] One of our procedures given in Section 4 can determine that the PE constraint

$$\texttt{Query(C.In.Name = P1)} \tag{1.1}$$

is a *logical consequence* of the PE constraints on class `Query` listed in Table 2 together with the PE and PFD constraints listed in Table 1. This means that a scan of the index with search argument `P1` cannot fail to locate `Course` objects satisfying the two selection conditions of the query, and therefore that this scan qualifies as one way of implementing the outer loop.

The second case relates to the "if not already in the output" part of the innermost statement. Since such a projection operation is expensive, there is considerable incentive for a query optimizer to be able to determine that the check is unnecessary. Another of our procedures can determine that the PFD constraint

$$\texttt{Query(P1 P2 S} \rightarrow \texttt{E C)} \tag{1.2}$$

is also a logical consequence of the same collection of PE and PFD constraints. Since `P1` and `P2` are constant

[1]Object indexing in this manner has been considered by a number of authors [8, 9, 23, 36].

parameters to the query, the innermost loop will never "visit" a given student object S more than once. This implies that the "if not already in the output" check may be safely eliminated.

Another early application of functional dependency theory in query optimization involves determining *minimal covers* of selection and join conditions [4]. Several authors have also suggested how they may be used to aid in automatically inserting cut operations in access plans based on nested iteration [18, 24, 25, 38].

PFD constraints for the above data model were first introduced and studied in [38], and for a more general model in which a user can define classes that have any number of superclasses in [37]. The problem of reasoning about equations has been studied extensively in the context of *equational logic programming.*[2] The implication problem for PE constraints alone is undecidable for arbitrary schema, and can be efficiently decided by simple variations of the congruence closure algorithms in [19, 27] if a schema satisfies a stratification condition which we introduce in Section 4. A special case of this condition is the set of so-called *acyclic* schema.

The applications of our theory sketched above may be viewed as *semantic query optimization* (SQO) in the context of object-oriented databases. The subject of SQO, as it relates to alternative relational systems such as Datalog [12, 33], may be found in [13, 14, 30]. Also, [29] considers specific problems of reasoning about disequations and inequalities for the relational model.

The remainder of the paper is organized as follows. A formal definition of the above data model, of path functions and of PE and PFD constraints is given in the next section. Following a general trend [5, 6, 17], the semantics of our data model is based on the notion of a database as a directed labeled graph: individual objects and property values correspond to vertices and arcs respectively. We present a sound and complete axiomatization with respect to this graph-based model theory for both forms of constraints in Section 3. Although the general problem is undecidable, Section 4 presents decision procedures for the implication problem for PE and PFD constraints when problem schema satisfy the above-mentioned stratification condition, and when all given functional dependencies are keys. Our summary comments follow in Section 5.

2. Definitions and basic concepts

To begin, we first present the syntax of our data model, commonly referred to as the *data definition language* (DDL). An instance of the DDL defines a space of possible databases, which in our case corresponds to labeled directed graphs.

[2]See [20] for a survey of the topic.

Table 3: The UNIVERSITY schema.

```
class Student { Name:String }

class Prof { Name:String; In:Dept; Boss:Prof }

class Dept { Name:String; Head:Prof }

class Course { Num:Integer; In:Dept; Room:Integer; Time:Integer; Inst:Prof }

class Enrollment { S: Student; C: Course; Mark:  Integer }

class Integer { }

class String { }
```

Definition 1: (*syntax—the* DDL) A *class schema* S consists of a finite set of complex object types of the form

$$\texttt{class}\ C\{P_1 : C_1; \ldots ; P_n : C_n\}$$

in which C is a class name, and the P_i are its properties, written $Props(C)$. Each property P_i is unique in a given class scheme, and its range, written $Ran(C, P_i)$, is the name C_i of another (not necessarily distinct) class scheme. The set of names of classes in S is denoted $Classes(S)$, and the domain of a property P, written $Dom(P)$, is defined as $\{C \in Classes(S) \mid P \in Props(C)\}$. □

The declarations for the UNIVERSITY schema outlined informally in Figure 1 are formally defined in Table 3. The schema illustrates that our model allows property names to be overloaded: a given property name may occur in any number of class declarations. Observe that the range defined for properties such as `S` or `C` are non-built-in classes. Also, in view of properties such as `In` and `Head`, it will be possible for a UNIVERSITY database to contain property value cycles.

Definition 2: (*semantics—a database*) A database for class schema S is a possibly infinite directed graph $G(V, A)$ with vertex and edge labels corresponding to class and property names respectively. G must also satisfy the following constraints, where the class name label of a vertex v is denoted $Cl(v)$.

1. (*property value integrity*) If $u \xrightarrow{P} v \in A$, then $Cl(u) \in Dom(P)$ and $Cl(v) = Ran(Cl(u), P)$.

2. (*property functionality*) If $u \xrightarrow{P} v, u \xrightarrow{P} w \in A$, then $v = w$.

3. (*property value completeness*) If $u \in V$, then there exists $u \xrightarrow{P} v \in A$ for all $P \in Props(Cl(u))$. □

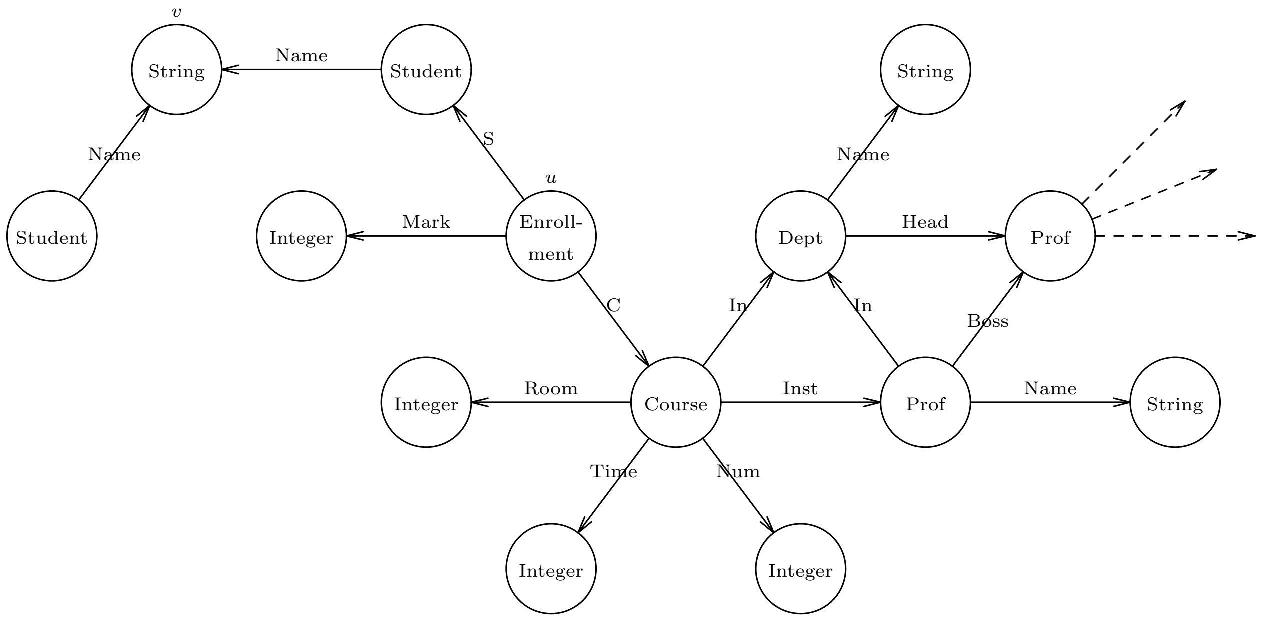


Figure 3: PART OF A DATABASE FOR THE UNIVERSITY SCHEMA.

The directed graph of Figure 3 depicts a portion of one possible database for the UNIVERSITY schema. Note that different `Integer` vertices represent different integers, although the particular integers or strings involved are never important to our presentation. Another possible database, referred to as a *schema graph* in our introductory comments, is depicted in Figure 1. In this case, a single object exists for each class C in $Classes$(UNIVERSITY).

Recall that we referred to component attributes of PE and PFD constraints as *path functions*. A general definition of path functions and the sense in which they *describe* property value paths in a database are given by the following respective definitions.

Definition 3: A *path function pf* over a class schema S is either: 1) a finite sequence of property names occurring in S which are separated by dots, or 2) the keyword `Id`, which we assume does not correspond to the name of any property in S. (Remember that the *identity* path function `Id` is a means of referring to property value paths of zero length, of referring to object identity directly.) The *composition* and *length* operators over path functions are defined as follows.

$$pf_1 \circ pf_2 \overset{\text{def}}{=} \begin{cases} pf_1 & \text{if } pf_2 \text{ is } \texttt{Id}, \\ pf_2 & \text{if } pf_1 \text{ is } \texttt{Id}, \\ pf_1.pf_2 & \text{otherwise.} \end{cases}$$

$$len(pf) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } pf \text{ is } \texttt{Id}, \\ 1 + len(pf_1) & \text{otherwise, where } pf \text{ has the form } pf_1 \circ P, \\ & \text{and } P \text{ is a property.} \end{cases}$$

Let X be the set of path functions $\{pf_1, \ldots, pf_n\}$. We write $pf \circ X$ to denote $\{pf \circ pf_1, \ldots, pf \circ pf_n\}$. □

Note that the composition operator is clearly associative; that is, $pf_1 \circ (pf_2 \circ pf_3) = (pf_1 \circ pf_2) \circ pf_3$. For example, with the UNIVERSITY schema, `S` $\circ$ `Name` is the path function `S.Name`, and both `Id` $\circ$ `C` and `C` $\circ$ `Id` denote the path function `C` (which is also a property). The expression `Id` $\circ$ `C` $\circ$ `Room` denotes either (`Id` $\circ$ `C`)$\circ$ `Room` or `Id` $\circ$(`C` $\circ$ `Room`), and in both cases is the path function `C.Room`. The following identity on *len* is also a straightforward consequence of our definitions.

$$len(pf_1 \circ pf_2) = len(pf_1) + len(pf_2)$$

Definition 4: A path $v_1 \rightarrow \ldots \rightarrow v_{n-1} \xrightarrow{P} v_n$ in a database $G(V, A)$ for class schema S is *described* by a path function pf if and only if either: 1) $pf = \texttt{Id}$ and $n = 1$ (the path consists of a single vertex), or 2) $v_1 \rightarrow \ldots \rightarrow v_{n-1}$ is described by pf_1, for some pf_1 such that $pf = pf_1 \circ P$ (note that $pf_1 = \texttt{Id}$ if $n = 2$; that is, if the original path consists of a single arc). □

For example, `S.Name` is a path function which describes a path from vertex u to vertex v in Figure 3. Now consider that `Name.S` is also a path function according to our definitions. But in this case, no path can ever exist in any database for the UNIVERSITY schema which is described by `Name.S` since none of the range classes of property `Name` includes property `S`. The notion of path function so far presented is therefore too general in an important sense.

In [38], it is proven that a subset of path functions for a given schema S, denoted $PF(S)$ below, satisfies a completeness property for databases over S: any path in any database over S can be described by a path function in $PF(S)$, and any path function in $PF(S)$ describes a path in some database over S. The same reference also proves an important sense in which the composition operator remains closed over $PF(S)$. Both these facts are reproduced as Lemma 1 and Lemma 2 below.

Definition 5: The set of *well-formed path functions* $PF(S)$ over class schema S is the smallest set of path functions over S satisfying the following two conditions. (Note that this extends our use of the notation *Dom* and *Ran* to apply to well-formed path functions as well as properties.)

1. `Id` $\in PF(S)$, where

(a) $\mathit{Dom}(\mathtt{Id}) \stackrel{\text{def}}{=} \mathit{Classes}(S)$, and

(b) $\mathit{Ran}(C, \mathtt{Id}) \stackrel{\text{def}}{=} C$, for all $C \in \mathit{Classes}(S)$.

2. If $\mathit{pf} \in \mathit{PF}(S)$, $C \in \mathit{Dom}(\mathit{pf})$ and $P \in \mathit{Props}(\mathit{Ran}(C, \mathit{pf}))$, then $\mathit{pf} \circ P \in \mathit{PF}(S)$, where

(a) $\mathit{Dom}(\mathit{pf} \circ P) \stackrel{\text{def}}{=} \{C_1 \in \mathit{Dom}(\mathit{pf}) \mid P \in \mathit{Props}(\mathit{Ran}(C_1, \mathit{pf}))\}$, and

(b) $\mathit{Ran}(C_1, \mathit{pf} \circ P) \stackrel{\text{def}}{=} \mathit{Ran}(\mathit{Ran}(C_1, \mathit{pf}), P)$, for all $C_1 \in \mathit{Dom}(\mathit{pf} \circ P)$.

Capital letters X, Y and Z are used to denote finite subsets of $\mathit{PF}(S)$ for some class schema S, and XY, for example, denotes the union of path functions mentioned in X and Y. By a slight abuse of notation, we write $\mathit{PathFuncs}(C)$ to denote all path functions $\mathit{pf} \in \mathit{PF}(S)$ where $C \in \mathit{Dom}(\mathit{pf})$, for $C \in \mathit{Classes}(S)$. A class schema S is *cyclic* if and only if there exists $\mathit{pf} \in \mathit{PF}(S) - \{\mathtt{Id}\}$ and $C \in \mathit{Dom}(\mathit{pf})$ where $C = \mathit{Ran}(C, \mathit{pf})$. (A simple consequence is that S is cyclic if and only if $\mathit{PF}(S)$ is infinite.) $\Box$

Note that the subset of well-formed path functions for cyclic class schema, however, continues to be infinite. For example, the UNIVERSITY schema has a well-formed "boss" function `Boss`, a "boss of the boss" function `Boss.Boss`, and so on. Other UNIVERSITY path functions include

$$\mathtt{S},\ \mathtt{S.Name},\ \mathtt{C},\ \mathtt{C.Room},\ \mathtt{C.Time},\ \mathtt{C.Inst},\ \mathtt{C.Inst.In} \text{ and } \mathtt{C.Inst.In.Head}.$$

Each of these latter path functions is in $\mathit{PathFuncs}(\mathtt{Enrollment})$. Also, for example,

$$\mathit{Dom}(\mathtt{Name}) = \{\mathtt{Prof}, \mathtt{Dept}, \mathtt{Student}\}$$

and

$$\mathit{Ran}(\mathtt{Enrollment}, \mathtt{C.Inst}) = \mathtt{Prof}.$$

Lemma 1: (*expressiveness of well-formed path functions—from* [38]) Let $G(V, A)$ be a database for a given class schema S. If a path $v_1 \to \ldots \to v_n$ exists in G, then there exists a unique $\mathit{pf} \in \mathit{PathFuncs}(\mathit{Cl}(v_1))$ describing $v_1 \to \ldots \to v_n$. Also, for every $u \in V$ and $\mathit{pf} \in \mathit{PathFuncs}(\mathit{Cl}(u))$, there exists a path in G described by pf. $\Box$

As we have outlined, Lemma 1 asserts that no two distinct paths with common end vertices can be described by the *same* path function, which justifies our choice of the phrase "path function" as opposed to, say, "path description". Thus, vertex v in Figure 3 is the *unique vertex reachable from vertex* u *by a path described by* `S.Name`. By a slight abuse of notation, we write $u.\mathtt{S.Name}$ to denote v, and in general $u.\mathit{pf}$ to denote the unique vertex w reachable from u by a path described by pf, whenever $\mathit{pf} \in \mathit{PathFuncs}(\mathit{Cl}(u))$.

Lemma 2: (*closure of composition—also from* [38]) Assume $C \in \mathit{Classes}(S)$, for some class schema S. Then

$$pf_1 \in \mathit{PathFuncs}(C),\ pf_2 \in \mathit{PF}(S) \text{ and } \mathit{Ran}(C, pf_1) \in \mathit{Dom}(pf_2)$$

if and only if

$$pf_1 \circ pf_2 \in \mathit{PathFuncs}(C). \qquad \Box$$

The remaining definitions in this section present the syntax of PE and PFD constraints, and define satisfaction and logical consequence as they relate to the above graph-based notion of a database.

Definition 6: The syntax of a *path equation* (PE) constraint over a class schema S is given by

$$C(pf_1 = pf_2).$$

The constraint is *well-formed* if: 1) $pf_1, pf_2 \in \mathit{PathFuncs}(C)$, and 2) $\mathit{Ran}(C, pf_1) = \mathit{Ran}(C, pf_2)$. A well-formed PE constraint $C(pf_1 = pf_2)$ is *satisfied* by a database $G(V, A)$ of S if and only if for every $v \in V$ where $Cl(v) = C$, $v.pf_1 = v.pf_2$.

The syntax of a *path functional dependency* (PFD) constraint over a class schema S is given by

$$C(pf_1 \cdots pf_m \to pf_{m+1} \cdots pf_n).$$

In this case, the constraint is *well-formed* if: 1) $1 \leq m < n$ (we disallow empty left or right-hand sides), and 2) $pf_i \in \mathit{PathFuncs}(C)$, for $1 \leq i \leq n$. A *key path functional dependency* (key PFD) is any path functional dependency with the single path function `Id` occurring on the right-hand side (after the arrow). A well-formed PFD constraint $C(pf_1 \cdots pf_m \to pf_{m+1} \cdots pf_n)$ is *satisfied* by a database $G(V, A)$ of S if and only if for any pair of vertices $u, v \in V$ where $Cl(u) = Cl(v) = C$, $u.pf_i = v.pf_i$, $1 \leq i \leq m$, implies $u.pf_j = v.pf_j$, $m < j \leq n$. $\Box$

Note that none of the UNIVERSITY data illustrated in either Figure 1 or Figure 3 exhibits a violation of any of the PE or PFD constraints listed in Table 1. In the case of Figure 1, this remains true of *any* collection of PE and PFD constraints since a single object exists for each class. However, Figure 3 does illustrate a violation of the key PFD constraint

`Student( Name → Id )`

which would require students to have unique names. The violation happens in the top-left corner of the graph.

Definition 7: (*logical consequence*) Let Σ denote a set of PE and PFD constraints over a class schema S, and let σ denote an arbitrary PE or PFD constraint also over S. σ is a *logical consequence* of Σ, written $\Sigma \models_S \sigma$, if and only if any database $G(V, A)$ satisfying all constraints in Σ must also satisfy σ. If S is clear from the context, then we write $\Sigma \models \sigma$. □

3. Axioms for Path Equations and Path Functional Dependencies

In this section, we prove that the six inference axioms listed in Table 4 are a sound and complete axiomatization for PE constraints, and that six additional inferences axioms listed in Table 5 yield a sound and complete axiomatization for PFD constraints. The results in both cases assume the above model theory based on the graph theoretic view of databases.

The initial five entries in Table 4 are essentially a restriction to unary functions over a single variable of Birkhoff's rules of inference for equational logic [10], although there is a slight complication due to the many-sorted nature of databases (a given function may not be defined for all objects in a database). A sound and complete set of inference axioms for PFD constraints has been derived in [38], and for a more general model permitting subclassing in [37]. The initial five entries in Table 5 are a simple variation of those given in the latter reference. Of these, the first three generalize a set of similar inference axioms well-known to be complete for ordinary functional dependencies. The last entry in each of the tables accounts for the interaction between PE and PFD constraints.

Definition 8: Let Σ be a set of constraints over class schema S. We write Σ_S^+, or simply Σ^+ when S is understood from context, to denote Σ together with all PE or PFD constraints derivable from Σ using the inference axioms in Tables 4 and 5. □

Theorem 1: (*well-formedness and soundness*) Let Σ denote a set of well-formed constraints over class schema S. If $\sigma \in \Sigma_S^+$, then: 1) σ is well-formed, and 2) $\Sigma \models \sigma$.

Proof Outline. The first part of the theorem is a direct consequence of Lemma 2. The soundness of each axiom follows in a straightforward manner from Lemma 1 and the labeling constraints satisfied by any database.[3]
□

Thus, by Theorem 1 together with Lemmas 1 and 2, we may safely ignore for the remainder of the paper any consideration of cases in which constraints are not well-formed. To illustrate the use of our inference

[3]A full proof may be found in [34].

Table 4: Axioms for PE constraints.

name	*definition*
PERef (*reflexivity*)	$\dfrac{pf \in PathFuncs(C)}{C(pf = pf)}$
PESym (*symmetry*)	$\dfrac{C(pf_1 = pf_2)}{C(pf_2 = pf_1)}$
PETrans (*transitivity*)	$\dfrac{C(pf_1 = pf_2),\ C(pf_2 = pf_3)}{C(pf_1 = pf_3)}$
PEAttr (*attribution*)	$\dfrac{C(pf_1 = pf_2),\ pf \in PathFuncs(Ran(C, pf_1))}{C(pf_1 \circ pf = pf_2 \circ pf)}$
PESubst (*substitution*)	$\dfrac{Ran(C, pf)(pf_1 = pf_2)}{C(pf \circ pf_1 = pf \circ pf_2)}$
PEIntro (*introduction*)	$\dfrac{\begin{array}{l} Ran(C, pf_r)(pf_1 \cdots pf_m \rightarrow Y), \\ C(pf_r \circ pf_1 = pf_s \circ pf_1),\ \ldots\ ,\ C(pf_r \circ pf_m = pf_s \circ pf_m), \\ Ran(C, pf_r) = Ran(C, pf_s) \end{array}}{C(pf_r \circ pf = pf_s \circ pf), \text{ where } pf \in Y}$

Table 5: Axioms for PFD constraints.

name	*definition*
PFDRef (*reflexivity*)	$\dfrac{\emptyset \subset Y \subseteq X \subseteq PathFuncs(C)}{C(X \rightarrow Y)}$
PFDTrans (*transitivity*)	$\dfrac{C(X \rightarrow Y),\ C(Y \rightarrow Z)}{C(X \rightarrow Z)}$
PFDAug (*augmentation*)	$\dfrac{C(X \rightarrow Y),\ Z \subseteq Pathfuncs(C)}{C(XZ \rightarrow YZ)}$
PFDAttr (*attribution*)	$\dfrac{pf \in PathFuncs(C)}{C(\mathtt{Id} \rightarrow pf)}$
PFDSubst (*substitution*)	$\dfrac{Ran(C, pf)(X \rightarrow Y)}{C(pf \circ X \rightarrow pf \circ Y)}$
PFDIntro (*introduction*)	$\dfrac{C(pf_1 = pf_2)}{C(pf_1 \rightarrow pf_2)}$

axioms, we return to the example UNVERSITY query outlined in our introduction. A derivation of the PE constraint (1.1) and the PFD constraint (1.2) is given in Tables 6 and 7 as the sixth and final entries respectively.

Theorem 2: (*consistency*) There is a database $G(V, A)$ for a given class schema S satisfying any set of constraints Σ over S where, for all $C \in Classes(S)$, there exists $v \in V$ such that $Cl(v) = C$.

Proof Outline. Let $G(V, A)$ be a directed labeled graph consisting of a single vertex $u \in V$ for each $C \in Classes(S)$, where $Cl(u)$ is assigned C, and where $u \xrightarrow{P} v \in A$ if and only if $P \in Props(Cl(u))$, and $Ran(Cl(u), P) = Cl(v)$. Then, since a single object exists for each class, G is a database which cannot fail to satisfy any PE or PFD constraint σ over S. (In our introductory comments, we referred to G as a *schema graph.*) □

Our concern in the remainder of this section is with the issue of completeness. We shall see that the inference axioms in Table 4 are complete for PE constraints, and that the addition of the six axioms in Table 5 yield a complete axiomatization for PFD constraints. To simplify the presentation, we assume the two additional inference axioms listed in Table 8; their derivation from PFDRef, PFDTrans and PFDAug is well-known [32, 22].

Our proof that the six inference axioms in Table 4 are complete for PE constraints is based on the following definition of an oriented Herbrand-like database called a *C-Graph*, where C is the name of a particular class in a given schema.

Definition 9: Let Σ denote a set of constraints over class schema S. A *C-Graph*, where $C \in Classes(S)$, is a directed labeled graph $G(V, A)$ constructed as follows. First, partition $PathFuncs(C)$ into a maximal number of subsets $\{P_1, P_2, \ldots\}$ in which $C(pf_1 = pf_2) \in \Sigma^+$ implies $pf_1, pf_2 \in P_i$, for some $1 \leq i$. The set of vertices and arcs are then determined as follows.

1. Create a vertex $u \in V$ for each partition P_i, and assign $Pf(u)$ (a new kind of vertex label) the set of path functions in P_i. Also assign $Cl(u)$ the class $Ran(C, pf)$, for some arbitrary $pf \in Pf(u)$. The vertex $u \in V$ where $\texttt{Id} \in Pf(u)$ is denoted as R_G, or simply R when G is understood from context.

2. For every $u, v \in V$ such that there exists $pf \in Pf(u)$ and $pf \circ P \in Pf(v)$, add arc $u \xrightarrow{P} v$ to A. □

Important conditions which hold on a *C-Graph* $G(V, A)$ are that it satisfies the constraints of property value integrity, functionality and completeness needed to qualify as a database, and that it contains at least

Table 6: Derivation of (1.1) and (1.2).

1.	`Course( Inst.In = In )`	(from Table 1)
2.	`Course( In = Inst.In )`	(1 and PESym)
3.	`Query( C.In = C.Inst.In )`	(2 and PESubst)
4.	`Query( C.In.Name = C.Inst.In.Name )`	(3 and PEAttr)
5.	`Query( C.Inst.In.Name = P1 )`	(from Table 2)
6.	`Query( C.In.Name = P1 )`	(4, 5 and PETrans)
7.	`Query( P1 = C.In.Name )`	(6 and PESym)
8.	`Query( P1` → `C.In.Name )`	(7 and PFDIntro)
9.	`Dept( Name` → `Id )`	(from Table 1)
10.	`Query( C.In.Name` → `C.In )`	(9 and PFDSubst)
11.	`Query( P1` → `C.In )`	(8, 10 and PFDTrans)
12.	`Query( P1 P2` → `C.In P2 )`	(11 and PFDAug)
13.	`Query( C.Num = P2 )`	(from Table 2)
14.	`Query( P2 = C.Num )`	(13 and PESym)
15.	`Query( P2` → `C.Num )`	(14 and PFDIntro)
16.	`Query( C.In P2` → `C.Num C.In )`	(15 and PFDAug)
17.	`Query( P1 P2` → `C.Num C.In )`	(12, 16 and PFDTrans)
18.	`Course( Num In` → `Id )`	(from Table 1)
19.	`Query( C.Num C.In` → `C )`	(18 and PFDSubst)
20.	`Query( P1 P2` → `C )`	(17, 19 and PFDTrans)
21.	`Query( P1 P2 E` → `E C )`	(20 and PFDAug)
22.	`Course( Id` → `Time )`	(PFDAttr)
23.	`Query( C` → `C.Time )`	(22 and PFDSubst)
24.	`Query( P1 P2` → `C.Time )`	(20, 23 and PFDTrans)

Table 7: Derivation of (1.1) and (1.2) (cont'd).

25.	`Query( C.Time = E.C.Time )`	(from Table 2)
26.	`Query( C.Time` $\rightarrow$ `E.C.Time )`	(25 and PFDIntro)
27.	`Query( P1 P2` $\rightarrow$ `E.C.Time )`	(24, 26 and PFDTrans)
28.	`Query( P1 P2 S` $\rightarrow$ `S E.C.Time )`	(27 and PFDAug)
29.	`Query( S = E.S )`	(from Table 2)
30.	`Query( S` $\rightarrow$ `E.S )`	(29 and PFDIntro)
31.	`Query( E.C S` $\rightarrow$ `E.S E.C )`	(30 and PFDAug)
32.	`Query( S E.C.Time` $\rightarrow$ `E.S E.C.Time )`	(30 and PFDAug)
33.	`Query( P1 P2 S` $\rightarrow$ `E.S E.C.Time )`	(28, 32 and PFDTrans)
34.	`Enrollment( S C.Time` $\rightarrow$ `C )`	(from Table 1)
35.	`Query( E.S E.C.Time` $\rightarrow$ `E.C )`	(34 and PFDSubst)
36.	`Query( P1 P2 S` $\rightarrow$ `E.C )`	(33, 35 and PFDTrans)
37.	`Query( P1 P2 S` $\rightarrow$ `E.C S )`	(36 and PFDAug)
38.	`Query( P1 P2 S` $\rightarrow$ `E.S E.C )`	(37, 31 and PFDTrans)
39.	`Enrollment( S C` $\rightarrow$ `Id )`	(from Table 1)
40.	`Query( E.S E.C` $\rightarrow$ `E )`	(39 and PFDSubst)
41.	`Query( P1 P2 S` $\rightarrow$ `E )`	(38, 40 and PFDTrans)
42.	`Query( P1 P2 S` $\rightarrow$ `P1 P2 E )`	(41 and PFDAug)
43.	`Query( P1 P2 S` $\rightarrow$ `E C )`	(42, 21 and PFDTrans)

Table 8: Additional axioms for PFD constraints.

name	*definition*
PFDAdd (*additivity*)	$\dfrac{C(X \rightarrow Y),\, C(X \rightarrow Z)}{C(X \rightarrow YZ)}$
PFDProj (*projectivity*)	$\dfrac{C(X \rightarrow YZ)}{C(X \rightarrow Y)}$

one C object. This and other properties needed in our proof of completeness for PE constraints are stated in the following lemma. A proof is given in Appendix B.

Lemma 3: Let $G(V, A)$ be the *C-Graph* corresponding to a set of constraints Σ over a class schema S constructed with respect to class $C \in Classes(S)$. Then each of the following properties is true of G.

P1. If $pf_1 \in Pf(u)$ for some $u \in V$, then $pf_2 \in Pf(u)$ if and only if $C(pf_1 = pf_2) \in \Sigma^+$.

P2. For any $u \in V$ and $pf \in Pf(u)$, $Ran(C, pf) = Cl(u)$.

P3. G is a database for S.

P4. For all $u \in V$, $u = R.pf$ if and only if $pf \in Pf(u)$. □

Theorem 3: The six axioms in Table 4 are complete for PE constraints; that is, given constraints $\Sigma \cup \{C(pf_1 = pf_2)\}$ over class schema S, if $\Sigma \models C(pf_1 = pf_2)$ then $C(pf_1 = pf_2)$ can be derived from Σ using these axioms alone.

Proof. By P3, the *C-Graph* $G(V, A)$ is a database for S. The theorem therefore follows if G satisfies all constraints in Σ^+, but not $C(pf_1 = pf_2)$ whenever $C(pf_1 = pf_2) \notin \Sigma^+$.

First, we show G does not satisfy $C(pf_1 = pf_2)$. If it did, then $R.pf_1 = R.pf_2$. But then $pf_1, pf_2 \in Pf(u)$ for some $u \in V$ by P4, and therefore $C(pf_1 = pf_2) \in \Sigma^+$ by P1—a contradiction.

We now show that G must satisfy all constraints in Σ^+.

First consider PE constraints. Assume $C_1(pf_3 = pf_4) \in \Sigma^+$, but that there exists some $v \in V$ where $Cl(v) = C_1$ and where $v.pf_3 \neq v.pf_4$. Without loss of generality, select some $pf \in Pf(v)$. Then $R.pf = v$ by P4, and therefore $R.(pf \circ pf_3) \neq R.(pf \circ pf_4)$. Now, $Ran(C, pf) = C_1$ by P2, and therefore $C(pf \circ pf_3 = pf \circ pf_4) \in \Sigma^+$ by the substitution axiom PESubst. Thus, $pf \circ pf_3$ and $pf \circ pf_4$ occur in the same partition P_i, and there exists some $u \in V$ such that $(pf \circ pf_3), (pf \circ pf_4) \in Pf(u)$. But then $R.(pf \circ pf_3) = u = R.(pf \circ pf_4)$ by P4—a contradiction.

Now consider some PFD $C_1(X \to Y) \in \Sigma^+$ together with $u, v \in V$ such that $Cl(u) = Cl(v) = C_1$, and where $u.pf_i = v.pf_i$ for each $pf_i \in X$. By definition of the construction of G and P2 and P4 above, there exists $pf_3 \in Pf(u)$ and $pf_4 \in Pf(v)$ such that $Ran(C, pf_3) = Ran(C, pf_4) = C_1$, $u = R.pf_3$ and $v = R.pf_4$. Thus $R.(pf_3 \circ pf_i) = R.(pf_4 \circ pf_i)$ for each $pf_i \in X$, and therefore $C(pf_3 \circ pf_i = pf_4 \circ pf_i) \in \Sigma^+$ by P4 again and P1. But then $C(pf_3 \circ pf_j = pf_4 \circ pf_j) \in \Sigma^+$ for each $pf_j \in Y$ by PEIntro. According to our earlier proof that G must satisfy all PE constraints, this then implies $u.pf_j = v.pf_j$ for each $pf_j \in Y$, and therefore that G must satisfy $C_1(X \to Y)$. □

Our proof of completeness for PFD constraints resembles the above; it is also based on creating an oriented Herbrand-like database. The construction of the database for this case starts with two copies of a *C-Graph*.

Definition 10: Let Σ denote a set of constraints over class schema S, and $C(X \to Y)$ for $C \in Classes(S)$ some choice of PFD not in Σ^+. A *Two-C-Graph* is a directed labeled graph $G(V, A)$ constructed from two copies $G_1(V_1, A_1)$ and $G_2(V_2, A_2)$ of the *C-Graph* for class C with respect to $C(X \to Y)$ as follows.

1. Remove any $u \in V_2$ and its incident arcs in A_2 whenever there exists $pf \in Pf(u)$ such that $C(X \to pf) \in \Sigma^+$. Add all vertices in $V_1 \cup V_2$ to V and all arcs in $A_1 \cup A_2$ to A.

2. For each $u \in V_2$ and $P \in Props(Cl(u))$ such that $u \xrightarrow{P} v \notin A$ for all $v \in V$, select an arbitrary $pf \in Pf(u)$ and add arc $u \xrightarrow{P} R_{G_1}.(pf \circ P)$ to A. □

A *Two-C-Graph* has the general form illustrated in Figure 4; that is, all vertices can be partitioned into three sets S_1, S_2 and S_3 such that areas S_1 and S_2 are isomorphic, and such that arcs connecting vertices in different sets must originate in either S_1 or S_2 and terminate in S_3. A key condition satisfied by a *Two-C-Graph* is that the two subgraphs corresponding to areas $S_1 \cup S_3$ and $S_2 \cup S_3$ are both *C-Graphs*, and therefore inherit many of the properties of the latter. These and additional properties of *Two-C-Graphs* needed in our

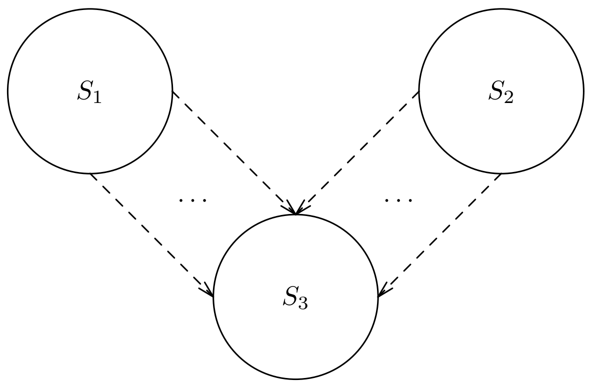


Figure 4: General form of a *Two-C-Graph*.

proof of completeness for PFD constraints are listed in our final lemma of this section. Again, we refer the reader to Appendix B for a proof.

Lemma 4: Let $G(V, A)$ be the *Two-C-Graph* corresponding to a set of constraints Σ over class schema S constructed with respect to class $C \in \mathit{Classes}(S)$, and $C(X \to Y) \notin \Sigma_S^+$. Then properties P1, P2 and P3 of Lemma 3 remain true of G, along with each of the following.

P5. Vertex R_{G_2} is not removed in the first step.

P6. If $u \in V_2$ is removed in the first step, then all $v \in V_2$ reachable from u are also removed.

P7. For all $u, v \in V$ such that a path from u to v exists which is described by pf_2, $(pf_1 \circ pf_2) \in Pf(v)$ for all $pf_1 \in Pf(u)$.

P8. $R_{G_1}.pf = R_{G_2}.pf$ if and only if $C(X \to pf) \in \Sigma^+$. □

Theorem 4: The twelve axioms in Tables 4 and 5 are complete for PFD constraints; that is, given constraints $\Sigma \cup \{C(X \to Y)\}$ over class schemes S, if $\Sigma \models C(X \to Y)$ then $C(X \to Y)$ can be derived from Σ using these axioms alone.

Proof. By P3, the *Two-C-Graph* $G(V, A)$ constructed with respect to $C(X \to Y) \notin \Sigma^+$ is a database for S; the theorem therefore follows if G satisfies all constraints in Σ^+, but not $C(X \to Y)$.

First, we show G does not satisfy $C(X \to Y)$. PFDRef implies $C(X \to pf_1)$ for all $pf_1 \in X$, and therefore $R_{G_1}.pf_1 = R_{G_2}.pf_1$ must hold by P8. Thus, if G does satisfy $C(X \to Y)$, then $R_{G_1}.pf_2 = R_{G_2}.pf_2$ for all $pf_2 \in Y$. But then by P8 again, $C(X \to pf_2) \in \Sigma^+$, and therefore $C(X \to Y) \in \Sigma^+$ follows by PFDAdd—contrary to assumptions.

What remains is to show that G does satisfy all constraints in Σ^+.

With respect to PE constraints, consider $C_1(pf_1 = pf_2) \in \Sigma^+$ and $u \in V$ such that $Cl(u) = C_1$. In light of our proof to Theorem 3, both $u.pf_1$ and $u.pf_2$ must be the same vertex v prior to the first step of the construction. By the converse to P6, if v is not removed during the first step, then both $u.pf_1$ and $u.pf_2$ remain v at the end of the second step. Otherwise, if v is removed during the first step, then u must originate in V_2 (nothing in G_1 is changed). Now, four uses of P4 imply there exists $pf_3 \in Pf(u)$ such that $pf_3 \circ pf_1$ and $pf_3 \circ pf_2$ are in $Pf(v)$. Since $C(pf_3 \circ pf_1 = pf_3 \circ pf_2) \in \Sigma^+$ by P2 and PESubst, P1 implies there remains a single vertex w (originating in V_1) at the end of the second step such that $(pf_3 \circ pf_1), (pf_3 \circ pf_2) \in Pf(w)$ at the end of the second step. Thus by P7: $u.pf_1 = u.pf_2 = w$.

We show finally that G also satisfies all PFD constraints in Σ^+. Assume $C_1(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$ is in Σ^+, but is not satisfied by G. Then there must exist $u, v \in V$ such that:

1. $Cl(u) = Cl(v) = C_1$,

2. $u.pf_i = v.pf_i$ for each $1 \leq i \leq m$, and

3. $u.pf_j \neq v.pf_j$ for some $m < j \leq n$.

Let pf_r and pf_s denote an arbitrary choice of path function in $Pf(u)$ and $Pf(v)$ respectively. The second condition above together with P7 imply $Pf(u.pf_i)$ contains both $pf_r \circ pf_i$ and $pf_s \circ pf_i$, and therefore $C(pf_r \circ pf_i = pf_s \circ pf_i) \in \Sigma^+$ follows by P1, for each $1 \leq i \leq m$. Since $Ran(C, pf_r) = Ran(C, pf_s) = C_1$ by P2, axiom PEIntro then implies $C(pf_r \circ pf_k = pf_s \circ pf_k) \in \Sigma^+$ for each $m < k \leq n$. Thus $R_{G_1}.(pf_r \circ pf_k) = R_{G_1}.(pf_s \circ pf_k)$ for each $m < k \leq n$ follows from our proof above that G satisfies all PE constraints in Σ^+, which also implies $u.pf_j = v.pf_j$ if u and v are both reachable from the same root vertex—contrary to the third condition above. Without loss of generality, we therefore assume $R_{G_1}.pf_r = u$ and $R_{G_2}.pf_s = v$. But then the second condition above is possible if and only if the first step of the construction removes vertex $w = R_{G_2}.(pf_s \circ pf_i)$ from V_2. This happens if and only if there exists $pf \in Pf(w)$ such that $C(X \rightarrow pf) \in \Sigma^+$, which implies

$C(X \to pf_s \circ pf_i) \in \Sigma^+$ by the following argument:

1. $pf_s \circ pf_i \in Pf(w)$ (P4),
2. $pf \in Pf(w)$ (by assumption),
3. $C(pf = pf_s \circ pf_i)$ (1, 2 and P1),
4. $C(pf \to pf_s \circ pf_i)$ (3 and PFDIntro),
5. $C(X \to pf)$ (by assumption), and
6. $C(X \to pf_s \circ pf_i)$ (5, 4 and PFDTrans).

Now observe that

$$C(pf_s \circ pf_1 \cdots pf_s \circ pf_m \to pf_s \circ pf_{m+1} \ldots pf_s \circ pf_n) \in \Sigma^+$$

is a consequence of the first condition above, P2 and PFDSubst. Therefore, for each $m < k \leq n$, $C(X \to pf_s \circ pf_k) \in \Sigma^+$ by PFDAdd, PFDTrans and PFDProj. P8 then implies $R_{G_1}.(pf_s \circ pf_k) = R_{G_2}.(pf_s \circ pf_k)$, and since $R_{G_1}.(pf_r \circ pf_k) = R_{G_1}.(pf_s \circ pf_k)$ (from above), $u.pf_k = v.pf_k$ for each $m < k \leq n$—contrary to the third condition above. □

4. Decision Procedures

In our introductory comments, we mentioned that the implication problem for PE constraints alone, over arbitrary schema, is undecidable. This follows from a straightforward reduction of the decision form of the word problem for monoids.

The latter problem takes as input a finite set of equations $E \cup \{e\}$ of the form "$f_i = f_j \circ f_k$". An interpretation I of the input is a domain D together with a total function over D for each symbol f occurring in the input. The interpretation satisfies $f_i = f_j \circ f_k$ if $f_i(d) = f_j(f_k(d))$ for all $d \in D$. The problem is to determine if every interpretation satisfying all equations in E must satisfy e.

Theorem 5: The implication problem for path equations alone is undecidable.

Proof. A reduction of the aforementioned word problem proceeds as follows. Letting $\{f_1, \ldots, f_n\}$ denote the set of all symbols mentioned in either E or e, include in the class schema S a single class scheme of the form

"`class C {` f_1`:C; ... ;` f_n`:C }`";

and for each equation $(f_i = f_j \circ f_k) \in E \cup \{e\}$, include in $\Sigma \cup \{\sigma\}$ a corresponding PE constraint `C(`$f_i = f_k.f_j$`)`. Clearly, since any domain D can be simulated by suitably populating class `C` with objects, $\Sigma \models_S \sigma$ if and

Table 9: Global data for procedures ASK-PE and ASK-PFD.

data	*description*
S	A class schema.
Σ_{PE}	A set of PE constraints over S.
Σ_{PFD}	A set of PFD constraints over S.
$G(V, A)$	A partial database for S.
Root	A distinguished vertex in G.
D	The current generation.

only if every interpretation I of $E \cup \{e\}$ satisfying all equations in E must satisfy e. The undecidability of the decision form of the word problem for monoids (e.g. see [21]) completes the proof. □

Conversely, if $PF(S)$ is finite, then one can clearly devise some procedure, based on exhaustively applying all inference axioms, that will decide membership in Σ^+ for an arbitrary set of constraints Σ. Recall that this happens exactly when S is acyclic. The procedures we shall now present are examples that may also be used to decide $\sigma \in \Sigma^+$ for cases in which Σ contains only key PFD constraints, and in which the class mentioned in σ is *stratified over* Σ (defined below). We shall also prove that they are semi-decision procedures in the general case.

Our procedures operate by creating and modifying a finite graph corresponding to a *partial database* for an input schema. We say that a graph $G(V, A)$ is a partial database for class schema S if the only condition not satisfied by G in order to qualify as a database for S is *property value completeness*; that is, we allow some vertices in V to be missing some of their property values.

Definition 11: A partial database for class schema S is a directed graph $G(V, A)$ with vertex and edge labels corresponding to class and property names respectively. G must also satisfy the conditions of property value integrity and property functionality as they apply to a (full) database. Also, in addition to its class label $Cl(v)$, each vertex in V is assigned a *generation* label, denoted $Gen(v)$. (The generation label is used to limit the eventual size of G.) □

To simplify the presentation, we assume all our procedures have access to the global data listed in Table 9. Essentially, our procedures operate by creating and manipulating the partial database included in the table.

Our first procedure is called ASK-PE and is listed in Figure 5. In addition to the data in Table 9, the procedure is supplied with a class name C and a set of PE constraints Σ on C. A third parameter, n, is a non-negative integer value which represents a limit on the number of *generations* of vertices which are added to the global partial database. This ensures termination of ASK-PE since the number of vertices in any given generation will always be finite (proven below).

ASK-PE attempts to determine if a given set of PE constraints Σ on a given class C are logical consequences of the PE and PFD constraints in Table 9. This is accomplished with the use of several utility routines given in Figures 6 and 7, which are responsible for the changes made to graph G in Table 9 following its initialization with a single vertex in Step 1 of ASK-PE. The routines, function FIND and procedures ADD-PROP and MERGE, are based on a dynamic version of the congruence closure algorithms in [27, 19] which were suggested earlier in [28]. The fact that G is a partial database together with another property of G essential to our proof of correctness of ASK-PE are stated in the following lemma. A proof is given in Appendix B.

Lemma 5: Consider an invocation of procedure ASK-PE with class C, PE constraints Σ and integer n as input. Let G_0 be the state of G in Table 9 after initialization in Step 1 of ASK-PE, and let $G_1, G_2, \ldots$ be the sequence of states of G occurring immediately after any calls from ASK-PE to procedure ADD-PROP, to function FIND or to procedure MERGE. Then the following conditions are true of G_i, for all i.

C1. G_i is a partial database for S.

C2. A path exists from *Root* to all vertices $v \in V_{G_i}$.

C3. Let pf_1 and pf_2 denote any two path functions describing any two (not necessarily distinct or simple) paths in G_i from *Root* to some vertex $v \in V_{G_i}$. Then $C(pf_1 = pf_2) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$. □

Theorem 6: (*correctness of* ASK-PE) Procedure ASK-PE always terminates. Also, if Σ consists of PE constraints on class C, then there exists an integer n such that, for any invocation of ASK-PE of the form

$$\text{ASK-PE}(C,\ \Sigma,\ m,\ \textit{Answer})$$

where $m \geq n$, *Answer* = "TRUE" if and only if $\Sigma \subseteq (\Sigma_{PE} \cup \Sigma_{PFD})^+$.

Proof. Clearly, procedure FIND must always terminate after adding at most $len(pf) - 1$ new vertices and arcs to G (by calling procedure ADD-PROP). This is also true of procedure MERGE if G is finite since any

Figure 5: PROCEDURE ASK-PE.

procedure ASK-PE(C, Σ, n, *Answer*)

Input: *a class name C, a set of PE constraints Σ on C and a generation bound n.*

Output: *Answer* = "TRUE" *only if* $\Sigma \subseteq (\Sigma_{PE} \cup \Sigma_{PFD})^+$*; otherwise, Answer* = "UNKNOWN".

Step 1. (*initialization*) Initialize G with a single vertex u, where $Cl(u) := C$ and $Gen(u) := 0$. $Root := u$. $D := 0$.

Step 2. For each vertex $u \in V_G$ such that $Gen(u) < D$, $P \in Props(Cl(u))$ and $u \overset{P}{\rightarrow} v \notin A_G$ for any vertex v, invoke ADD-PROP(u, P).

Step 3. $D := D + 1$. If $D \leq n$, then repeat from Step 2.

Step 4. (*check for violation of PE constraints*) For each $u \in V_G$ such that $Gen(u) < D$ and $Cl(u)(pf_1 = pf_2) \in \Sigma_{PE}$, invoke MERGE(FIND($u$, pf_1), FIND(u, pf_2)).

Step 5. (*check for violation of PFD constraints*) $D := D + 1$. For each $u, v \in V_G$ such that $u \neq v$, $Gen(u) < D$, $Gen(v) < D$, $C(X \rightarrow Y) \in \Sigma_{PFD}$ and $Cl(u) = Cl(v) = C$: if FIND(u, pf_i) = FIND(v, pf_i) for all $pf_i \in X$, then invoke MERGE(FIND(u, pf_j), FIND(v, pf_j)) for each $pf_j \in Y$.

Step 6. Remove each $C(pf_1 = pf_2) \in \Sigma$ such that FIND(*Root*, pf_1) = FIND(*Root*, pf_2). If Σ is empty, then *Answer* := "TRUE"; otherwise, *Answer* := "UNKNOWN". Return.

Figure 6: FUNCTION FIND AND PROCEDURE ADD-PROP.

function FIND(u, pf)

Input: *a vertex $u \in V_G$ and path function pf.*

Output: *returns a vertex $v \in V_G$ such that $u.pf = v$. (Note that* FIND *ensures v exists by invoking* ADD-PROP *to create any missing vertices and arcs in G.)*

Step 1. If $pf = \texttt{Id}$, then return u.

Step 2. (*Since $pf \neq \texttt{Id}$, we may assume $pf = P \circ pf_1$, for some property P and path function pf_1.*) If $u \xrightarrow{P} v \notin A_G$ for some vertex $v \in V_G$, then invoke ADD-PROP(u, P). Assuming v denotes the (possible newly created) vertex in V_G such that $u \xrightarrow{P} v \in A_G$, return FIND($v$, pf_1).

procedure ADD-PROP(u, P)

Input: *a vertex $u \in V_G$ and property P.*

Effect: *updates the global partial database G by creating a new vertex and arc representing a P property value for vertex u.*

Step 1. Add a new vertex v to V_G and arc $u \xrightarrow{P} v$ to A_G. Initialize the vertex labels for v as follows: $Cl(v) := Ran(Cl(u), P)$ and $Gen(v) := D$. Return.

Figure 7: Procedure MERGE.

procedure MERGE(u, v)

Input: *vertices $u, v \in V_G$. and property P.*

Effect: *updates the global partial database G by merging the subgraphs rooted at vertices u and v.*

Step 1. If $u = v$ then return.

Step 2. If $OutDegree(u) < OutDegree(v)$ then invoke MERGE(v, u) and return. (*Note that the notation "OutDegree(v)" represents the number of arcs originating from vertex v.*)

Step 3. If $Root = v$ then $Root := u$.

Step 4. For each vertex $w \in V_G$ and property P such that $w \xrightarrow{P} v \in A$, redirect $w \xrightarrow{P} v$ to u.

Step 5. For each vertex $w \in V_G$ and property P such that $v \xrightarrow{P} w \in A_G$ do the following. Remove $v \xrightarrow{P} w$ from A_G. If there exists vertex $x \in V_G$ such that $u \xrightarrow{P} x \in A_G$, then invoke MERGE($x$, w); otherwise, add $u \xrightarrow{P} w$ to A_G.

Step 6. Remove vertex v from V_G and return.

recursive calls imply at least one vertex is removed from G. Finally, the preconditions on the vertex label $Gen(v)$ ensure that any of Steps 2, 4 or 5 in procedure ASK-PE itself will terminate.

The "only if" part of the second assertion follows directly from Lemma 5. Now consider where no such n exists, and let G denote the partial database obtained by taking the limit of the sequence of graphs $G_0, G_1, \ldots$ resulting from the sequence of calls

ASK-PE(C, Σ, 0, *Answer*), ASK-PE(C, Σ, 1, *Answer*), ...

Since *Answer* = "UNKNOWN" for all calls, the *Root* vertex of G must be a C-object which fails to satisfy some PE constraint $\sigma \in \Sigma$. Also, by taking G to be the limit of this sequence of calls, a simple inspection of Step 2 of ASK-PE implies that G will also satisfy property value completeness, and is therefore a (full) database for S. Since Step 4 and Step 5 of ASK-PE ensure G satisfies all constraints in $\Sigma_{PE} \cup \Sigma_{PFD}$, the consequent follows by Theorem 1 (soundness). □

A simple example should help to clarify how procedure ASK-PE works. Assume the entries S and Σ_{PE} in Table 9 are assigned the sets

{ `a{ A: b }`, `b{ B: b }` }

and

{ `a( A = A.B.B.B )`, `a( A = A.B.B.B.B.B )` }

respectively, and also that Σ_{PFD} is empty. Observe that S is cyclic because of the `B` property of class `b`. Figure 8 presents a sequence of "snapshots" of the partial database G in Table 9 which result from a call to ASK-PE of the form

ASK-PE(`a`, {`a( A = A.B )`}, 0, *Answer*).

Note that the value of the generation label $Gen(v)$, for each vertex v, appears below its class label $Cl(v)$. Figure 8(a) indicates the state of G at the start of Step 4, and Figure 8(f) indicates the final state of G after ASK-PE returns in Step 6. Now consider this final state together with the function calls "FIND(*Root*, `A`)" and "FIND(*Root*, `A.B`)". Since the same vertex is returned for both calls, procedure ASK-PE will assign *Answer* the value "TRUE" before returning in Step 6. Thus, by Theorem 6 above and Theorem 1 (soundness), we can conclude that the PE constraint "`a( A = A.B )`" is a logical consequence of the two in Σ_{PE}.

To illustrate how this final state is reached, Figure 8(b) to (e) traces the state of G immediately prior to each of the four calls of procedure MERGE which occurs during the execution of Step 4 of ASK-PE. (Step 5

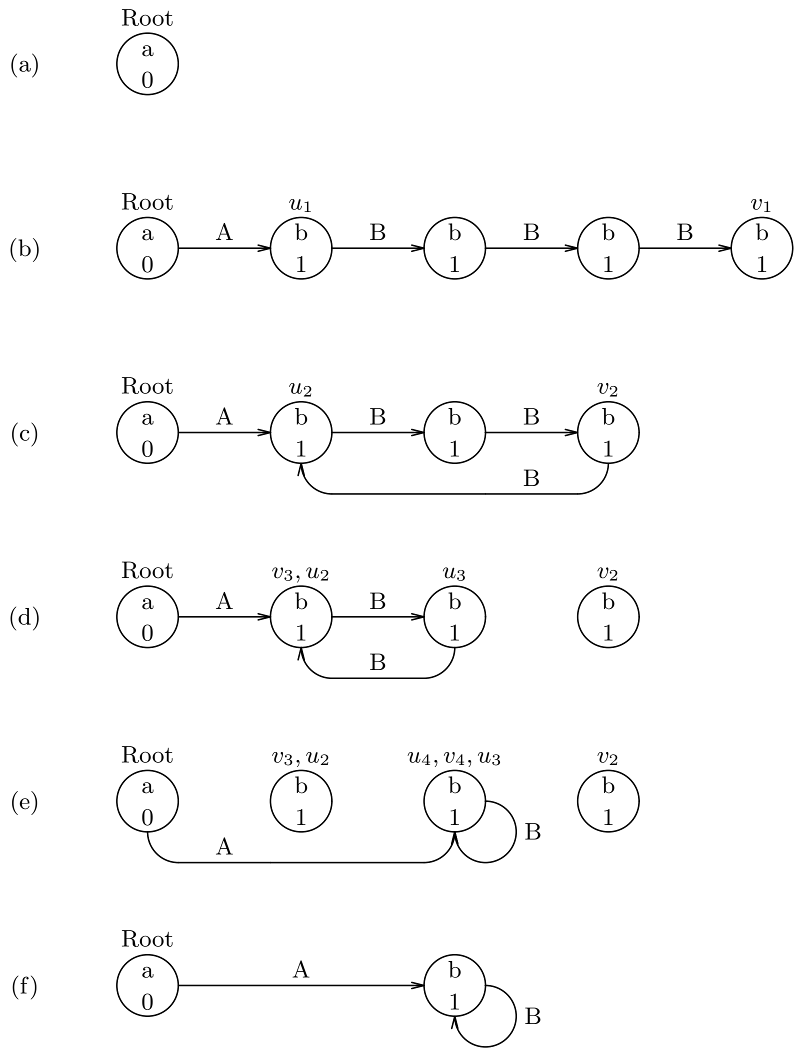


Figure 8: A sequence of partial databases produced by ASK-PE.

Figure 9: Procedure ASK-PFD.

```
procedure ASK-PFD($C$, $X$, $Y$, $n$, $Answer$)

Input: a class name $C$, a set of path functions $XY$ on $C$ and a generation bound $n$.

Output: $Answer$ = "TRUE" only if $C(X \to Y) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$; otherwise, $Answer$ =
"UNKNOWN".

Step 1.   Add class scheme "class Q{ A: $C$; B: $C$ }" to $S$ (assuming Q is not the name
          of any existing class in $S$). For each $pf_i \in X$, add "Q(A $\circ\, pf_i =$ B $\circ\, pf_i$)" to $\Sigma_{PE}$.

Step 2.   Let $\Sigma$ denote the set of all PE constraints of the form "Q(A $\circ\, pf_j =$ B $\circ\, pf_j$)", where
          $pf_j \in Y$. Invoke ASK-PE(Q, $\Sigma$, $n+1$, $Answer$), and return.
```

will have no effect on G since we have assumed Σ_{PFD} is empty.) Note that the bindings of parameters u and v for the ith call are indicated as vertices u_i and v_i respectively. The first two calls, Figure 8(b) and (c), correspond to the top level calls of MERGE, directly from Step 4, which process the two PE constraints in Σ_{PE}, starting with the constraint `a( A = A.B.B.B )`. The second two calls, Figure 8(d) and (e), correspond to the recursive calls of MERGE which result from the second of the two top level calls.

Our last procedure is called ASK-PFD, and is listed in Figure 9. ASK-PFD attempts to determine if a given PFD constraint $C(X \to Y)$ is in $(\Sigma_{PE} \cup \Sigma_{PFD})^+$. Roughly, the procedure operates by creating an additional class scheme and PE constraints, and then using ASK-PE to effectively simulate an attempt to create a partial database containing two C objects which agree on each of their X path functions, but disagree on one of their Y path functions. The additional class scheme, called `Q`, is a simple expedient to "force" the existence of the requisite C objects. The following theorem establishes that this works correctly, and that ASK-PFD therefore qualifies as a semi-decision procedure for PFD constraints. Since this case is a bit more involved than the previous case relating to PE constraints, the complete proof can be found in Appendix B.

Theorem 7: (*correctness of* ASK-PFD) Procedure ASK-PFD always terminates. Also, if X and Y denote two sets of path functions on class C, then there exists an integer n such that, for any invocation of ASK-PFD of the form

$$\text{ASK-PFD}(C,\ X,\ Y,\ m,\ \mathit{Answer})$$

where $m \geq n$, *Answer* = "TRUE" if and only if $C(X \rightarrow Y) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$. □

To illustrate how procedure ASK-PFD works, we return to the example UNIVERSITY query given in the introduction. First, assume entry S in Table 9 is the class schema presented in Table 3 with the following complex object type representing the query class illustrated in Figure 2 added.

```
class Query { P1:String; P2:Integer; S:Student; E:Enrollment; C:Course }
```

And second, assume entries Σ_{PE} and Σ_{PFD} in Table 9 are assigned the PE and PFD constraints occurring in Tables 1 and 2. Now consider a call to ASK-PFD of the form

ASK-PFD(`Query`, {`P1`, `P2`, `S`}, {`E`, `C`}, 2, *Answer*).

Step 1 modifies entries S and Σ_{PE} in Table 9 by adding the object type

```
class Q{ A: Query; B: Query }
```

to the former and the PE constraints

`Q( A.P1 = B.P1 )`, `Q( A.P2 = B.P2 )` and `Q( A.S = B.S )` (4.1)

to the latter. Procedure ASK-PFD then returns in Step 2 after a call to procedure ASK-PE of the form

ASK-PE(`Q`, {`Q( A.E = B.E )`, `Q( A.C = B.C )`}, 3, *Answer*).

Figure 10 illustrates the state of the partial database G in Table 9 subsequent to the return from this call. Since the `E` and `C` property values for the two `Query` objects coincide, procedure ASK-PE assigns *Answer* the value "TRUE" in Step 6. Thus, by Theorem 7, the results of the original call to ASK-PFD can be interpreted as confirmation that PFD constraint (1.2) in the introduction is a logical consequence of the constraints in Tables 1 and 2.

To clarify how this final state is reached, we have added tabs on eight of the vertices (i.e. the small squares). The numbering on the tabs indicates a sequence of the most relevant calls to procedure MERGE which eventually merges all subgraphs of the two `Query` objects. These calls to MERGE occur in Steps 4 and 5 of ASK-PE, and recursively from MERGE itself. We consider each of the tabbed cases in sequence.

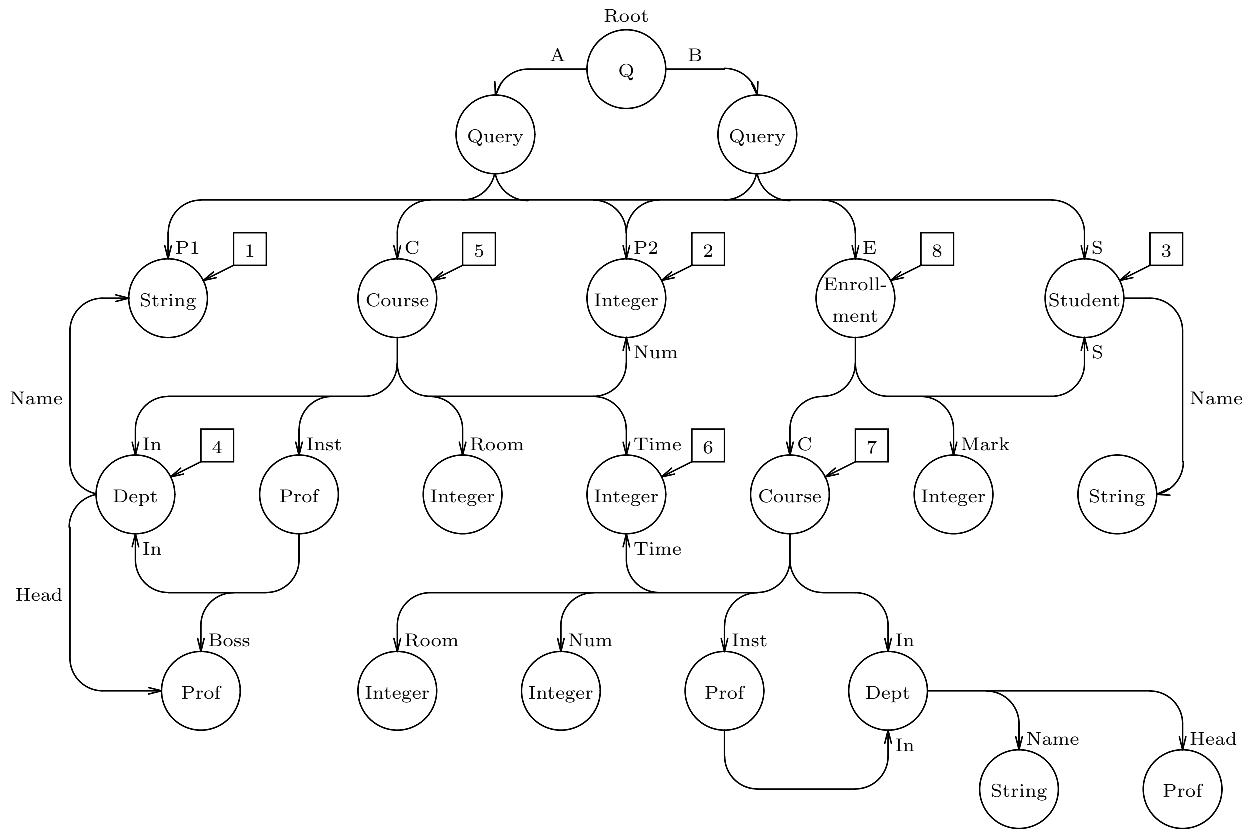


Figure 10: A partial database produced by ASK-PFD.

- (*tabs numbered 1 to 3*) At the start of Step 4 of procedure ASK-PE, all arcs outgoing from the two `Query` objects point to disjoint subgraphs. The separate subgraphs for the `P1`, `P2` and `S` property values of the two `Query` objects are merged sometime during the execution of Step 4 because of additional PE constraints, (4.1) above, which were added to Σ_{PE}.

- (*tab number 4*) Since two `Dept` objects now have the same `Name` property value, Step 5 of ASK-PE merges the corresponding subgraphs because of the constraint "`Dept( Name` $\rightarrow$ `Id )`" in Σ_{PFD}.

- (*tabs numbered 5 and 6*) Subgraphs rooted at the separate `Course` objects which were the `C` property values of the two `Query` objects are now merged because of the constraint "`Course( Num In` $\rightarrow$ `Id )`" in Σ_{PFD}.

- (*tab number 7*) Subgraphs rooted at the two other `Course` objects are merged because of the constraint "`Enrollment( S C.Time` $\rightarrow$ `C )`" in Σ_{PFD}. (According to the cases for tabs numbered 3 and 6 discussed above, their parent `Enrollment` objects will now agree on the values of path functions `S` and `C.Time`.)

- (*tab number 8*) Finally, the subgraphs rooted at the two `Enrollment` objects which were the `E` property values of the two `Query` objects are merged because of the constraint "`Enrollment( S C` $\rightarrow$ `Id )`" in Σ_{PFD}.

As outlined at the start of the section, one case in which procedures ASK-PE and ASK-PFD can be used to decide $\sigma \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$ happens when the problem schema satisfies two additional conditions. The first requires Σ_{PFD} to consist of only key PFD constraints. The second is a stratification condition for Σ_{PE} which effectively limits the number of possible applications of the substitution axioms PESubst. A formal definition of this second condition is as follows.

Definition 12: Let Σ denote a set of PE constraints over a class schema S. Then $C_1 \in Classes(S)$ is *stratified over* Σ if there exists an integer n satisfying

$$pf \in PathFuncs(C_1),\ Ran(C_1, pf) = C_2 \text{ and } C_2(pf_1 = pf_2) \in \Sigma \text{ imply } len(pf) \leq n.$$

If n is minimal, then we say that C_1 is stratified over Σ *with size* n. □

Note that a breadth first search of a class schema graph may be employed to efficiently decide questions of the form: "Is class C stratified over PE constraints Σ with size n?". Also note that our stratification condition continues to admit some forms of cyclic schema, and that all classes necessarily satisfy the condition for acyclic schema (regardless of the selection of PE constraints). For example, in the case illustrating the operation of procedure ASK-PE discussed above, class `a` is stratified with respect to the two PE constraints with size 0. In the subsequent case illustrating the operation of procedure ASK-PFD, class `Query` would satisfy a stratification condition of size 2 were it not for the PE constraint "`Prof( Boss = In.Head )`" in Table 1.

Theorem 8: Assume $C \in Classes(S)$ is stratified over PE constraints Σ_{PE} with size n, and also that Σ_{PFD} consists only of key PFD constraints. Also let Σ denote a set of PE constraints on C, and let X and Y denotes sets of path functions in $PathFuncs(C)$.

1. A call to ASK-PE of the form

$$\text{ASK-PE}(C,\ \Sigma,\ n,\ \mathit{Answer})$$

decides if $\Sigma \subseteq (\Sigma_{PE} \cup \Sigma_{PFD})^+$; that is, Answer = "TRUE" if and only if $\Sigma \subseteq (\Sigma_{PE} \cup \Sigma_{PFD})^+$.

2. A call to ASK-PFD of the form

ASK-PFD(C, X, Y, n, $Answer$)

decides if $C(X \rightarrow Y) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$; that is, $Answer$ = "TRUE" if and only if $C(X \rightarrow Y) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$.

Proof. Consider the first part of the theorem, letting G_1 be the state of the global partial database G after invocation of ASK-PE, and G_2 the new state that results after a call to function FIND($Root$, pf), for all $pf \in PathFuncs(C)$. First observe that all three conditions of Lemma 5 continue to be satisfied by G_2, in light of its proof, and that G_2 must also satisfy property value completeness by virtue of the calls to FIND. Thus, G_2 qualifies as a (full) database. Since $Gen(v) > n$, for all $v \in V_{G_2} - V_{G_1}$, and since C is stratified over Σ_{PE} with size n, no vertex v exists in V_{G_2} such that $Gen(v) > n$, and such that $Cl(v) = C_1$ for some $C_1(pf_1 = pf_2) \in \Sigma_{PE}$. G_2 must therefore satisfy all constraints in Σ_{PE}, according to the fourth step of ASK-PE.

Now consider $C(X \rightarrow \mathtt{Id}) \in \Sigma_{PFD}$ together with any choice of two succinct vertices $u, v \in V_{G_2}$ such that $Cl(u) = Cl(v) = C$. Without loss of generality, assume $Gen(u) > n + 1$. Then there are two possibilities for when u was added: 1) $u \in (V_{G_2} - V_{G_1})$, or 2) u was added by virtue of a call to FIND from Step 5 of ASK-PE. In either case, it is not possible for u and v to agree on any $pf \in X$.[4] Therefore, according to Step 5 of ASK-PE, G_2 satisfies all constraints in Σ_{PFD}, and the first part of the theorem follows by Lemma 5 and Theorem 1 (soundness).

The proof of the second part of the theorem is a simple consequence of the first and Theorem 7. □

A number of factors affect the running time of a call to procedure ASK-PE. For example, the class schema itself and the value of parameter n determine the size of the partial database created by the first three steps of the procedure. In particular, is it straightforward to prove that V_G will consist of one vertex for each $pf \in PathFuncs(C)$ such that $len(pf) \leq n$, and that G itself will be a tree rooted at vertex $Root$. Since a class scheme will usually include more than one property in its definition, the expected running time for the first three steps is therefore likely to remain exponential in n.

Table 10 summarizes execution time bounds for the various steps of procedure ASK-PE in terms of the number of calls to procedure ADD-PROP, either directly in Step 3 or indirectly by virtue of a call to function FIND, and to procedure MERGE. The formulas assume that $|S|$ denotes the number of different property names in a given class schema S, and that $|\Sigma_{PE}|$ (resp. $|\Sigma_{PFD}|$ and $|\Sigma|$) denote the number of property name

[4]This would not be the case if we permitted non-key PFD constraints.

Table 10: TIME BOUNDS FOR PROCEDURE ASK-PE.

steps	ADD-PROP	MERGE
1, 2 and 3	$O(k^n)$	N/A
4	$O(\|\Sigma_{PE}\| \cdot k^n)$	$O(k^n)$
5	$O(\|\Sigma_{PFD}\| \cdot (\|\Sigma_{PE}\| \cdot k^n)^2)$	$O(\|\Sigma_{PFD}\| \cdot \|\Sigma_{PE}\| \cdot k^n)$
6	$O(\|\Sigma\|)$	N/A

where k is $O(|S|)$

Table 11: A WORST-CASE FOR ASK-PE.

S	Σ_{PE}	Σ_{PFD}
`class R` $\{A_1$`:R;` ... `;` A_m`:R; B: S`$\}$	`S(`$C_1.D_1 = C_{m+1}.D_1$`)`	`T(`$D_1 \rightarrow D_{m+1}$`)`
`class S` $\{C_1$`:T;` ... `;` C_{2m}`:T`$\}$	...	...
`class T` $\{D_1$`:U;` ... `;` D_{2m}`:U`$\}$	`S(`$C_m.D_m = C_{2m}.D_m$`)`	`T(`$D_m \rightarrow D_{2m}$`)`
`class U { }`		

occurrences in Σ_{PE} (resp. Σ_{PFD} and Σ). Also, the formulas for the fifth step assume (reasonably) that Σ_{PE} is non-empty. The bounds derive straightforwardly from two observations.

1. The number of indirect calls to procedure ADD-PROP from function FIND is bound by the sum of the argument lengths for all top-level calls to FIND from ASK-PE.

2. Since a vertex is removed from G for each recursive call of procedure MERGE (from itself), the total number of calls to the procedure is also bound by the number of top-level calls from ASK-PE.

Problem schema and constraints which have the pattern outlined in Table 11 are cases which demonstrate that the bounds in Table 10 for the first five steps are all tight, given a call of the form: "ASK-PE(`R`, { }, n, *Answer*)".

Appendix A describes a data structure which may be employed to achieve the best possible asymptotic execution times for the "standard" directed labeled graph access and manipulation operations. Adopting this data structure, all calls to procedure FIND will run in time linear in the length of the argument path function, and procedure ADD-PROP will run in constant time.

Now let k denote the maximum number of properties included in a class definition for a given problem schema, which is $O(|S|)$, and let m denote the total number of vertices created in all calls to procedure ADD-PROP during an invocation of ASK-PE. The same choice of data structure outlined in Appendix A together with the second observation above then implies that the total run time for all calls to procedure MERGE will be dominated by the execution time for its fourth step, and for the cases in its fifth step in which a recursive call is not made.

First consider the latter. The second step of MERGE ensures that the "from" vertex of any given arc is never changed more than $O(\log k)$ times. Since the number of arcs never exceeds m, a time bound for these cases is $O(m \cdot \log k)$.

Now consider the overhead for the fourth step. Since this step is essentially performing a disjoint set union operation, in which the set associated with each vertex is its incoming arcs, a fast disjoint-set union algorithm (such as Algorithm 4.3 in [3]) can be used to achieve a worst case run time of $O(m \cdot \alpha(m))$, where α is the single-variable inverse Ackermann function.

In summary, assuming each of Σ_{PE} and Σ_{PFD} are non-empty, execution time for procedure ASK-PE is bound by the number of possible indirect calls to procedure ADD-PROP from Step 5 and Step 6, which is

$$O(|\Sigma_{PFD}| \cdot (|\Sigma_{PE}| \cdot k^n)^2 + |\Sigma|).$$

Also, we believe the expected case run-time can be considerably improved by more careful indexing of vertices with respect to their class labels. It is straightforward, for example, to maintain a list for each class C of all vertices v where $Cl(v) = C$.

5. Summary

We have presented a sound and complete axiomatization for the combination of a form of equational constraint and functional dependency constraint for a data model supporting complex object types. Both kinds of constraints may be considered special cases of a general category of "path constraints" in which component attributes may correspond to descriptions of property value paths. Also presented were decision procedures for the implication problem for both kinds of constraints when the problem schema satisfies a stratification condition, and when all input functional dependencies are keys.

In our introductory comments, we reviewed a number of applications of our theory. Indeed, our own experience is that procedures like those presented in the previous section are valuable components of object-oriented query optimizers.

Acknowledgements

The authors are grateful for the many excellent comments and suggestions made by the anonymous referees. Thanks also go to Frank Tompa and Ed Chan for their helpful discussions on the topic.

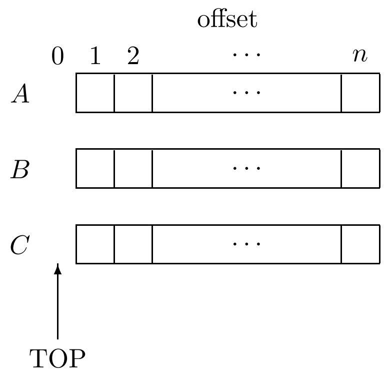
6. APPENDIX A

The technique outlined in problem 2.12 of [3] for initializing an entry in a matrix to zero the first time it is accessed can be simply adapted to support best possible asymptotic execution times for a number of standard directed labeled graph access and manipulation operations required by our decision procedures. Recall that edge labels in our case correspond to property names, and that no vertex has more than one outgoing edge with the same label.

The required operations on a graph $G(V, A)$ are as follows.

1. Add a new vertex to V.
2. Given $u, v \in V$ and property P, add a new arc $u \xrightarrow{P} v$ to A.
3. Given $u \in V$ and property P, determine if there exists $v \in V$ such that $u \xrightarrow{P} v \in A$. Return v if this is the case.
4. Given $u \in V$, find all properties P and $v \in V$ such that $u \xrightarrow{P} v \in A$.

The technique first requires that property names are mapped to a unique integer offset in the range $[1, \ldots, n]$, for some integer n. When a vertex is first created, it is then allocated $3n + 1$ units of store organized in three parallel arrays and a TOP index as suggested in [7].



To add a new arc with offset i to vertex v, increment TOP, and set $A[i]$, $B[\text{TOP}]$ and $C[i]$ the values TOP, i and "pointer to v" respectively. An arc with offset j exists if: 1) $1 \leq A[j] \leq \text{TOP}$, and 2) $j = B[A[j]]$. Finally, the offset of all outgoing arcs can be found by scanning array B in the range $[1, \ldots, \text{TOP}]$.

7. APPENDIX B

Proof of Lemma 3.

We prove properties P1 to P4 in sequence.

(*proof of P1*)

(if part) Follows directly from the definition of the construction.

(only if part) Without loss of generality, consider an arbitrary $u \in V$. We prove by induction that there exists $S \subseteq Pf(u)$ of arbitrary size satisfying P1. For any subset $S = \{pf\}$ of $Pf(u)$, consisting of a single path function, $C(pf = pf) \in \Sigma^+$ by the reflexivity axiom PERef. Now assume P1 holds for some subset $S' = \{pf_1, \ldots, pf_n\} \subset Pf(u)$ of size n. By definition of the construction, there must exist $pf_1 \in S'$ and $pf_2 \in (Pf(u) - S')$ such that either $C(pf_1 = pf_2) \in \Sigma^+$ or $C(pf_2 = pf_1) \in \Sigma^+$ (since otherwise, we have not created a maximum partitioning of $PathFuncs(C)$). In either case, by the induction assumption together with the remaining equality axioms PESym and PETrans, the set $S = S' \cup \{pf_2\}$ also satisfies P1.

(*proof of P2*)

The first step of the construction assigns $Cl(u)$ the class $Ran(C, pf_1)$ for some arbitrary $pf_1 \in Pf(u)$. By P1, $C(pf_1 = pf_2) \in \Sigma^+$, for all $pf_2 \in Pf(u)$, and therefore $Ran(C, pf_1) = Ran(C, pf_2)$ by Theorem 1. (The inference axioms derive only well-formed constraints.)

(*proof of P3*)

We must show that the constraints we have imposed on a directed labeled graph in order to qualify as an interpretation are satisfied. First consider property value integrity. If $u \xrightarrow{P} v \in A$, then there exists some $pf \in Pf(u)$ where $pf \circ P \in Pf(v)$, according to the second step of the construction of G. Since $pf, (pf \circ P) \in PathFuncs(C)$, $P \in Props(Ran(C, pf))$ by definition, $Cl(u) = Ran(C, pf)$ by P2, and therefore $P \in Props(Cl(u))$. It also follows by P2, and by definition of the composition operator and of well-formed path functions that:

$$Cl(v) = Ran(C, pf \circ P) = Ran(Ran(C, pf), P) = Ran(Cl(u), P).$$

Now consider property functionality. If $u \xrightarrow{P} v, u \xrightarrow{P} w \in A$, then, according to the second step of the construction, there exists $pf_1, pf_2 \in Pf(u)$ such that $pf_1 \circ P \in Pf(v)$ and $pf_2 \circ P \in Pf(w)$. But then $C(pf_1 = pf_2) \in \Sigma^+$ by P1, and therefore $C(pf_1 \circ P = pf_2 \circ P) \in \Sigma^+$ by the attribution axiom PEAttr. Thus, $pf_1 \circ P$ and $pf_2 \circ P$ must occur in the same partition P_i, and therefore $v = w$.

Finally, property value completeness is a simple consequence of the definition of path functions, of P2 and of the second step of the construction.

(*proof of P4*)

We prove P4 by induction on the length of pf. If $len(pf) = 0$, then pf is $\texttt{Id}$, $R.\texttt{Id} = R$ and $\texttt{Id} \in Pf(R)$. Now consider where pf has the form $pf_1 \circ P$, for some property P.

(if part) Choose $u, v \in V$ such that $pf_1 \circ P \in Pf(v)$ and $u = R.pf_1$. By the inductive assumption, $pf_1 \in Pf(u)$, and therefore the second step of the construction adds $u \xrightarrow{P} v$ to A. But then $R.(pf_1 \circ P) = v$, by P3.

(only if part) Choose $u \in V$ as before, and $v \in V$ such that $v = R.(pf_1 \circ P)$. The second step of the construction added $u \xrightarrow{P} v$ to A, which implies there exists $pf_2 \in Pf(u)$ such that $pf_2 \circ P \in Pf(v)$. By the inductive assumption, $pf_1 \in Pf(u)$, and therefore $C(pf_1 = pf_2) \in \Sigma^+$ by P1. But then $C(pf_1 \circ P = pf_2 \circ P) \in \Sigma^+$, by the attribution axiom PEAttr. Thus, $pf_1 \circ P$ and $pf_2 \circ P$ occur in the same partition P_i, and therefore $pf_1 \circ P \in Pf(v)$. $\square$

Proof of Lemma 4.

The construction of any *Two-C-Graph G* starts from two *C-Graphs*. Since neither of the two steps modifies any *Pf* labeling, the proofs of P1 and P2 for Lemma 3 apply unchanged. We prove each of the remaining properties in sequence.

(*proof of P3*)

It suffices to show that property value integrity is not violated at some point during the second step. If $u \xrightarrow{P} v$ is added in the second step, then clearly $Cl(u) \in Dom(P)$, and for some $pf \in Pf(u)$, $R_{G_1}.(pf \circ P) = v$. Since no arcs are added to any vertex originating in G_1, P4 then implies $pf \circ P \in Pf(v)$, and therefore $Ran(C, pf \circ P) = Cl(v)$ by P2. Integrity follows since the definition of (well-formed) path functions and P2 (again) implies

$$Ran(C, pf \circ P) = Ran(Ran(C, pf), P) = Ran(Cl(u), P).$$

(*proof of P5*)

If R_{G_2} is removed during the first step, then there must exist $pf_1 \in Pf(R_{G_2})$ such that $C(X \to pf_1) \in \Sigma^+$. Since $\texttt{Id} \in Pf(R_{G_2})$, $C(pf_1 = \texttt{Id}) \in \Sigma^+$ by P1, and therefore $C(pf_1 \to \texttt{Id}) \in \Sigma^+$ by PFDIntro. But PFDAttr and two uses of PFDTrans derive $C(X \to pf_2)$ for each $pf_2 \in Y$, and thus by PFDAdd: $C(X \to Y) \in \Sigma^+$—contrary to assumptions.

(*proof of P6*)

Assume u is removed in the first step since $C(X \to pf_1) \in \Sigma^+$ for some $pf_1 \in Pf(u)$. According to Lemma 2 and P3, if v is reachable from u, then there must exist some $pf_2 \in PathFuncs(Cl(u))$ such that

$u.pf_2 = v$. Thus $pf_1 \circ pf_2 \in Pf(v)$ by two uses of P4. But then $C(X \rightarrow pf_1 \circ pf_2) \in \Sigma^+$ by the following argument:

1.	$Cl(u)(\texttt{Id} \rightarrow pf_2)$	(PFDAttr),
2.	$Ran(C, pf_1)(\texttt{Id} \rightarrow pf_2)$	(1 and P2),
3.	$C(pf_1 \rightarrow pf_1 \circ pf_2)$	(2 and PFDSubst),
4.	$C(X \rightarrow pf_1)$	(by assumption) and
5.	$C(X \rightarrow pf_1 \circ pf_2)$	(4, 3 and PFDTrans).

Thus v must also be removed in the first step.

(*proof of P7*)

We prove P7 by induction on the length of pf_2. The basis case holds since $len(pf_2) = 0$ implies pf_2 is `Id`, and therefore that $u = v$. Now assume pf_2 has the form $pf_3 \circ P$, and that $u.pf_3 = w$. By the inductive assumption, $pf_1 \circ pf_3 \in Pf(w)$ for all $pf_1 \in Pf(u)$. P2 and P3 together imply there exists $w \xrightarrow{P} v \in A$. If $w \xrightarrow{P} v$ was not added to A during the second step of the construction, then P5 and P6 imply that w and v originated from the same *C-Graph* Thus $pf_1 \circ pf_3 \circ P \in Pf(v)$ follows by two uses of P4. If $w \xrightarrow{P} v$ was added to A during the second step of the construction, then $R_{G_1}.(pf_1 \circ pf_4 \circ P) = v$ for some $pf_1 \circ pf_4 \in Pf(w)$, and therefore $pf_1 \circ pf_4 \circ P \in Pf(v)$ by P4. Since $Pf(w)$ contains both $pf_1 \circ pf_3$ and $pf_1 \circ pf_4$, $C(pf_1 \circ pf_3 = pf_1 \circ pf_4) \in \Sigma^+$ by P1, and therefore $C(pf_1 \circ pf_3 \circ P = pf_1 \circ pf_4 \circ P)$ follows by JCAttr. Thus $pf_1 \circ pf_3 \circ P \in Pf(v)$ follows by P1.

(*proof of P8*)

Proof of P8 is also by induction, this time on the length of pf. The basis case follows directly from P5, and then if pf has the form $pf_1 \circ P$, there are two cases to consider.

If $C(X \rightarrow pf_1) \in \Sigma^+$, then $C(X \rightarrow pf_1 \circ P) \in \Sigma^+$ by an argument entirely analogous to the proof above of P6. Also, by the inductive assumption, $R_{G_1}.pf_1 = R_{G_2}.pf_1$, and therefore $R_{G_1}.(pf_1 \circ P) = R_{G_2}.(pf_1 \circ P)$ follows by P3.

If $C(X \rightarrow pf_1) \notin \Sigma^+$, then $R_{G_1}.pf_1 \neq R_{G_2}.pf_1$ by the inductive assumption. Thus $R_{G_1}.(pf_1 \circ P) = R_{G_2}.(pf_1 \circ P)$ if and only if the second step of the construction adds arc $R_{G_2}.pf_1 \xrightarrow{P} R_{G_1}.(pf_1 \circ P)$ to A. But this happens if and only if the first step has removed vertex $R_{G_2}.(pf_1 \circ P)$ from V_2, which in turn can happen if and only if there exists $pf_2 \in Pf(R_{G_2}.(pf_1 \circ P))$ such that $C(X \rightarrow pf_2) \in \Sigma^+$. If such a pf_2 exists and is not

$pf_1 \circ P$, then $C(X \rightarrow pf_1 \circ P) \in \Sigma^+$ by the following argument:

1.	$C(pf_2 = pf_1 \circ P)$	(P1),
2.	$C(pf_2 \rightarrow pf_1 \circ P)$	(1 and PFDIntro),
3.	$C(X \rightarrow pf_2)$	(by assumption) and
4.	$C(X \rightarrow pf_1 \circ P)$	(3, 2 and PFDTrans).

□

Proof of Lemma 5.

Proof is by induction on the sequence of states G_i. In the basis case, G_0 consists of the single *Root* vertex where $Gen(Root) = 0$, and therefore must clearly satisfy the first two conditions. Also, since the only path in G_0 consists of the single *Root* vertex which can only be described by `Id`, condition C3 also holds since $C(\texttt{Id} = \texttt{Id})$ follows by axiom PERef.

Now assume the conditions hold up to state G_i, for some $i > 0$. There are two general cases to consider.

Case 1: G_i occurs after a call to ADD-PROP or to FIND. The three conditions are a simple consequence of the inductive assumption, axiom PEAttr and a simple induction on the (possibly empty) set of new vertices and arcs created by procedure ADD-PROP by virtue of an indirect call from function FIND.

Case 2: G_i occurs after a call to MERGE from ASK-PE. A simple inspection of the body of MERGE suffices to confirm the first two conditions. Now consider the third. We shall say that a call of MERGE is *justified* if there exists $C(pf_1 = pf_2) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$ such that pf_1 and pf_2 describe two paths from *Root* to vertices u and v respectively. First consider where a call of MERGE is justified. If MERGE returns in Step 1, then there is no change to G. If MERGE returns in Step 2, then the recursive call to MERGE in Step 2 is also clearly justified. If MERGE does not return in the first two steps, then axioms PESym and PETrans imply that the second condition remains true of vertex u at the end of Step 4, and PEAttr implies this is also true of any vertex reachable from u or v. If each of u and v have an outgoing arc with the same property label P, then axiom PEAttr also implies that the recursive call of MERGE in Step 5 is justified.

What remains is to prove that the original top-level call to procedure MERGE from ASK-PE when G was in state G_{i-1} is justified, of which there are two possible sites within ASK-PE itself. If this call to MERGE occurs in Step 4 of ASK-PE, then, by the inductive assumption, there exists a path from *Root* to vertex u in the partial database G_{i-1} described by some path function $pf \in PathFuncs(C)$, and this call of MERGE is then justified by axiom PESubst. If the call occurs in step 5, then, by the inductive assumption, there exists paths from *Root* to vertices u and v in the partial database G_{i-1} described by two path functions $pf_1, pf_2 \in PathFuncs(C)$ respectively, and this call of MERGE is then justified by axiom PEIntro. □

Proof of Theorem 7.

The fact that ASK-PFD always terminates is a direct consequence of Theorem 6, while the "only if" part of the second assertion follows from Lemma 5 and axiom PEIntro.

Now consider the "if" part of the second assertion, letting L_i denote either property `A` or property `B`. We first prove by induction on the length of derivation of a PE constraint σ of the form $\mathtt{Q}(L_1 \circ pf_1 = L_2 \circ pf_2)$ the two conditions:

1. $C(pf_1 = pf_2) \in (\Sigma_{PE} \cup \Sigma_{PFD})^+$, and
2. if $L_1 \neq L_2$, then $\{C(X \to pf_1), C(X \to pf_2)\} \subseteq (\Sigma_{PE} \cup \Sigma_{PFD})^+$.

If the derivation consists of a single step, then $\sigma \in \Sigma_{PE}$, or σ follows by axiom PERef. In either case, both conditions are clearly satisfied.

Now consider a derivation of σ of length $n > 1$. There are five cases corresponding to the five possible axioms justifying the final step in its derivation.

Case 1: axiom PESym. Both conditions are a simple consequence of the inductive assumption.

Case 2: axiom PETrans. Assume axiom PETrans is used to derive $\mathtt{Q}(L_1 \circ pf_1 = L_3 \circ pf_3)$ from earlier derivations of $\mathtt{Q}(L_1 \circ pf_1 = L_2 \circ pf_2)$ and $\mathtt{Q}(L_2 \circ pf_2 = L_3 \circ pf_3)$. Then both $C(pf_1 = pf_2)$ and $C(pf_2 = pf_3)$ can be derived according to the inductive assumption, and $C(pf_1 = pf_3)$ follows also by PETrans. Now consider the second condition. If L_1 is not the same property as L_3, then either $L_1 \neq L_2$ or $L_2 \neq L_3$ (or both). With loss of generality, assume the latter. Then the second condition holds by the following argument:

1. $C(pf_1 = pf_3)$ (first condition),
2. $C(pf_3 = pf_1)$ (1 and PESym),
3. $C(pf_3 \to pf_1)$ (2 and PFDIntro),
4. $C(X \to pf_3)$ (inductive assumption) and
5. $C(X \to pf_1)$ (4, 3 and PFDTrans).

Case 3: axiom PESubst. Assume axiom PESubst is used instead to derive $\sigma = \mathtt{Q}(L \circ pf \circ pf_1 = L \circ pf \circ pf_2)$ from an earlier derivation of a PE constraint of the form $Ran(\mathtt{Q}, L \circ pf)(pf_1 = pf_2)$. Clearly, the second condition is satisfied, and a reuse of PESubst derives $C(pf \circ pf_1 = pf \circ pf_2)$.

Case 4: axiom PEAttr. Now assume $\sigma = \mathtt{Q}(L_1 \circ pf_1 \circ pf = L_2 \circ pf_2 \circ pf)$ follows by axiom PEAttr from an earlier derivation of $\mathtt{Q}(L_1 \circ pf_1 = L_2 \circ pf_2)$. Then the first condition follows simply by the inductive assumption and a reuse of axiom PEAttr. If $L_1 \neq L_2$, then we can derive $C(X \to pf_1 \circ pf)$ by the following argument:

1.	$C(X \to pf_1)$	(inductive assumption),
2.	$Ran(C, pf_1)(\mathtt{Id} \to pf)$	(PFDAttr),
3.	$C(pf_1 \to pf_1 \circ pf)$	(2 and PFDPAug) and
4.	$C(X \to pf_1 \circ pf)$	(1, 3 and PFDTrans).

The case for $C(X \to pf_2 \circ pf)$ is analogous.

Case 5: axiom PEIntro. Finally, assume $\sigma = \mathtt{Q}(L_1 \circ pf_r \circ pf = L_2 \circ pf_s \circ pf)$ follows by axiom PEIntro from an earlier derivation of $Ran(\mathtt{Q}, L_1 \circ pf_r)(pf_1 \cdots pf_m \to Y)$, where $pf \in Y$, and of m earlier derivations of $\mathtt{Q}(L_1 \circ pf_r \circ pf_i = L_2 \circ pf_s \circ pf_i)$, $1 \leq i \leq m$. Note that this implies:

$$Ran(\mathtt{Q}, L_1 \circ pf_r) = Ran(\mathtt{Q}, L_2 \circ pf_s) = Ran(C, pf_r) = Ran(C, pf_s).$$

Again, the first condition follows simply by the inductive assumption and a reuse this time of axiom PEIntro. If $L_1 \neq L_2$, then we can derive $C(X \to pf_r \circ pf)$ as follows:

1.	$C(X \to pf_r \circ pf_1)$	(inductive assumption),
$\ldots$	$\ldots$	$\ldots$
m.	$C(X \to pf_r \circ pf_m)$	(inductive assumption),
$\ldots$	$\ldots$	$\ldots$
$2m-1$.	$C(X \to pf_r \circ pf_1 \cdots pf_r \circ pf_m)$	(1 to m, and $m-1$ uses of PFDAdd),
$2m$.	$Ran(C, pf_r)(pf_1 \cdots pf_m \to Y)$	(given),
$2m+1$.	$C(pf_r \circ pf_1 \cdots pf_r \circ pf_m \to pf_r \circ Y)$	($2m$ and PFDSubst),
$2m+2$.	$C(X \to pf_r \circ Y)$	($2m-1$, $2m+1$ and PFDTrans) and
$2m+3$.	$C(X \to pf_r \circ pf)$	($2m+2$ and PFDProj).

And again, the case for $C(X \to pf_s \circ pf)$ is analogous.

The "if" part of the second assertion is now a simple consequence of the second condition and axiom PFDAdd. □

References

[1] S. Abiteboul and S. Grumbach. COL: a logic-based langauge for complex objects. In *Proc. 1st International Conference on Extending Database Technology (EDBT)*, pages 271–293, 1988.

[2] S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 159–173, June 1989.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[4] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, December 1979.

[5] C. Beeri. Formal models for object-oriented databases. In *Proc. 1st International Conference on Deductive and Object-Oriented Databases*, pages 370–395, December 1989.

[6] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5:353–382, 1990.

[7] J. Bentley. Programming pearls. *Comm. of the ACM*, 26(9):623–628, September 1983.

[8] E. Bertino. An indexing technique for object-oriented databases. In *Proc. 7th International Conference on Data Engineering*, pages 160–170, April 1991.

[9] E. Bertino and W. Kim. Indexing techniques for queries on nested objects. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):196–214, June 1989.

[10] G. Birkhoff. On the structure of abstract algebras. *Proc. of the Cambridge Philisophical Society*, 31(4):433–454, October 1935.

[11] A. Borgida. Features of languages for the development of information systems at the conceptual level. *IEEE Software*, 2(1):63–72, January 1985.

[12] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989.

[13] U. S. Chakravarthy, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive datbases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–273. Morgan Kaufmann, Los Altos, California, 1987.

[14] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[15] S. Cluet, C. Delobel, C. Lécluse, and P. Richard. Reloop, an algebra based query language for an object-oriented database system. In *Proc. 1st Inter. Conf. in Deductive and Object-Oriented Databases*, pages 294–313, December 1989.

[16] Computer Corporation of America. *ADAPLEX: Rational and reference manual*, cca-83-08 edition, May 1983.

[17] U. Dayal. Queries and views in an object-oriented data model. In *Proc. 2nd International Workshop on Database Programming Languages*, pages 80–102, June 1989.

[18] S. K. Debray and D. S. Warren. Functional computation in logic programs. *ACM Transactions on*

Programming Languages and Systems, 11(3):451–481, July 1989.

[19] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, October 1980.

[20] G. Huet and D. Oppen. Equations and rewrite rules: A survey. In R. Book, editor, *Formal Language Perspectives and Open Problems.* Academic Press, 1980.

[21] M. Machtey and P. Young. *An Introduction to the General Theory of Algorithms.* North-Holland, 1978.

[22] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[23] D. Maier and J. Stein. Indexing in an object-oriented dbms. In *Proc. International Workshop on Object-Oriented Database Systems*, 1986.

[24] A. O. Mendelzon. Functional dependencies in logic programs. In *Proc. 11th International Conference on Very Large Data Bases*, pages 324–330, 1985.

[25] A. O. Mendelzon and P. T. Wood. Functional dependencies in horn clause queries. *ACM Transactions on Database Systems*, 16(1):31–55, March 1991.

[26] J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong. A language facility for designing database-intensive applications. *ACM Transactions on Database Systems*, 5(2):185–207, June 1980.

[27] C. G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, April 1980.

[28] C.G. Nelson and D. C. Oppen. Fast decision procedures based on union and find. In *Proc. 18th Annual Symposium on Foundations of Computer Science*, 1977.

[29] D. J. Rosenkrantz and H. B. Hunt III. Processing conjunctive predicates and queries. In *Proc. 6th Int. Conf. on Very Large Data Bases*, October 1980.

[30] S. T. Shenoy and Z. M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):344–361, September 1989.

[31] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1986.

[32] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, 1982.

[33] J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, 1989.

[34] M. F. van Bommel. Theory and applications of join conditions and path functional dependencies for object-oriented data models. Master's thesis, Department of Computer Science, University of Waterloo, 1989.

[35] M. F. van Bommel and G. E. Weddell. Reasoning about equations and functional dependencies on complex objects. Research Report CS-90-45, Department of Computer Science, University of Waterloo, 1990.

[36] G. E. Weddell. Selection of indices to memory-resident entities for semantic data models. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):274–284, June 1989.

[37] G. E. Weddell. A theory of functional dependencies for object-oriented data models. In *Proc. 1st International Conference on Deductive and Object-Oriented Databases*, pages 150–169, December 1989.

[38] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Transactions on Database Systems*, 17(1):32–64, March 1992.

[39] C. Zaniolo. The database language gem. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 207–218, May 1983.