UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT
COMPUTER SCIENCE DEPARTMENT

Reasoning About Functional Dependencies Generalized for Semantic Data Models

Grant E. Weddell

Research Report
CS-89-14

April, 1989

Reasoning About Functional Dependencies Generalized for Semantic Data Models

Grant E. Weddell

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

ABSTRACT

We propose a more general form of functional dependency for semantic data models that derives from their common feature in which the separate notions of *domain* and *relation* in the relational model are combined into a single notion of *class*. The feature manifests itself in a richer terminological component for their query languages in which a single term may traverse any number of properties (including none). We prove the richer expressiveness of this more general functional dependency, and exhibit a sound and complete set of inference axioms. Decision procedures are developed that apply when the dependencies included in a schema correspond to keys, or when the schema itself is acyclic. The theory is then extended to include a generalization of select-join queries. Of particular significance is that the queries become an additional source of functional dependency constraints. Finally, we outline several applications of the theory to various problems in physical design and in query optimization. The applications derive from an ability to predict when queries can have at most one solution.

The research reported here was supported in part by the Natural Sciences and Engineering Research Council of Canada, Bell-Northern Research Ltd, the University of Waterloo and the Multimedia Laboratory.

1. INTRODUCTION

There are several problems with the relational model when used for complex applications.[9] Some of these problems derive from its notion of a property, and its strict separation of objects that must have property values, called *tuples*, from objects that can be property values, called *domain values*. An important consequence is that query languages which are variations of a "typed" form of the tuple calculus, such as SQL [5] or QUEL [8], require all terms to denote objects that are domain values. This implies that users must introduce properties of objects to serve as their means of reference, and that all relationships between objects must be expressed indirectly in terms of these properties.

In order to overcome these difficulties, more recent *semantic data models* [1, 4, 6, 12, 13, 18] and *object-oriented languages* [7, 14] have usually combined the separate notions of domain and relation into a single notion of *class*, and thereby allowed properties to be defined between any pair of classes. Terms in query languages are then permitted to "traverse" any number of properties: none, in recognition that objects that were tuples have separate identity, or more than one, since objects that were domain values are now permitted to also have structure.

If we think of an *attribute* as what the terminological component of a query language permits one to ask of an object, then the variety of attributes possible with semantic data models and object-oriented databases corresponds to all possible property paths. Also, the possibility exists with most such data models that a schema may be defined in which the number of different attributes is countably infinite. This happens for cyclic schema. For example, if employee objects have a "boss" property which is also an employee object, then there is a "boss" attribute, a "boss of the boss" attribute, and so on.

In this paper, we propose a more general form of functional dependency constraint for semantic data models, called a *path functional dependency* (PFD), in which component attributes may correspond to property paths. In Section 2, we define a simple semantic data model together with a notion of interpretation based on directed graphs. Section 3 defines PFDs, proves their richer expressiveness, and presents a set of inference axioms. A proof of their soundness and completeness with respect to our graph theoretic notion of interpretation is also included. In Section 4, we derive some decision procedures that apply in cases where the PFDs included with a schema correspond to keys, or when the schema itself is acyclic. We also prove that a closure construction used by the procedures can require exponential time in the more general case of cyclic schema with non-key PFDs. In Section 5, we extend the theory by introducing a generalization of select-join queries. In the process, we unveil a new source of PFDs induced by the join conditions

of a query. A motivation for the theory is given in Section 6, where we present a number of applications of the theory to problems in physical design and query optimization. The applications derive from an ability to predict when at most one solution is possible to a query. Our summary comments are given in Section 7, where we also mention some open problems.

2. DEFINITIONS

In the relational model, a relation R with attributes $\{A_1, \cdots, A_n\}$ and corresponding domains $\{D_1, \cdots, D_n\}$ is usually declared by a *relational scheme* of the form

$$R\{A_1 : D_1, \cdots, A_n : D_n\} \tag{2.1}$$

There are two consequences of this structuring of a relation on the forms that terms may have for query languages based on the tuple calculus, such as SQL or QUEL. First, a variable v in a query is assumed to range over a particular relation. And second, a term is required to have the form "v.A" where A is an attribute of the relation over which v ranges. Only a small finite number of terms can therefore be expressed for a given variable.

In contrast, a semantic data model combines the separate notions of domain and relation into a single notion of *class*, and allows attributes, typically called *instance variables*, to then have any class as their range.† In terms of the relational model, this is analogous to permitting tuple-valued attributes. This idea is manifest in the following definition of a simple semantic data model.

Definitions. A database is defined by a *class schema* S consisting of a set of *class schemes* of the form

$$C\{P_1 : C_1, \cdots, P_n : C_n\} \tag{2.2}$$

Each attribute P_i, hereafter called a *property*, is unique in a given class scheme. Its type, written *Ran*(C, P_i), is the name C_i of another (not necessarily distinct) class scheme. The set of names of class schemes in S is written *Classes*(S), and the set of property names occurring in the definition of a particular class scheme with the name C is written *Props*(C).

□

† The distinction remains, however, for "built-in" *Integer* or *String* classes. They may serve as a range for a property, but may not themselves have properties.

Example. A database storing information about student course enrollment might be defined by the set of five class schemes listed in Table 1, where the last two are perhaps "built-in". Note that the value of either the S or C property for any enrollment object will be other objects that also have property values.
□

An important consequence is that query languages can now allow terms to have any number of "dots", including none. In other words, terms can navigate from object to object by following a property value path, possibly of zero length. For example, if a variable v ranges over all objects in the Enrollment class, then a query language might permit a term such as "$v.Student.Name$", or even just "v" to be expressed. Furthermore, if a class schema is cyclic, in the sense formally defined below, then the variety of terms expressible for a given variable can be countably infinite. For example, given the class scheme C { P : C } and variable v ranging over C, possible terms include "v", "v.P", "v.P.P", and so on.

Enrollment {S : Student, C : Course, Mark : Integer}
Student {Name : String}
Course {Room : Integer, Time : Integer}
Integer { }
String { }

Table 1. An Enrollment Class Schema

The concept of a *path function* will be our means of referring to a sequence of properties that may now occur in a term.

Definition. The set of *property induced path functions* $PF_{Prop}(S)$ over class schema S is the smallest set such that

1. $\{P \mid P \in Props(C), \text{ for some } C \in Classes(S)\} \subseteq PF_{Prop}(S)$, where

   $$Dom(P) \equiv \{C \in Classes(S) \mid P \in Props(C)\}$$

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

$Dom(pf.\mathrm{P}) \equiv Dom(pf)$

$Ran(\mathrm{C}, pf.\mathrm{P}) \equiv Ran(Ran(\mathrm{C}, pf), \mathrm{P})$

□

Example. The set of property induced path functions for the enrollment schema is

{S, S.Name, C, C.Room, C.Time, Mark, Name, Room, Time}

where, for example, $Dom(\mathrm{C.Room})$ is {Enrollment} and $Ran(\mathrm{Enrollment}, \mathrm{C.Room})$ is Integer. Note that the domain of all other path functions are also singleton sets, since no two class schemes have the same name for any of their properties.

□

Definition. The set of *identity path functions* $PF_{Id}(\mathrm{S})$ over class schema S consists of $\mathrm{Id_C}$ for each $\mathrm{C} \in Classes(\mathrm{S})$, where $Dom(\mathrm{Id_C}) \equiv \{\mathrm{C}\}$ and $Ran(\mathrm{Id_C}) \equiv \mathrm{C}$. (Identity path functions will be our means of referring to property value paths of zero length. Note that they are not considered properties.)

□

Definitions. The set of path functions over class schema S, written $PF(\mathrm{S})$, is defined as $PF_{Prop}(\mathrm{S}) \cup PF_{Id}(\mathrm{S})$ (i.e. the set of all property induced path functions and identity path functions over S). The capital letters X, Y and Z are used to denote subsets of $PF(\mathrm{S})$ for some class schema S. For each $\mathrm{C} \in Classes(\mathrm{S})$, we write $PathFuncs(\mathrm{C})$ to denote all $pf \in PF(\mathrm{S})$ where $\mathrm{C} \in Dom(pf)$.

□

Definition. The *length* of a path function is the number of property names that it contains; that is,

$$Len(pf) \equiv \begin{cases} 1 + Len(\hat{pf}) & ; \text{ if } pf \text{ has the form } \hat{pf}.\mathrm{P}, \\ 0 & ; \text{ if } pf \in PF_{Id}(\mathrm{S}), \\ 1 & ; \text{ otherwise.} \end{cases}$$

□

Example. For the enrollment schema, $Len(\mathrm{S.Name}) = 2$, and $Len(\mathrm{Id_{Student}}) = 0$.

□

The following lemma is a simple consequence of the fact that the number of difference properties mentioned in a given class schema is finite.

Lemma 1. *For any class schema* S *and integer* n, *the set* $\{ pf \in PF(\mathrm{S}) \mid Len(pf) \leq n \}$ *is finite.*

A cyclic schema can now be defined in terms of the path function concept.

Definition. A class schema S is cyclic *iff* there exists $pf \in PF_{Prop}(S)$ and $C \in Dom(pf)$ where $C = Ran(C, pf)$.
□

Lemma 2, also given without proof, is a straightforward consequence of Lemma 1 above.

Lemma 2. *A class schema* S *is acyclic iff* $PF(S)$ *is finite.*

Finally, we define an operator denoting the composition of two path functions that will help to simplify our notation in the remainder of the paper. Since composition is clearly associative, we shall omit explicit mention of precedence in expressions with more than instance of the operator.

Definition. Given class schema S and pf_1 , $pf_2 \in PF(S)$ where there exists $C \in Dom(pf_1)$ such that $Ran(C, pf_1) \in Dom(pf_2)$, the *composition* of pf_1 and pf_2, written $pf_1 \circ pf_2$, is defined as follows: 1) pf_2 if pf_1 is an identity path function, 2) pf_1 if pf_2 is an identity path function, or 3) $pf_1.pf_2$ otherwise.
□

Example. In the enrollment class schema, $S \circ Name$ is the path function S.Name, and both $Id_{Enrollment} \circ C$ and $C \circ Id_{Course}$ are the path function C. The expression $Id_{Enrollment} \circ C \circ Room$ denotes either $(Id_{Enrollment} \circ C) \circ Room$ or $Id_{Enrollment} \circ (C \circ Room)$ (by associativity of composition), and in both cases is the path function C.Room.
□

2.1. Directed Graphs as Interpretations of Class Schema

We adopt a form of labelled directed graph as our notion of interpretation for class schema.†

Definition. An *interpretation* for class schema S is a (possibly infinite) directed graph $G(V, A)$ with vertex and edge labelling as class and property names respectively. G must

† A more traditional notion based on some choice of relations over some domain of discourse is also possible. However, we have found it more natural and intuitive (what really matters when choosing a model theory) to think of entities or objects as vertices and of arcs as property values.

also satisfy the following constraints (note that the class name label of a vertex v is written $l_{Cl}(v)$).

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

□

Property value integrity ensures that property values satisfy the "typing" of properties indicated in a class schema. Property functionality ensures that properties are single-valued. Note that this constraint would not apply for nested relational models. Property value completeness ensures that properties are "total" on their domain classes. This constraint would not apply for data models that permit "null-inapplicable" values for properties.

Example. The directed graph of Figure 1 depicts one possible interpretation for the enrollment schema. The eight vertices of the graph correspond to eight different entities of which two are students, three are integers, and so on. Note that different *Integer* vertices represent different integers. The particular integers involved (or strings) are not important to our presentation, and therefore are not mentioned in our example interpretations.

□

The relationship between the path functions of a class schema and paths in an interpretation of the schema is established by the concept of *path description* and *vertex naming*, which we now define. When starting from a particular vertex, a subsequent lemma establishes that any given path to another (not necessarily distinct) vertex is described by a unique path function.

Definition. A path $u \xrightarrow{P} v \rightarrow \cdots \rightarrow w$ in an interpretation $G(V, A)$ for class schema S is *described* by a path function $pf \in PF(S)$ *iff* either: 1) the path consists of a single vertex u and pf is the identity path function $Id_{l_{Cl}(u)}$, or 2) pf is $P \circ \hat{pf}$, where $v \rightarrow \cdots \rightarrow w$ is described by $\hat{pf}$.

□

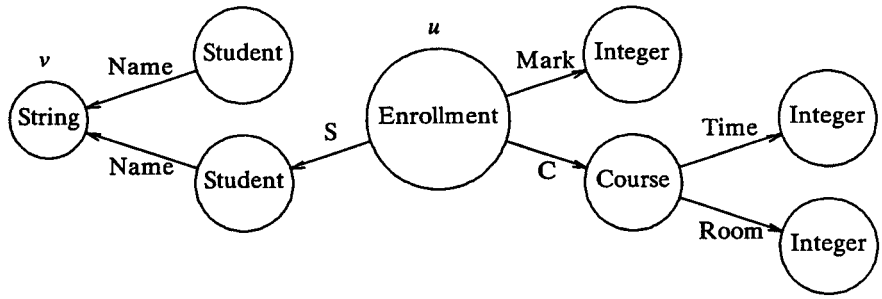


Figure 1. An Interpretation of the Enrollment Schema

Definition. A vertex $v \in V$ is *named* by vertex $u \in V$ and path function $pf \in PF(S)$ in an interpretation $G(V, A)$ of class schema S *iff* there exists a path $u \rightarrow \cdots \rightarrow v$ in G described by *pf*.

□

Lemma 3. *Given $u \in V$ and $pf \in PathFuncs(l_{Cl}(u))$ in an interpretation $G(V, A)$ of class schema* S, *there is a unique $v \in V$ named by u and pf, written $u.pf$.*

Proof. The lemma follows by a simple induction on the length of *pf*, together with the constraints on arcs and their labelling that G must satisfy (since it is an interpretation).

□

Example. In Figure 1, *u.S.Name* is vertex *v*. That is, vertex *v* is the unique vertex reachable from *u* via a path described by the path function S.Name.

□

3. PATH FUNCTIONAL DEPENDENCIES

Two important classes of constraints in the relational model are key constraints and functional dependencies. Given a relational scheme of the form (2.1) above, the latter are written $(A_{i_1} \cdots A_{i_m} \rightarrow A_{i_{m+1}} \cdots A_{i_n})$, and assert that no two tuples in R will have different A_{i_j} values, $m < j \leq n$, whenever they have the same values for A_{i_k}, $1 \leq k \leq m$. Key constraints are a kind of functional dependency that are usually expressed in a special way, such as with the form $Key(A_{i_1} \cdots A_{i_m})$, and assert that no two tuples in R will have different values for *any* attribute values whenever they have the same values for

A_{i_k}, $1 \leq k \leq m$. In the case of semantic data models, a more accurate statement is that each R-object has as unique combination of A_{i_k} attribute values, since there is usually no requirement that the combination of all attribute values must uniquely identify the object, that every class has at least one key.

We propose a more general form of functional dependency constraint for a class scheme in which any choice of path function is permitted as a component.

Definitions. A *path functional dependency* (PFD) over class schema S has the form

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

where $1 \leq m < n$, and where $pf_i \in PathFuncs(C)$, for $1 \leq i \leq n$. A *key path functional dependency* (key PFD) over class schema S is any path functional dependency over S with the form

$$C(pf_1 \cdots pf_m \rightarrow Id_C)$$

That is, a key PFD has a single identity path function occurring on its right-hand-side. A PFD is said to be *trivial* if its right-hand-side path functions are a subset of its left-hand-side path functions. The capital letter F is used to denote a finite set of PFDs over S.
□

Example. PFDs for the enrollment schema are listed in Table 2. All but the last are key PFDs. Informally, the first and second assert that students have a unique name and that no two courses can be given in the same room at the same time. The third asserts that a student can enroll at most once in a given course; and the last, which is not a consequence of the first three, is justified by virtue of a physical limitation — it asserts that a student cannot be enrolled in two separate courses at the same time.
□

Definition. A PFD $C(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$ over class schema S is *satisfied* by an interpretation $G(V, A)$ *iff* for any pair of vertices $u, v \in V$ where $l_{Cl}(u) = l_{Cl}(v) = C$, $u.pf_i = v.pf_i$, $1 \leq i \leq m$ implies $u.pf_j = v.pf_j$, $m < j \leq n$.
□

Example. The interpretation for the enrollment schema depicted in Figure 1 satisfies all PFDs in Table 2 except the first, since there are two students with the same name.
□

Student(Name $\rightarrow$ $Id_{Student}$)
Course(Room Time $\rightarrow$ Id_{Course})
Enrollment(S C $\rightarrow$ $Id_{Enrollment}$)
Enrollment(S C.Time $\rightarrow$ C)

Table 2. PFDs for the Enrollment Schema

Definition. Given PFDs $F \cup \{C(X \rightarrow Y)\}$ over class schema S, $C(X \rightarrow Y)$ is a *logical consequence* of F, written $F \models C(X \rightarrow Y)$, *iff* any interpretation $G(V, A)$ satisfying all PFDs in F must satisfy $C(X \rightarrow Y)$.
□

3.1. Expressiveness of Path Functional Dependencies

Traditional functional dependencies can all be expressed as PFDs in which no component path function has length exceeding one (in which no "dots" occur). This is also true for the key constraint languages for all semantic data models of which we are aware, such as ADAPLEX *uniqueness rules*, TAXIS *key properties* or GEM *key specifications*. In this section, we prove that permitting arbitrary path functions as components of PFDs increases the expressiveness of such constraints. Note that the last PFD on the enrollment schema given in Table 2 is at least one motivating example.

Theorem 1. *There exists PFDs over some class schema for which the set of possible satisfying interpretations is not the same as the set of possible satisfying interpretations for any combination of simpler PFDs, in which no component path function has length greater than one, over the same class schema. Thus, limiting PFDs to the form*

$$C(P_1 \cdots P_m \rightarrow P_{m+1} \cdots P_n)$$

reduces their expressiveness.

Proof. Consider the following class schema.

$$R\{A:S\} \qquad S\{B:T\} \qquad T\{\}$$

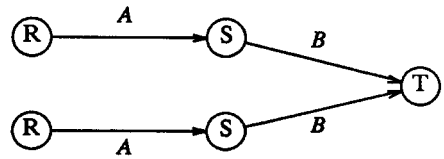
Clearly, only two non-trivial PFDs can be expressed in which no component path function has length exceeding one: $R(A \rightarrow Id_R)$ and $S(B \rightarrow Id_S)$. Note that both are key PFDs.

We prove the theorem by comparing the four possible sets of these two constraints to the following "non-simple" PFD.

$$R(A.B \rightarrow Id_R) \tag{3.1}$$

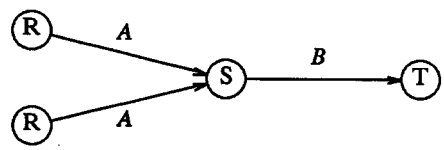
In each case, we exhibit an interpretation satisfying one side of the comparison only.

- Comparing (3.1) to the empty set or to the singleton set $\{R(A \rightarrow Id_R)\}$, the interpretation



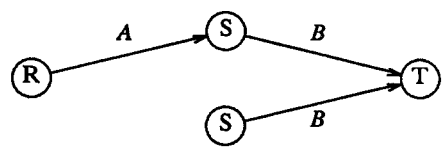
  satisfies either of the latter but not the former. In particular, there are two R-objects that have the same T-object as their A.B values, but different S-objects as their A values. Thus A values uniquely identify R-objects, but A.B values do not.

- Comparing (3.1) to the singleton set $\{S(B \rightarrow Id_S)\}$, the interpretation



  also satisfies the latter, but not the former. In this case, the single S-object is the A value of both R-objects. Again, A.B values do not uniquely identify R-objects, whereas this is trivially true of B values for S-objects.

- Finally, comparing (3.1) to the set $\{R(A \rightarrow Id_R), S(B \rightarrow Id_S)\}$, the interpretation



  now satisfies the former, but not both PFDs in the latter set. Here, B values fail to uniquely identify S-objects, whereas this is trivially true of both A and A.B values of R-objects.

Thus the set of possible interpretations consistent with (3.1) differs from the set consistent with any combination of the two possible non-trivial PFDs with zero length components. □

A simple corollary is that key PFDs are a richer form of key constraint than any in which no component path function has length exceeding one. This follows since all PFDs mentioned in the proof are key PFDs.

3.2. Inference Axioms for Path Functional Dependencies

In this section, we exhibit a sound and complete axiomatization for path functional dependencies. Five inference axioms are necessary, including three that are generalizations of a set of similar inference axioms complete for traditional functional dependencies. The remaining two are new, and are referred to as *simple attribution* and *simple prefix augmentation*. Simple attribution is a consequence of the functionality of properties, while simple prefix augmentation captures the intuitive notion that interpretations satisfying any PFD for a given class must satisfy similar PFDs for any other class for which the first occurs as the "type" of a subpart.

Definition. Given PFDs $F \cup \{C(X \rightarrow Y)\}$ over class schema S, $C(X \rightarrow Y)$ is *derivable* from F, written $F \vdash C(X \rightarrow Y)$, *iff* it is a member of F or is the result of one or more applications of the following inference axioms.

A1. (reflexivity) For every $C \in Classes(S)$ and $Y \subseteq X$ where X is a finite subset of $PathFuncs(C)$, derive $C(X \rightarrow Y)$.

A2. (path function augmentation) For every $C \in Classes(S)$ and finite subsets X, Y and Z of $PathFuncs(C)$, if $C(X \rightarrow Y)$ can be derived, then so can $C(XZ \rightarrow YZ)$ (where XZ, for example, denotes the union of all path functions in X and Z).

A3. (transitivity) For every $C \in Classes(S)$ and finite subsets X, Y and Z of $PathFuncs(C)$, if both $C(X \rightarrow Y)$ and $C(Y \rightarrow Z)$ can be derived, then so can $C(X \rightarrow Z)$.

A4. (simple attribution) For every $C \in Classes(S)$ and $P \in Props(C)$, derive $C(Id_C \rightarrow P)$.

A5. (simple prefix augmentation) For every $C_1 \in Classes(S)$ and $P \in Props(C_1)$, if

$$C_2(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$$

can be derived, where $C_2 = Ran(C_1, P)$, then so can

$$C_1(P\,o\,pf_1 \cdots P\,o\,pf_m \rightarrow P\,o\,pf_{m+1} \cdots P\,o\,pf_n)$$

□

Example. Table 3 lists a twelve step derivation of the PFD

$$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{Mark})$$

from those given in Table 2. The derived PFD asserts that at most one mark exists for a given combination of student name and course time.
□

	PFD	Justification
1.	$\text{Student}(\text{Name} \rightarrow \text{Id}_{\text{Student}})$	(given)
2.	$\text{Enrollment}(\text{S.Name} \rightarrow \text{S})$	(1 and *A5*)
3.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{S C.Time})$	(2 and *A2*)
4.	$\text{Enrollment}(\text{S C.Time} \rightarrow \text{C})$	(given)
5.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{C})$	(3, 4 and *A3*)
6.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{S S.Name C.Time})$	(3 and *A2*)
7.	$\text{Enrollment}(\text{S S.Name C.Time} \rightarrow \text{S C})$	(5 and *A2*)
8.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{S C})$	(6, 7 and *A3*)
9.	$\text{Enrollment}(\text{S C} \rightarrow \text{Id}_{\text{Enrollment}})$	(given)
10.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{Id}_{\text{Enrollment}})$	(8, 9 and *A3*)
11.	$\text{Enrollment}(\text{Id}_{\text{Enrollment}} \rightarrow \text{Mark})$	(*A4*)
12.	$\text{Enrollment}(\text{S.Name C.Time} \rightarrow \text{Mark})$	(10, 11 and *A3*)

Table 3. Derivation of a PFD

Theorem 2. *Inference axioms A1 to A5 are sound; that is,* $F \vdash C(X \rightarrow Y)$ *implies* $F \models C(X \rightarrow Y)$ *for any PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over a given class schema.*

Proof. Soundness for each axiom is a straightforward consequence of Lemma 3 and our definition of PFD satisfaction. We consider simple prefix augmentation as an example. If an interpretation $G(V, A)$ over a class schema S does not satisfy the PFD

$$C_1(\mathrm{P} \circ pf_1 \cdots \mathrm{P} \circ pf_m \rightarrow \mathrm{P} \circ pf_{m+1} \cdots \mathrm{P} \circ pf_n)$$

then there exists two vertices $u, v \in V$ where $u.\mathrm{P} \circ pf_i = v.\mathrm{P} \circ pf_i$ for all $1 \leq i \leq m$, but where $u.\mathrm{P} \circ pf_j \neq v.\mathrm{P} \circ pf_j$ for some $m < j \leq n$. The latter can only be true according to property functionality and Lemma 3 if there exists two distinct vertices $w = u.\mathrm{P}$ and $x = v.\mathrm{P}$, where $l_{Cl}(w) = l_{Cl}(x) = Type(\mathrm{C}_1, \mathrm{P})$. Thus, $w.pf_i = x.pf_i$ for all $1 \leq i \leq m$, but $w.pf_j \neq x.pf_j$ for some $m < j \leq n$, and therefore G cannot satisfy

$$\mathrm{C}_2(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$$

where $\mathrm{C}_2 = Type(\mathrm{C}_1, \mathrm{P})$.

□

It will simplify matters if we now introduce *additivity* and *projectivity* axioms, along with more general forms of attribution and prefix augmentation.

A6. (additivity) For every $\mathrm{C} \in Classes(\mathrm{S})$ and finite subsets X, Y and Z of $PathFuncs(\mathrm{C})$, if $\mathrm{C}(X \rightarrow Y)$ and $\mathrm{C}(X \rightarrow Z)$ can be derived, then so can $\mathrm{C}(X \rightarrow YZ)$.

A7. (projectivity) For every $\mathrm{C} \in Classes(\mathrm{S})$ and finite subsets X, Y and Z of $PathFuncs(\mathrm{C})$, if $\mathrm{C}(X \rightarrow YZ)$ can be derived, then so can $\mathrm{C}(X \rightarrow Y)$.

A8. (attribution) For every $\mathrm{C} \in Classes(\mathrm{S})$ and $pf \in PathFuncs(\mathrm{C})$, derive $\mathrm{C}(\mathrm{Id}_{\mathrm{C}} \rightarrow pf)$.

A9. (prefix augmentation) For every $\mathrm{C}_1 \in Classes(\mathrm{S})$ and $pf \in PathFuncs(\mathrm{C}_1)$, if

$$\mathrm{C}_2(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$$

can be derived, where $\mathrm{C}_2 = Ran(\mathrm{C}_1, pf)$, then so can

$$\mathrm{C}_1(pf \circ pf_1 \cdots pf \circ pf_m \rightarrow pf \circ pf_{m+1} \cdots pf \circ pf_n)$$

Lemma 4. *Inference axioms A6 to A9 are also sound.*

Proof. Derivations of the additivity and projectivity axioms from reflexivity, path function augmentation and transitivity are well-known.[10, 15] The more general forms of attribution and prefix augmentation can be derived by simple inductions on the length of *pf*. For the latter, assume there exists $\mathrm{C}_1 \in Classes(\mathrm{S})$ and $\mathrm{pf} \in PathFuncs(\mathrm{C}_1)$, and that the PFD

$$\mathrm{C}_2(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$$

can be derived, where $Ran(\mathrm{C}_1, pf) = \mathrm{C}_2$. If $Len(pf) = 0$, then pf is the identity path function $\mathrm{Id}_{\mathrm{C}_1}$. The conclusion follows since $\mathrm{C}_2 = Ran(\mathrm{C}_1, \mathrm{Id}_{\mathrm{C}_1}) = \mathrm{C}_1$ and $\mathrm{Id}_{\mathrm{C}_1} \circ pf_i = pf_i$. Now assume pf has the form $\mathrm{P} \circ \hat{pf}$. By the inductive assumption, we can derive

$$Ran(C_1, P)(\hat{pf} \circ pf_1 \cdots \hat{pf} \circ pf_m \rightarrow \hat{pf} \circ pf_{m+1} \cdots \hat{pf} \circ pf_n)$$

and by simple prefix augmentation

$$C_1(P \circ \hat{pf} \circ pf_1 \cdots P \circ \hat{pf} \circ pf_m \rightarrow P \circ \hat{pf} \circ pf_{m+1} \cdots P \circ \hat{pf} \circ pf_n)$$

The conclusion follows since $pf = P \circ \hat{pf}$.

For the more general form of attribution, consider a $C \in Classes(S)$ and $pf \in PathFuncs(C)$. Again, if $Len(pf) = 0$, then pf is the identity path function Id_C, in which case $C(Id_C \rightarrow Id_C)$ can be derived by reflexivity. Now assume pf has the form $\hat{pf} \circ P$. By the inductive assumption, we can derive $C(Id_C \rightarrow \hat{pf})$. Simple attribution justifies the derivation of

$$Ran(C, \hat{pf})(Id_{Ran(C, \hat{pf})} \rightarrow P)$$

Use of prefix augmentation gives $C(\hat{pf} \rightarrow \hat{pf} \circ P)$, and then $C(Id_C \rightarrow pf)$ follows by transitivity.

□

To prove completeness of our axiomatization, we shall introduce the concepts of a C-*Tree* and *Two*-C-*Tree*, which are interpretations for a given class schema S induced by a particular choice of class scheme $C \in Classes(S)$. In preparation, we extend the notions of the closure F^+ of a given set of functional dependencies F, and of the closure X^+ of a set of attributes with respect to a set of functional dependencies.

Definition. The *closure* of a set of PFDs F over class schema S, written F^+, is the smallest set containing all PFDs $C(X \rightarrow Y)$ over S where $F \vdash C(X \rightarrow Y)$.

□

Definition. Given a set F of PFDs over class schema S, the *closure* of a finite set of path functions $X \subseteq PathFuncs(C)$ for some $C \in Classes(S)$, written X^+, is the smallest set containing all $pf \in PathFuncs(C)$ where $F \vdash C(X \rightarrow pf)$. (Note that X^+ may not be finite.)

□

Lemma 5. *Given PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over class schema S,* $F \vdash C(X \rightarrow Y)$ *iff* $Y \subseteq X^+$.

Proof. (if part) Additivity implies $F \vdash C(X \rightarrow Y)$ since $F \vdash C(X \rightarrow pf)$ for all $pf \in Y$.

(only if part) Projectivity implies $F \vdash C(X \rightarrow pf)$ for all $pf \in Y$, and therefore that $Y \subseteq X^+$.

□

F^+ for any choice of PFDs F over a class schema S will be countably infinite whenever S is cyclic. For example, let S consist of the following two class schemes which describe a database about employees and their names and bosses.

$$\text{E}\ \{\ \text{N} : \text{String}\ ,\ \text{B} : \text{E}\ \} \qquad \text{String}\ \{\ \} \tag{3.2}$$

Note that S is cyclic because of the "boss" property B. The attribution axiom implies that F^+ will contain all PFDs of the form $\text{E}(\text{Id}_\text{E} \rightarrow \text{B}^i)$, where B^i represents B occurring i times. If F is $\{\text{E}(\text{B.N} \rightarrow \text{B})\}$ (i.e. a single path functional dependency asserting that bosses have unique names), then prefix augmentation implies that F^+ also contains all PFDs of the form $\text{E}(\text{B}^i.\text{N} \rightarrow \text{B}^i)$, for $i > 0$. The closure of a set of path functions can also be countably infinite for cyclic schema. For example, if S is as above, then X^+ is $PF(\text{S})$ if X consists of both identity path functions Id_E and $\text{Id}_{\text{Integer}}$, regardless of the selection of PFDs F.

Definition. Given class schema S and class scheme $\text{C} \in \mathit{Classes}(\text{S})$, a C-*Tree* is a (possibly infinite) directed graph $G(V, A)$ with vertices and arcs constructed as follows.

Step 1. For each $pf \in \mathit{PathFuncs}(\text{C})$, add vertex v with $l_{Cl}(v)$ assigned $\mathit{Ran}(\text{C}, pf)$, and with an additional label $l_{Pf}(v)$ (called its *path function labelling*) assigned pf. The single vertex u with $l_{Pf}(u) = \text{Id}_\text{C}$ is denoted as $\mathit{Root}(G)$.

Step 2. For each $u, v \in V$ where $l_{Pf}(u) = pf$ and $l_{Pf}(v) = pf \circ \text{P}$, add $u \xrightarrow{\text{P}} v$ to A.

□

Example. Figure 2(a) illustrates the top levels of the E-Tree for the employee class schema (3.2) above. The vertex denoted as $\mathit{Root}(G)$ is also indicated. The sequence of property names on the path from $\mathit{Root}(G)$ to any other vertex v is its path function labelling $l_{Pf}(v)$ (proven below).

□

Lemma 6. *The* C-*Tree defined by class scheme* $\text{C} \in \mathit{Classes}(\text{S})$ *for a given class schema* S *is an interpretation of* S *that satisfies the following two conditions.*

1. For all $u, v \in V$ and $pf \in \mathit{PathFuncs}(u)$, if $u.pf = v$ then $l_{Pf}(u) \circ pf = l_{Pf}(v)$.
2. For all $u \in V$, $u = \mathit{Root}(G).l_{Pf}(u)$.

Proof. We first prove that G is an interpretation of S. Consider property value integrity. If $u \xrightarrow{\text{P}} v \in A$ where $l_{Pf}(u) = pf$, then $\mathit{Ran}(\text{C}, pf) = l_{Cl}(u)$ according to Step 1, and therefore $l_{Cl}(u) \in \mathit{Dom}(\text{P})$ by definition of composition of path functions. By

definition of path functions

$$Ran(C, pf \circ P) = Ran(Ran(C, pf), P) = Ran(l_{Cl}(u), P)$$

From Step 2, $u \xrightarrow{P} v \in A$ also implies $l_{Pf}(v) = pf \circ P$, and therefore from Step 1 $Ran(C, pf \circ P) = l_{Cl}(v)$. Thus, $l_{Cl}(v) = Ran(l_{Cl}(u), P)$. Now consider property functionality. According to Step 2, if $u \xrightarrow{P} v$, $u \xrightarrow{P} w \in V$, then $l_{Pf}(v) = l_{Pf}(w)$. But then $v = w$ since Step 1 of the construction implies no two vertices have the same path function labelling. To establish property value completeness, let $l_{Pf}(u) = pf$ for some $u \in V$. From Step 1, $P \in Props(l_{Cl}(u))$ implies $P \in Props(Ran(C, pf))$. By definition of path functions, $pf \circ P \in PathFuncs(C)$, and therefore $u \xrightarrow{P} v \in A$ for some $v \in V$.

Both conditions can be established by a simple induction on the length of path function labelling for vertex u. For the first, if $Len(pf) = 0$, then pf is $Id_{l_{Cl}(u)}$, and therefore $v = u$. The consequent follows since, by definition of composition, $l_{Pf}(u) \circ Id_{Cl}(u) = l_{Pf}(u)$. Now let $pf = \hat{pf} \circ P$. By the inductive assumption $u.\hat{pf} = w$ implies $l_{Pf}(u) \circ \hat{pf} = l_{Pf}(w)$. If $u.\hat{pf} \circ P = v$ then $w \xrightarrow{P} v$ must have been added in Step 2, and therefore $l_{Pf}(v) = l_{Pf}(u) \circ \hat{pf} \circ P$. Now consider the second condition. If $Len(l_{Pf}(u)) = 0$, then $l_{Pf}(u)$ is Id_C, and therefore

$$u = Root(G) = Root(G).Id_{l_{Pf}(Root(G))}$$

Let $l_{Pf}(u) = pf \circ P$. By the inductive assumption, the $v \in V$ where $l_{Pf}(v) = pf$ satisfies $v = Root(G).pf$. Since Step 2 adds $v \xrightarrow{P} u$ to A, $u = Root(G).pf \circ P$, and therefore satisfies $u = Root(G).l_{Pf}(u)$.

□

Definition. Given PFDs F over class schema S and PFD $C(X \rightarrow Y) \notin F^+$, a *Two-C-Tree* is a (possibly infinite) directed graph $G(V, A)$ constructed as follows.

Step 1. Construct two C-Trees $G_1(V_1, A_1)$ and $G_2(V_2, A_2)$. Let $R1$ and $R2$ denote $Root(G_1)$ and $Root(G_2)$ respectively.

Step 2. Remove any $v \in V_2$ and its incident arcs from A_2 whenever $l_{Pf}(v) \in X^+$. (Note that $R2$ is not removed, since removal together with attribution, additivity and transitivity would imply $C(X \rightarrow Y) \in F^+$, contrary to assumptions.) Add all vertices in $V_1 \cup V_2$ to V and all arcs in $A_1 \cup A_2$ to A.

Step 3. For each $u \in V$ and $P \in Props(l_{Cl}(u))$ where $u \xrightarrow{P} w \notin A$ for all $w \in V$, add arc $u \xrightarrow{P} R1.l_{Pf}(u) \circ P$ to A.

□

Example. Figure 2(b) illustrates the top levels of the Two-E-Tree for the singleton set $F = \{E(B.N \rightarrow B)\}$ over the employee class schema (3.2) above, and the PFD $E(B.N \rightarrow Id_E)$ not in F^+. Note that vertices in V_2 (i.e. originating from the second of the two E-Trees used in the construction) are represented as small squares. The two vertices denoted as *R1* and *R2* are also indicated. In this case, the sequence of property names on the path from *R1* or *R2* to any other vertex is its path function labelling.

□

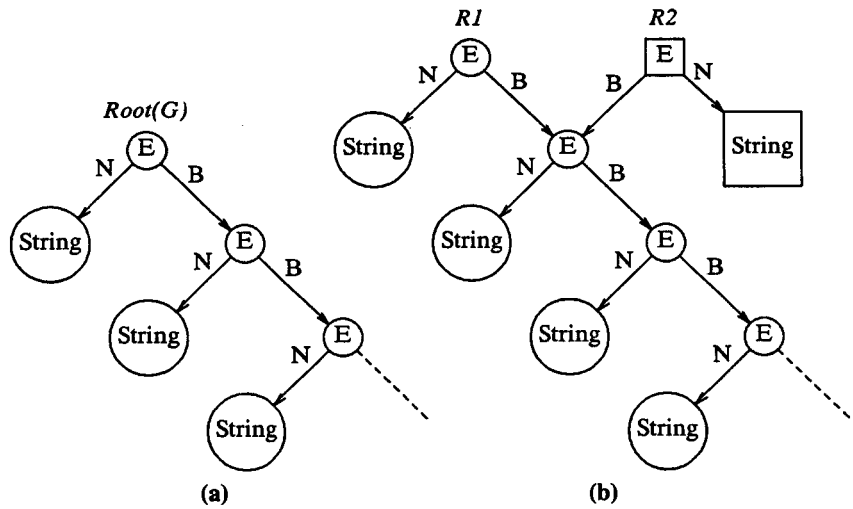


Figure 2. E-Tree and Two-E-Tree for the Employee Schema.

Lemma 7. *The Two-C-Tree* $G(V, A)$ *defined by PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over class schema* S, *where* $C(X \rightarrow Y) \notin F^+$, *is an interpretation of* S *satisfying the following two conditions.*

1. For all $u, v \in V$ and $pf \in PathFuncs(u)$, if $u.pf = v$ then $l_{Pf}(u) \circ pf = l_{Pf}(v)$.
2. One and only one of the following holds for each $u \in V$
   - $u = R1.l_{Pf}(u)$, $u \neq R2.l_{Pf}(u)$ and $l_{Pf}(u) \notin X^+$.
   - $u \neq R1.l_{Pf}(u)$, $u = R2.l_{Pf}(u)$ and $l_{Pf}(u) \notin X^+$.

- $u = R1.l_{Pf}(u)$, $u = R2.l_{Pf}(u)$ and $l_{Pf}(u) \in X^+$.

Proof. We first prove that G is an interpretation. From Lemma 6, G can only violate property value completeness at the end of Step 2 since its only effect is to remove vertices from an interpretation G_2. Any violation of property value completeness is then resolved in Step 3 by adding arcs to A, clearly without any possibility of violating property functionality. What remains is to demonstrate that Step 3 does not compromise property value integrity. If $u \xrightarrow{P} v$ is added in Step 3, then $l_{Cl}(u) \in Dom(P)$ and $R1.l_{Pf}(u) \circ P = v$. By Lemma 6, the latter implies $l_{Pf}(v) = l_{Pf}(u) \circ P$, and therefore by definition of path function labelling

$$l_{Cl}(v) = Ran(C, l_{Pf}(u) \circ P) = Ran(Ran(C, l_{Pf}(u)),P) = Ran(l_{Cl}(u), P)$$

As in Lemma 6, both conditions can be established by induction on the length of path function labelling. Since we know from above that $l_{Pf}(v) = l_{Pf}(u) \circ P$ is true for any arc $u \xrightarrow{P} v$ added in Step 3, the proof of the first condition for Lemma 6 applies essentially unchanged. To prove the second condition, we first prove that if $u \in V_2$ is removed in Step 2, then all vertices $v \in V_2$ reachable from u are also removed. To begin, v is reachable implies there exists $pf \in PathFuncs(l_{Cl}(u))$ such that $u.pf = v$. By Lemma 6, $l_{Pf}(u) \circ pf = l_{Pf}(v)$. Since u is removed, we know $F \vdash C(X \rightarrow l_{Pf}(u))$. By attribution $F \vdash l_{Cl}(u)(Id_{l_{Cl}(u)} \rightarrow pf)$, and by prefix augmentation $F \vdash C(l_{Pf}(u) \rightarrow l_{Pf}(u) \circ pf)$ or $F \vdash C(l_{Pf}(u) \rightarrow l_{Pf}(v))$. Thus $F \vdash C(X \rightarrow l_{Pf}(v))$ by transitivity, and therefore v must also have been removed.

Consequently, according to the second condition of Lemma 6, for all $u \in V$, we know

$$u = R1.l_{Pf}(u) \quad \text{or} \quad u = R2.l_{Pf}(u)$$

The second condition then follows if

$$R1.l_{Pf}(u) = R2.l_{Pf_u}(u) \quad \textit{iff} \quad l_{Pf}(u) \in X^+$$

holds for all $u \in V$, which we now prove by induction on $Len(l_{Pf}(u))$. If $Len(l_{Pf}(u)) = 0$, then $l_{Pf}(u) = Id_C$. Thus $l_{Pf}(u) \notin X^+$, and u must be either $R1$ or $R2$, but not both. Now let $l_{Pf}(u) = pf \circ P$. There are two cases to consider. First, if $pf \in X^+$, then $pf \circ P \in X^+$ by simple attribution, prefix augmentation and transitivity. According to the inductive assumption and property functionality, this then implies $R1.pf \circ P = R2.pf \circ P$. Second, if $pf \notin X^+$, then by the inductive assumption $R1.pf \neq R2.pf$. Thus, $R1.pf \circ P = R2.pf \circ P$ *iff* Step 3 adds arc $R2.pf \xrightarrow{P} R1.pf \circ P$ to A. But this happens *iff* Step 2 has removed a vertex $w \in V_2$ where $l_{Pf}(w) = pf \circ P$, which in turn can happen *iff* $pf \circ P \in X^+$.

□

Corollary 1. *Given a Two-C-Tree* $G(V, A)$ *defined by PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over class schema* S, *where* $C(X \rightarrow Y) \notin F^+$, *for any distinct* $u, v \in V$ *where* $l_{CI}(u) = l_{CI}(v)$, *if* $u.pf = v.pf$ *for some* $pf \in PathFuncs(l_{CI}(u))$, *then* $l_{Pf}(u) = l_{Pf}(v)$ *and* $l_{Pf}(u) \circ pf \in X^+$.

Proof. By virtue of condition one of Lemma 7, if $u.pf = v.pf$ then $l_{Pf}(u) \circ pf = l_{Pf}(v) \circ pf$, and therefore $l_{Pf}(u) = l_{Pf}(v)$. Without loss of generality, condition two of Lemma 7 first implies $u = R1.l_{Pf}(u)$ and $v = R2.l_{Pf}(v)$. Thus $R1.l_{Pf}(u) \circ pf = R2.l_{Pf}(v) \circ pf$, and condition two of Lemma 7 then implies $l_{Pf}(u) \circ pf \in X^+$.

□

Theorem 3. *Inference axioms A1 to A5 are complete; that is,* $F \models C(X \rightarrow Y)$ *implies* $F \vdash C(X \rightarrow Y)$ *for any PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over a given class schema.*

Proof. Given PFDs F over class schema S, for any $C(X \rightarrow Y)$ over S not in F^+, we prove that the Two-C-Tree interpretation $G(V, A)$ satisfies all PFDs in F^+, but not $C(X \rightarrow Y)$.

First, we show G does not satisfy $C(X \rightarrow Y)$. From Lemma 7, $R1.pf = R2.pf$ for all $pf \in X$. If G did satisfy $C(X \rightarrow Y)$, then $R1.pf = R2.pf$ for all $pf \in Y$, which implies $C(X \rightarrow pf) \in F^+$. But use of the additivity axiom then implies $C(X \rightarrow Y) \in F^+$, a contradiction.

We now show that G does satisfy all PFDs in F^+. Assume

$$\hat{C}(pf_1 \cdots pf_m \rightarrow pf_{m+1} \cdots pf_n)$$

is in F^+, and that there exists distinct $u, v \in V$ where $u.pf_i = v.pf_i$ for all $1 \leq i \leq m$, but where $u.pf_j \neq v.pf_j$ for some $m < j \leq n$. By Corollary 1, $l_{Pf}(u) = l_{Pf}(v)$ since u and v agree on some path function, and therefore by prefix augmentation and projectivity, there is some PFD

$$C(l_{Pf}(u) \circ pf_1 \cdots l_{Pf}(u) \circ pf_m \rightarrow l_{Pf}(u) \circ pf_j)$$

also in F^+, where $R1.l_{Pf}(u) \circ pf_i = R2.l_{Pf}(u) \circ pf_i$ for all $1 \leq i \leq m$, but where $R1.l_{Pf}(u) \circ pf_j \neq R2.l_{Pf}(u) \circ pf_j$ for some $m < j \leq n$. Corollary 1 also establishes $l_{Pf}(u) \circ pf_i \in X^+$ for all $1 \leq i \leq m$. This is contradictory since, by m–1 applications of additivity and by transitivity, $l_{Pf}(u) \circ pf_j \in X^+$, and therefore by Lemma 7, $R1.l_{Pf}(u) \circ pf_j = R2.l_{Pf}(u) \circ pf_j$.

□

4. DECISION PROCEDURES

The decision procedures for PFDs that we derive can be used when all antecedents are key PFDs, or when the schema is acyclic. To begin, we first define two subsets of F^+ that are induced by a particular choice of class scheme. These subsets will prove to be rich enough to permit the derivation of any PFD in F^+, on the same class scheme, with the use of the first three of our inference axioms alone; that is, to derive the PFD using reflexivity, path function augmentation and transitivity.

Definition. Given class schema S and C $\in$ *Classes*(S), the set of all property induced PFDs over S with respect to C, written F_1, consists of all PFDs over S of the form $C(pf \rightarrow pf \circ P)$. For $i \geq 0$, $F_1(i)$ denotes all PFDs $C(pf \rightarrow pf \circ P) \in F_1$ where $Len(pf \circ P) \leq i$.
□

Definition. Given PFDs F over class schema S and C $\in$ *Classes*(S), the set of all F-induced PFDs over S with respect to C, written F_2, consists of all PFDs over S defined on C that can be derived from F by a single use of prefix augmentation (axiom *A9*). For $i \geq 0$, $F_2(i)$ denotes all PFDs $C(X \rightarrow Y) \in F_2$ where $Len(pf) \leq i$, for all $pf \in XY$.
□

Theorem 4. *Given PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over class schema S,* $F \vdash C(X \rightarrow Y)$ *iff* $C(X \rightarrow Y)$ *can be derived from* $F_1 \cup F_2$ *by reflexivity, path function augmentation and transitivity.*

Proof. (if part) It suffices to show that all PFDs in $F_1 \cup F_2$ can be derived from F by using any of the inference axioms. This is clearly true for all PFDs in F_2. The PFDs in F_1 are of one of two forms: $C(Id_C \rightarrow P)$, or $C(pf \rightarrow pf.P)$. Any of the first form derive immediately from simple attribution. Also by simple attribution, we can derive $Ran(C, pf)(Id_{Ran(C, pf)} \rightarrow pf.P)$, and therefore any of the second form by prefix augmentation.

(only if part) Let $\hat{F}_1 = \bigcup_{C \in S} F_1$ and $\hat{F}_2 = \bigcup_{C \in S} F_2$. We begin by showing how any derivation of $C(X \rightarrow Y)$ from F can be modified to a derivation from $\hat{F}_1 \cup \hat{F}_2$ by reflexivity, path function augmentation and transitivity alone.

First observe that any use of simple attribution to derive a PFD of the form $\hat{C}(Id_{\hat{C}} \rightarrow P)$ is unnecessary since the PFD is already in $\hat{F}_1$. Now let pfd_1 and pfd_2 denote two PFDs in the derivation where the second is justified by simple prefix augmentation on the first. If pfd_1 is in F then pfd_2 is in $\hat{F}_2$, and this use of simple prefix augmentation can

be removed. We refer to this case as an *initial* use of prefix augmentation. Otherwise, we can replace this use of simple prefix augmentation by at most two new uses applied to PFDs preceding pfd_1 in the derivation as follows. If pfd_1 was derived by reflexivity, then pfd_2 can also be derived by reflexivity. If pfd_1 was derived by path function augmentation or by transitivity from one or two previous PFDs, then pfd_2 can also be derived by path function augmentation or by transitivity from one or two *new* PFDs added to the derivation, which are themselves the consequence of simple prefix augmentation applied to the one or two previous PFDs from which pfd_1 was derived. For example, without loss of generality, the necessary modification for the case of simple path function augmentation is illustrated as follows (where "$\mathrm{P} \circ XZ$", for example, denotes the set consisting of the composition of P with each path function in XZ).

	PFD	**Justification**
	. . .	
n.	$\mathrm{C}_1(X \rightarrow Y)$	?
n+1.	$\mathrm{C}_1(XZ \rightarrow YZ)$	(n and *A2*)
n+2.	$\mathrm{C}_2(\mathrm{P} \circ XZ \rightarrow \mathrm{P} \circ YZ)$	(n+1 and *A5*)
	. . .	

(before)

	PFD	**Justification**
	. . .	
n.	$\mathrm{C}_1(X \rightarrow Y)$	?
n+1.	$\mathrm{C}_1(\mathrm{P} \circ X \rightarrow \mathrm{P} \circ Y)$	(n and *A5*)
n+2.	$\mathrm{C}_1(XZ \rightarrow YZ)$	(n and *A2*)
n+3.	$\mathrm{C}_2(\mathrm{P} \circ XZ \rightarrow \mathrm{P} \circ YZ)$	(n+1 and *A2*)
	. . .	

(after)

Finally, if pfd_1 was itself derived from an earlier PFD by simple prefix augmentation, then pfd_2 can also be derived from the earlier PFD by using the more general prefix augmentation inference axiom with a larger prefix. Thus, by a simple induction, any use of simple prefix augmentation can be replaced by *initial* uses of prefix augmentation (as described above), and can therefore be removed.

The derivation of $\mathrm{C}(X \rightarrow Y)$ that results now consists of a sequence of PFDs justified by virtue of membership in $\hat{F}_1 \cup \hat{F}_2$, or by virtue of reflexivity, path function augmentation or transitivity applied to PFDs earlier in the derivation. Since these inference axioms only permit the derivation of PFDs *on the same class*, we can further modify the derivation by removing any PFD not defined on C, since they cannot be of any help in deriving $\mathrm{C}(X \rightarrow Y)$. The remaining PFDs mentioned in the derivation that were originally justified by virtue of membership in $\hat{F}_1 \cup \hat{F}_2$ are now justified by virtue of membership in $F_1 \cup F_2$ since this final modification ensures that they are defined on C.

□

A *Closure* construction will form the basis of our decision procedures. In contrast to similar constructions used to decide functional dependencies, this first version of *Closure* is only effective for acyclic schema since cyclic schema can cause argument sets to be infinite.

Definition. Given PFDs F over class schema S and $X \subseteq PathFuncs(C)$ for some $C \in S$, $Closure(X, PathFuncs(C), F_1 \cup F_2) \equiv \bigcup_{i \geq 0} X^i$, where each X^i is determined in sequence as follows.

Step 1. $i \leftarrow 0$; $X^0 \leftarrow X$.

Step 2. $X^{i+1} \leftarrow X^i \cup \{ pf \in PathFuncs(C) \mid$ there exists $C(Y \rightarrow Z) \in F_1 \cup F_2$ s.t. $Y \subseteq X^i$, $pf \notin X^i$ and $pf \in Z \}$

Step 3. $i \leftarrow i + 1$; and repeat from Step 2.

□

Theorem 5. *Given PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over class schema S,* $F \vdash C(X \rightarrow Y)$ *iff* $Y \subseteq Closure(X, PathFuncs(C), F_1 \cup F_2)$.

Proof. A proof that $Closure(X, PathFuncs(C), F_1 \cup F_2)$ is X^+ easily follows from the correctness of ALGORITHM 1 in [2]. The theorem follows by Lemma 5.
□

Example. Table 4 exhibits the operation of *Closure* for the PFD

Enrollment(S.Name C.Time → Mark)

over the enrollment schema. This is the same PFD used to demonstrate PFD derivation in Table 3. By Theorem 5, the PFD can be derived from those in Table 2 since Mark $\in X^4$. Note that only X^i for $i \leq 4$ are indicated, since clearly $X^j = X^4$ for $j > 4$.
□

It is straightforward to improve the effectiveness of *Closure* by modifying Step 2 to a form that requires examination of finite sets only. First, we extend the definition of *PathFuncs* in order to refer to finite subsets of path functions for cyclic schema.

Definition. Given class schema S, for each $C \in Classes(S)$ and $i \geq 0$, $PathFuncs(C, i)$ consists of all $pf \in PathFuncs(C)$ where $Len(pf) \leq i$.
□

F_1 = { Enrollment($Id_{Enrollment} \rightarrow$ S), Enrollment(S $\rightarrow$ S.Name),
Enrollment($Id_{Enrollment} \rightarrow$ C), Enrollment(C $\rightarrow$ C.Room),
Enrollment(C $\rightarrow$ C.Time), Enrollment($Id_{Enrollment} \rightarrow$ Mark) }

F_2 = { Enrollment(S.Name $\rightarrow$ S), Enrollment(C.Room C.Time $\rightarrow$ C),
Enrollment(S C $\rightarrow$ $Id_{Enrollment}$), Enrollment(S C.Time $\rightarrow$ C) }

X^0 = {S.Name, C.Time}
X^1 = {S.Name, C.Time, S}
X^2 = {S.Name, C.Time, S, C}
X^3 = {S.Name, C.Time, S, C, C.Room, $Id_{Enrollment}$}
X^4 = {S.Name, C.Time, S, C, C.Room, $Id_{Enrollment}$, Mark}

Table 4. Example of *Closure* for the Enrollment Schema.

According to Lemma 1, *PathFuncs*(C, i) must always denote a finite set, which is clearly also true for $F_1(i)$ and $F_2(i)$ defined above. Procedures to enumerate any finite sets denoted by *PathFuncs*(C, i), $F_1(i)$ or $F_2(i)$ for any integer $i \geq 0$ are straightforward.

A minimum constant α can be determined by a simple inspection of any finite set of PFDs F such that, for any PFD C($X \rightarrow Y$) $\in F$, α satisfies the condition

$$\left(\max_{pf \in XY} Len(pf) - \min_{pf \in XY} Len(pf) \right) \leq \alpha$$

That is, α bounds the difference in length between the longest and shortest component path functions of any PFD in F. Note that $\alpha = 1$ will suffice to bound PFDs in F_1 for any class schema S. Now consider the set F_2 of F-induced PFDs with respect to some class C $\in$ *Classes*(S). Since prefix augmentation does not derive PFDs with a larger length differential, this same bound will also hold for F_2.

These observations guarantee that X^i is always finite in the above *Closure* construction, and allow us to improve its effectiveness by limiting the number of path functions and PFDs considered in each invocation of Step 2. The new version of Step 2 is as follows.

Step 2. $X^{i+1} \leftarrow X^i \cup \{\, pf \in \mathit{PathFuncs}(C, j) \mid$ there exists $C(Y \rightarrow Z) \in F_1(j) \cup F_2(j)$ s.t. $Y \subseteq X^i$, $pf \in Z$, $pf \notin X^i$ and $j = \max(1, \alpha) + \max_{pf \in X^i} \mathit{Len}(pf) \,\}$

A simple variation of this more effective definition of *Closure*, in which we check for containment of another finite set of path functions at each iteration, also establishes that the set of PFDs which are logical consequences of a given set of finite PFDs is recursively enumerable. However, this dependency verifier may still require examining a number of path functions and PFDs exponential in the length of encoding of a class schema and given set of PFDs.

Theorem 6. *There exists class schema* S *and PFDs* $F \cup \{C(X \rightarrow Y)\}$ *over* S, *where* $F \models C(X \rightarrow Y)$, *for which Closure requires a number of iterations i exponential in the length of encoding of* S *and* F *before* $Y \subseteq X^i$.

Proof. Let $p_1, \cdots, p_n$ be the first n primes, and let S and F consist of the following class schema and PFDs (where P^m denotes a path function with property P occurring m times).

$$R\ \{\, A_1 : \mathit{Integer}, \cdots, A_n : \mathit{Integer}, B : R \,\}$$
$$\text{Integer}\ \{\,\}$$

$$R(A_1 \rightarrow B^{p_1}.A_1)$$
$$\cdots$$
$$R(A_n \rightarrow B^{p_n}.A_n)$$
$$R(B.A_1 \cdots B.A_n \rightarrow Id_R)$$

Now consider the use of *Closure* to verify that the PFD

$$R(B^{p_1}.A_1 \cdots B^{p_n}.A_n \rightarrow Id_R)$$

is a logical consequence of F. *Closure* will require $O(p_1 \times \cdots \times p_n)$ iterations before obtaining an X^i with a combination of path functions that permit a first use of a PFD in F_2 derived from the last PFD of those above. About the same number of iterations using similarly derived PFDs in F^+ are then required to add Id_R. The result follows since the sum of the first n primes is bound by a polynomial in n.
□

If we remove the last key PFD in the above, then the example used by the proof also establishes the existence of problems for which no finite subset $F_2(i)$ of F_2 can be used by *Closure* to produce the same set of path functions. For example, removing

$R(B^{l \cdot p_1}.A_1 \rightarrow B^{(l+1) \cdot p_1}.A_1)$ from F_2, for $l \geq 1$, removes all path functions $B^{m \cdot p_1}.A_1$, where $m > l$, from the value of *Closure*. In this case, each non-key PFD in F satisfies the condition that its right-hand-side path function exceeds its left-hand-side path function in length, but this is not a necessary condition. Consider where S and F consist of the following class schemes and (single) PFD.

$$R\{A : S, B : R\} \qquad S\{C : T\} \qquad T\{D : \text{Integer}\}$$
$$R(A.C.D \rightarrow B.A)$$

All PFDs in F_2 must then have the form

$$R(B^i.A.C.D \rightarrow B^{i+1}.A)$$

where $i \geq 0$, and therefore satisfy the opposite condition that their left-hand-side path function exceeds their right-hand-side path functions in length. Again, removing any PFDs in F_2 has the effect of "breaking a chain", and therefore removing path functions from the result of *Closure*, which otherwise consists of all path functions ending with the property A, C or D. It is also possible to devise a refinement to the example used in the proof of Theorem 6, based on the idea used here, to demonstrate a case requiring exponential time in the length of encoding of S and F where all PFDs in F do satisfy this opposite condition that left-hand-side path functions are longer.

However, if all PFDs in F are keys, then it is possible to find a sufficient finite subset of F_2 for which *Closure* yields the same set of path functions.

Lemma 8. *Given key PFDs F over class schema S, let* α *equal*

$$\max_{C(X \rightarrow \mathrm{Id}_C) \in F} \left(\max_{pf \in X} Len(pf) \right)$$

(i.e. the length of the longest path function mentioned in F). For any $C \in Classes(S)$ *and finite subset of path functions* $X \subseteq PathFuncs(C)$

$$Closure(X, PathFuncs(C), F_1 \cup F_2(\beta)) = Closure(X, PathFuncs(C), F_1 \cup F_2)$$

where $\beta = \max\left(1, \alpha + \max_{pf \in X} Len(pf)\right)$.

Proof. We prove that any PFD in $F_2 - F_2(\beta)$ is never used in the construction. Without loss of generality, consider a first use of such a PFD at step i. Our condition that F consists of key PFDs alone implies that the PFD has the form

$$C(pf.P_1.\ \cdots\ .P_l\ \ pf.pf_1\ \cdots\ pf.pf_m \rightarrow pf)$$

where $m \geq 0$ and $l \geq 1$ ($l = 0$ implies $pf \in X^{i-1}$). Now consider the set of path functions $Z = \{pf, pf.P_1, \cdots, pf.P_1.\ \cdots\ .P_l\}$. With respect to the longest path function in Z

- by definition of the construction, $pf.P_1.\ \cdots\ .P_l \in X^{i-1}$, and
- by definition of $F_2(\beta)$ (implying that $Len(pf) > Len(\hat{pf})$ for all $\hat{pf} \in X$), $pf.P_1.\ \cdots\ .P_l \notin X^0$.

Since, for any PFD $C(X \rightarrow \hat{pf}) \in F_2(\beta)$, $Len(\hat{pf}) < Len(pf)$, we know that $pf.P_1.\ \cdots\ .P_l$ could only have been added by the construction at some step $j < i$ by virtue of some other PFD in F_1. Since this PFD must have the form

$$C(pf.P_1.\ \cdots\ .P_{l-1} \rightarrow pf.P_1.\ \cdots\ .P_{l-1}.P_l)$$

we conclude that $pf.P_1.\ \cdots\ .P_{l-1} \in X^{i-1}$, and by a simple induction on the remaining attributes in Z, that $pf \in X^{i-1}$ — a contradiction.
□

A decision procedure that applies whenever the antecedent PFDs are keys derives in an obvious way from the following theorem.

Theorem 7. *Given key PFDs F over class schema* S, *let* α *equal*

$$\max_{C(X \rightarrow Id_C) \in F} \left(\max_{pf \in X} Len(pf) \right)$$

For any PFD $C(X \rightarrow Y)$ *also over* S, $F \models C(X \rightarrow Y)$ *iff*

$$Y \subseteq Closure(X, PathFuncs(C, \lambda), F_1(\lambda) \cup F_2(\beta))$$

where $\lambda = \alpha + \max_{pf \in XY} Len(pf)$, *and* $\beta = \alpha + \max_{pf \in X} Len(pf)$.

Proof. Without loss of generality, consider an arbitrary $pf \in Y$. If $pf \in X$, then the theorem trivially holds for this case. Otherwise, by Lemma 8

$$pf \in Closure(X, PathFuncs(C), F_1 \cup F_2)$$

if and only if

$$pf \in Closure(X, PathFuncs(C), F_1 \cup F_2(\beta))$$

Now consider a PFD $C(\hat{pf} \rightarrow \hat{pf} \circ P) \in F_1$. Clearly, if $Len(\hat{pf} \circ P) > \lambda$, then it is of no help in deriving pf, or in deriving any left-hand-side path function of any PFD in $F_2(\beta)$.
□

A straightforward variation of the mechanisms used in the linear time decision procedure for functional dependencies, called ALGORITHM 2 in [2], can also be employed to achieve a running time in this case of $O(\,|\,PathFuncs(C, \lambda)\,|\,)$. Unfortunately, $|\,PathFuncs(C, \lambda)\,|$ will almost certainly be exponential in λ, since a class scheme will usually include more than one property in its definition. Efficient decision procedures for PFDs with lengthy path functions as components is a topic for future research.

5. CONJUNCTIVE QUERIES

The applications discussed in the next section of an ability to express and reason about PFDs are better illustrated by reference to a form of conjunctive query. In effect, each application will reduce to the problem of determining if at most one solution is possible to a such a query on any consistent database (satisfying all constraints). The class of conjunctive queries we define is analogous to the class of select-join queries in the relation model, with a restricted form of projection manifested in so-called *functional joins*. In the process, we unveil a new source of PFDs induced by the join conditions of a query.

Informally, conjunctive queries that will concern us can be described in the following SQL-like manner.

SELECT $V_1 , \cdots , V_n$
FROM $C_1 , \cdots , C_n$
WHERE jc_1 AND $\cdots$ AND jc_l AND
sc_1 AND $\cdots$ AND sc_m

The "SELECT...FROM..." clauses characterize a space of possible solutions for a given interpretation that consists of n-tuples $[v_1 , \cdots , v_n]$ of vertices where $l_{Cl}(v_i) = C_i$. The *join conditions* jc_i and *selection conditions* sc_j in the "WHERE..." clause have forms $V_j \circ pf_1 = V_k \circ pf_2$ and $V_j \circ pf =$ <a constant> respectively, and are constraints on the possible solutions.

Our formal characterization of a conjunctive query is based on a simple expedient to avoid additional notation. The idea is to abstract "SELECT...FROM..." clauses as an additional class scheme of the form

$$Q\,\{\,V_1 : C_1 , \cdots , V_n : C_n\,\} \tag{5.1}$$

Each solution to the query is then represented as an entity in Q. This also permits a particularly simple characterization of the set of terms that may occur as arguments of selection or join conditions for a query. They are simply the set of property induced path

functions defined on Q. Finally, it will be unnecessary for our purposes to mention the constants that occur in selection conditions.

Definitions. A conjunctive query over a class schema S is a 3-tuple [Q, J, T], where Q is an additional class scheme of the form (5.1) above, J is a set of constraints of the form $pf_1 = pf_2$ representing *join conditions*, and T is a set of path functions representing *selection conditions*. Any path function mentioned in J or T must occur in the set *Terms*(Q), defined as

$$\{ pf \in PF_{Props}(S \cup \{Q\}) \mid Q \in Dom(pf) \}$$

We use t (possibly subscripted) to denote a member of this set. Each join condition $t_1 = t_2 \in J$ must also satisfy $Ran(Q, t_1) = Ran(Q, t_2)$.
□

Example. Consider a request for "all enrollments of Fred in courses at 9 am". The query might be expressed in the above SQL-like fashion as

```
SELECT   E, C
FROM     Enrollment, Course
WHERE    E.C = C AND
         E.S.Name = "Fred" AND C.Time = 900
```

The query includes one join condition and two selection conditions. Our abstraction of the query is the 3-tuple

$$[Q\{ E : Enrollment , C : Course \}, \{E.C = C\}, \{E.S.Name, C.Time\}]$$

□

Satisfaction of join conditions and the sense in which at most one solution is possible to a query is formally defined as follows.

Definition. Given a query [Q, J, T] over class schema S, an interpretation $G(V, A)$ satisfies a join condition $t_1 = t_2 \in J$ *iff* for any vertex $u \in V$ where $l_{Cl}(u) = Q$, $u.t_1 = u.t_2$.
□

Definition. Given PFDs F over class schema S, a query [Q, J, T] over S is *single solution limited iff* any interpretation $G(V, A)$ over S∪ {Q} that satisfies all join conditions J and PFDs $F \cup \{Q(Props(Q) \rightarrow Id_Q)\}$, must also satisfy $Q(T \rightarrow Id_Q)$.

□

To paraphrase, at most one solution can exist to a query [Q, J, T] if entities in Q, representing query solutions, are uniquely determined by the values of the path functions corresponding to the terms occurring in selection conditions. Note that admitting the PFD $Q(Props(Q) \rightarrow Id_Q)$ is mandated on the grounds that any particular solution need only be recorded once. Also note that it is unnecessary to allow only interpretations that "maximally populate" Q; that is, that encode all possible solutions to the query. The issue is clearly resolved if and only if there can exist *any* interpretation not satisfying $Q(T \rightarrow Id_Q)$.

Example. Consider the above enrollment query. If F includes the PFDs mentioned in Table 2, then the query is single solution limited since

$$Q(E.S.Name\ C.Time \rightarrow Id_Q)$$

can be derived from F and $Q(E\ C \rightarrow Id_Q)$.

□

A complete axiomatization for a theory of both PFDs and join conditions is not yet available. However, in the remainder of this section, we define a class of PFDs induced by join conditions satisfying a more limited form of completeness.

The problem of reasoning about join conditions in the absence of PFDs has been fully developed in [3], from which we reproduce the following definition and theorem.

Definition. Given a query [Q, J, T] over class scheme S, a join condition $t_1 = t_2$ is *derivable* from J, written $J \vdash_{Eq} t_1 = t_2$, *iff* it is a member of J or is the result of one or more applications of the following inference axioms.

E1. (reflexivity of equality) For every $t \in Terms(Q)$, derive $t = t$.

E2. (symmetry of equality) If $t_1 = t_2$ can be derived, then so can $t_2 = t_1$.

E3. (transitivity of equality) If $t_1 = t_2$ and $t_2 = t_3$ can be derived, then so can $t_1 = t_3$.

E4. (attribution of equality) For every $P \in Props(Ran(Q, t_1))$, if $t_1 = t_2$ can be derived, then so can $t_1.P = t_2.P$.

□

Theorem 8. *Given a query* [Q, J, T] *over class schema* S, $J \vdash_{Eq} t_1 = t_2$ *iff any interpretation* $G(V, A)$ *of* $S \cup \{Q\}$ *satisfying all join conditions in* J *must also satisfy* $t_1 = t_2$.

The set of PFDs induced by the join conditions of a query that we shall consider are defined as follows.

Definition. Given a query [Q, J, T] over class schema S, the set of all J-induced PFDs, written F_3, consists of all PFDs over S of the form $Q(t_1 \circ pf \rightarrow t_2 \circ pf)$, where $t_1 \circ pf$, $t_2 \circ pf \in Terms(Q)$ and either $t_1 = t_2 \in J$ or $t_2 = t_1 \in J$. For $i \geq 0$, $F_3(i)$ denotes all PFDs $Q(t_1 \rightarrow t_2) \in F_3$ where $Len(t_1) \leq i$ and $Len(t_2) \leq i$.
□

Soundness of an inference axiom permitting the derivation of $Q(t_1 \rightarrow t_2)$ from $t_1 = t_2$, which establishes the admissibility of PFDs in F_3, is a simple consequence of the definition of join condition satisfaction, and transitivity of equality. The sense in which F_3 satisfies a limited form of completeness is expressed in the following theorem.

Theorem 9. *Given a query* [Q, J, T] *over class schema* S, $J \vdash_{Eq} t_1 = t_2$ *only if* $F_3 \vdash Q(t_1 \rightarrow t_2)$.

Proof. Let J^+ denote any join condition derivable from J by symmetry or attribution of equality alone (i.e. by axioms *E2* or *E4*). Clearly, $Q(t_1 \rightarrow t_2) \in F_3$ iff $t_1 = t_2 \in J^+$. We show that a derivation of $t_1 = t_2$ from J can be modified to a derivation from J^+ by reflexivity and transitivity of equality alone (i.e. by axioms *E1* or *E3*). The derivation of $Q(t_1 \rightarrow t_1)$ from F_3 can then be obtained in a straightforward manner by replacing any use of *E1* by a use of *A1*, and any use of *E3* by a *double use* of *A3* (in order to preserve symmetry).

First observe that any use of *E2* or *E4* to derive a join condition jc_2 from another join condition jc_1 justified by *E1* is unproductive in the first case, and unnecessary in the second since *E1* will also suffice to justify jc_2. If jc_1 is justified by *E3*, then this use of either *E2* or *E4* can be replaced by a use of *E3* on new uses of *E2* or of *E4* introduced earlier in the derivation. For example, without loss of generality, the necessary modification for the case where *E4* is applied to a join condition derived by *E3* is illustrated as follows.

	Join Condition	Justification
	. . .	
n.	$t_1 = t_2$	?
n+1.	$t_2 = t_3$	?
n+2.	$t_1 = t_3$	(*n*, *n*+1 and *E3*)
n+3.	$t_1 \circ P = t_3 \circ P$	(*n*+2 and *E4*)
	. . .	

(before)

	Join Condition	Justification
	. . .	
n.	$t_1 = t_2$	?
n+1.	$t_1 \circ P = t_2 \circ P$	(*n* and *E4*)
n+2.	$t_2 = t_3$	?
n+3.	$t_2 \circ P = t_3 \circ P$	(*n*+2 and *E4*)
n+4.	$t_1 = t_3$	(*n*, *n*+2 and *E3*)
n+5.	$t_1 \circ P = t_3 \circ P$	(*n*+1, *n*+3 and *E3*)
	. . .	

(after)

Thus, by a simple induction, the derivation can be modified to a form in which a use of *E1* or *E3* is never followed by a use of *E2* or *E4*.
□

Corollary 2. *Let* [Q, J, T] *and* $F \cup \{Q(Props(Q) \rightarrow Id_Q)\}$ *denote a query and a set of PFDs over class schema* S. *The query is single solution limited if*

$$Id_Q \in Closure(T, PathFuncs(Q), F_1 \cup F_2 \cup F_3)$$

(where F_1 *and* F_2 *are generated with respect to* Q*).*

Proof. The corollary is a simple consequence of Theorem 9 above, Theorem 5 in the previous section, and the admissibility of PFDs in F_3.
□

An approximate procedure for determining if a query is single solution limited derives in a straightforward manner from Corollary 2. If the schema is acyclic, or at least that part of the schema referenced by the query is acyclic, then all argument sets to the closure construction are finite. Otherwise, the procedure can be supplied with a limit on the maximum length path function to be considered.

One special circumstance for cyclic schema happens when all PFDs mentioned in *F* are key PFDs, and when all join conditions mentioned in the query have *equal length argument terms*. In this case, a simple generalization of Theorem 7 in the previous section applies.

Theorem 10. *Let* [Q, J, T] *and* $F \cup \{Q(Props(Q) \rightarrow Id_Q)\}$ *denote a query and a set of key PFDs over class schema* S *in which all join conditions have equal length argument*

terms; that is, where $t_1 = t_2 \in J$ *implies* $Len(t_1) = Len(t_2)$. *Also let* α *equal*

$$\max_{pf \in T} Len(pf) + \max_{C(X \to Idc) \in F} \left(\max_{pf \in X} Len(pf) \right)$$

(i.e. the length of the longest selection condition plus the length of the longest path function mentioned in F). Then

$$Id_Q \in Closure(T, PathFuncs(Q), F_1 \cup F_2 \cup F_3)$$

is true iff

$$Id_Q \in Closure(T, PathFuncs(Q, \alpha), F_1(\alpha) \cup F_2(\alpha) \cup F_3(\alpha))$$

Proof. Clearly, if all join conditions have equal length argument terms, then no PFD in F_3 will have a right-hand-side path function that is shorter than its left-hand-side path function. This observation allows a simple generalization of Lemma 8 to be used in a proof of the above which is entirely analogous to the proof of Theorem 7.
□

6. APPLICATIONS

6.1. Choice of Index Type for Interactive Information Systems

Assume a combined index on an arbitrary choice of m properties of a class scheme $C\{P_1 : C_1, \cdots, P_n : C_n\}$ is declared as an m+1-tuple of the form

$$[<\text{IndexType}>, P_{i_1}, \cdots, P_{i_m}]$$

where <IndexType> is the kind of index, such as "BTree" or "LinearHash", and where the sequence of properties P_{i_j} establish the search criteria of the index. Clearly, any conjunctive query

$$[Q\{V : C\}, \{\}, \{P_{i_1}, \cdots, P_{i_n}\}] \tag{6.1}$$

specifying selection conditions on the m properties can be efficiently evaluated by using the index.

For many applications, it will be important to know that the index can also be efficiently maintained. Most interactive information systems, for example, have "on-line" transactions that are small-scale, involving very few changes. The overhead of maintaining a combined secondary index for such transactions may be unacceptable if the indexed properties have very little selectivity, since this will invariable require searching lengthy accession lists for individual entries. A simple application of our results helps to resolve the issue. If a query of the form (6.1) is single solution limited, then any combination of

values for the indexed properties of the index can locate at most one index entry — therefore ensuring that no search of lengthy accession lists will ever be needed.

The application can be extended in two ways. First, an answer to the same question can help with index type arbitration; that is, with selecting from among a variety of index types. For example, if it is imperative that *some* index exists supporting the above query, then determining that the query is single solution limited suggests a choice of a hash index such as [LinearHash, P_{i_1} , $\cdots$, P_{i_m}]. Otherwise a choice of an ordered index with the identity property added, such as [BTree, P_{i_1} , $\cdots$, P_{i_m} , Id_C], can still be used.

However, the theory is still overkill in the sense that these applications never require reasoning about path functions with more than one property. This is not true for databases that are memory resident. In this case, there is considerable performance incentive to allow more general path functions as search conditions for an index.[16, 17] The second way that this application can be extended therefore relates to the evaluation of memory resident combined indices of the form [<IndexType>, pf_1 , $\cdots$, pf_m]. The same issues of maintenance overhead and index type arbitration continue to apply.

6.2. Projection Elimination

Let $\pi_i(Q)$ denote the projection of the set of n-tuples that are solutions to the query

$$[\, Q\, \{ V_1 : C_1 , \cdots , V_n : C_n \}, \{ jc_1 , \cdots , jc_l \}, \{ t_1 , \cdots , t_m \}\,] \qquad (6.2)$$

on the first i components. Informally, $\pi_i(Q)$ might be expressed in an SQL-like manner as

```
SELECT   V1 , ··· , Vi
FROM     C1 , ··· , Ci
WHERE    EXIST (  SELECT   Vi+1 , ··· , Vn
                  FROM     Ci+1 , ··· , Cn
                  WHERE    jc1 AND ··· AND jcl AND
                           t1 = c1 AND ··· AND tm = cm   )
```

where the c_k are a choice of constants for the selection terms of Q. Often, a projection is necessary when evaluating a query because of the possibility of duplicate solutions; that is, the possibility that the number of solutions to $\pi_i(Q)$ is less than the number of solutions to Q. Since projection operations are expensive (they require temporary storage of solutions, along with special indices on the temporary store or a sort in order to efficiently eliminate duplicate solutions), there is a performance incentive to be able to determine

when this cannot happen.

Another application of our results derives from the observation that the number of solutions to $\pi_i(Q)$ and to Q must be equal if the first i query variables are a "key" of the relation consisting of all query solutions to Q. This is certainly true if Q is single solution limited, and will also be true if the PFD

$$Q(V_1 \cdots V_i \rightarrow V_{i+1} \cdots V_n)$$

can be derived from the user specified PFDs and the PFDs induced by the join conditions of Q. A more efficient way of stating this, involving fewer terms, is to determine if the alternative query

$$[\, \hat{Q}\, \{\, V_{i+1} : C_{i+1}\, ,\ \cdots\ ,\, V_n : C_n\, \},\, J,\, T\,]$$

is single solution limited, where the join conditions J and selection terms T are derived from (6.2) in the following manner. First, each join condition of Q contributes a join condition, a selection condition or nothing at all to $\hat{Q}$, depending on occurrences of V_k , $1 \leq k \leq i$. And second, each selection term t of Q is added to T if $t \in \mathit{Terms}(\hat{Q})$.

6.3. Cut Insertion

Mendelzon [11] has demonstrated an application of functional dependency theory to the problem of automatically inserting "cut" operations into the Horn clauses of a Prolog program. An analogous application of the theory applies to circumstances where the nested iteration method is used for evaluating join operations. However, before proceeding with further discussion, we shall require some notation for expressing an evaluation strategy for a query, sometimes called an *access strategy language* (ASL).

A simple ASL that will suffice for our illustrative purposes consists of sentences that are a finite sequence $[\, e_1\, ,\ \cdots\ ,\, e_n\,]$, where each element e_i can have one of four possible forms: 1) a query variable V : C, 2) a join condition $t_1 = t_2$, 3) a selection term t, or 4) a cut operator !V. Sentences also satisfy the property that any query variables mentioned in a join condition, selection term or cut operator occur previously in the sequence. Recursive "re-write" rules suggestive of an operational semantics for the ASL, in terms of a Pascal-like language, are as follows.

```
[V : C , ··· ]   ≡   CutV := false;
                    V := "first C object";
                    while not CutV and not V = nil do begin
                       [ ··· ];
                       if not CutV then V := "next C object"
                    end

[t1 = t2 , ··· ]  ≡  if t1 = t2 then begin [ ··· ] end

[t , ··· ]        ≡  if t = "the selection value for t" then begin
                       [ ··· ]
                    end

[!V , ··· ]       ≡  CutV := true; [ ··· ]

[ ]               ≡  "remember the solution"
```

Example. Consider the enrollment query

[Q { E : Enrollment }, { }, {E.S.Name, E.C.Time}]

denoting all enrollments for a given combination of student name and course time. Since the query is single solution limited (according to step 10 in Table 3), a strategy for evaluating the query can use a cut operator.

[E:Enrollment, E.S.Name, E.C.Time, !V]

Assuming "Fred" and 900 are the selection values for terms E.S.Name and E.C.Time respectively, then the above operational semantics defines this strategy as the following Pascal-like code.

```
CutE := false;
E := "first Enrollment object";
while not CutE and not E = nil do begin
   if E.S.Name = "Fred" then begin
      if E.C.Time = 900 then begin
         CutE := true;
         "remember the solution"
      end
   end;
   if not CutE then E := "next Enrollment Object"
end
```

Clearly, the cut operator improves the performance of the strategy since it terminates the scan of enrollment after finding the first (and necessarily last) enrollment object that satisfies the two selection conditions.
□

In general, we would like to know when cuts may be safely inserted; that is, we require a means of determining if a sentence in our ASL of the form

$$[\ \cdots\ , V : C\ ,\ \cdots\ ,\ e_i\ ,\ e_{i+1}\ ,\ \cdots\ ] \tag{6.3}$$

produces the same set of query solutions as the sentence

$$[\ \cdots\ , V : \dot{C}\ ,\ \cdots\ ,\ e_i\ ,\ !V\ ,\ e_{i+1}\ ,\ \cdots\ ] \tag{6.4}$$

The application of our results to this problem should now be clear: (6.3) can be replaced by (6.4) if the query

$$[\ Q\{\ V : C\ \},\ J,\ T\ ]$$

is single solution limited, where the join conditions J and selection terms T are determined from those occurring in the sequence prior to entry e_{i+1} in the following manner. First, a join condition entry contributes a join condition, a selection term or nothing at all, depending on occurrences of V. And second, each selection term entry t is added to T if $t \in Terms(Q)$.

Two obvious ways that this application improves on that of Mendelzon's are due to the more general form of functional dependency considered and to some accounting of the effects of join conditions. Another less apparent improvement derives from the fact that cuts in our case *name a particular variable*. For example, there is no possible annotation of horn clauses with cut operators that will result in the same evaluation strategy as

the following ASL sentence.

$$[\cdots, V_1 : C_1, \cdots, V_2 : C_2, \cdots, !V_1, \cdots, !V_2, \cdots]$$

7. DISCUSSION AND SUMMARY

Object-oriented data models manifest a more general form of aggregation abstraction in comparison to the relational model. In particular, they allow objects to have properties which are other objects which may themselves have properties and so on. Typically, this results in a query language in which terms can navigate from entity to entity by following a property value path. Since all objects are assumed to exist separately, such paths may be of zero length.

We have proposed a more general functional dependency constraint, called a *path functional dependency* (PFD), to account for this circumstance. First, we allow component attributes to correspond to a sequence of property names, called *property induced path functions*. Such sequences are a means of characterizing a property value path of non-zero length in an object-oriented database. Second, we allow component attributes to correspond to what we call *identity path functions*, which are a means of describing zero-length paths; that is, a means of referencing object identity directly. And third, we require each PFD to name a particular object class for which it applies.

Inference axioms for PFDs are presented, and are proven sound and complete with respect to a graph theoretic notion of interpretation. Decision procedures are then derived that can be used whenever a schema is acyclic, or when antecedent PFDs are keys. The theory was then extended to incorporate a limited form of select-project-join query. Of particular significance is that the join conditions of a query become an additional source of PFD constraint. A limited completeness result for this more general circumstance is given. Finally, we outlined several applications of the theory to various problems in physical design and in query optimization. The applications derive from an ability to predict when queries can have at most one solution.

There are many interesting questions about the theory of PFDs that remain. For example, our proof of completeness requires that one allows infinite interpretations for cases where a schema is cyclic. Proving "finite completeness" is an open problem. Another example concerns the general problem of PFD derivability. We know the set of PFDs derivable from a given set is recursively enumerable, this is established in the paper, but we do not know if the set is recursive. Even for acyclic schema, the number of path functions that may have to be examined in the forward reasoning approach manifest in our decision procedures will almost certainly be exponential in the length of path

function permitted. Alternative goal-directed approaches are an attractive possibility.

8. References

1. ADAPLEX: Rational and reference manual, CCA-83-08, Computer Corporation of America (May 1983).
2. C. Beeri and P. A. Bernstein, Computational problems related to the design of normal form relational schemas, *ACM Transactions on Database Systems* **4**(1) pp. 30-59 (March 1979).
3. M. van Bommel and G. E. Weddell, Reasoning about selection conditions for an object-oriented data model, submitted for publication (1989).
4. A. Borgida, Features of languages for the development of information systems at the conceptual level, *IEEE Software* **2**(1) pp. 63-72 (January 1985).
5. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, SEQUEL 2: A unified approach to data definition, manipulation, and control, *IBM Journal of Research and Development* **20**(6) pp. 560-575 (November 1976).
6. E. F. Codd, Extending the database relational model to capture more meaning, *ACM Transactions on Database Systems* **4**(4) pp. 397-434 (December 1979).
7. A. Goldberg and D. Robson, *Smalltalk-80; The Language and its Implementation,* Addison-Wesley (1983).
8. G. D. Held, M. R. Stonebraker, and E. Wong, INGRES - A relational data base system, *Proc. National Computer Conference* **44**(1975).
9. W. Kent, Limitations of record-based information models, *ACM Transactions on Database Systems* **4**(1) pp. 107-131 (March 1979).
10. D. Maier, *The Theory of Relational Databases,* Computer Science Press (1983).
11. A. Mendelzon, Functional dependencies in logic programs, *Proc. Eleventh International Conference on Very Large Data Bases*, pp. 324-330 (August 1985).
12. 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) pp. 185-207 (June 1980).
13. J. M. Smith and D. C. P. Smith, A database approach to software specification, Technical Report 17, Computer Corporation of America (1979).

14. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley (1986).

15. J. D. Ullman, *Principles of Database Systems (second edition)*, Computer Science Press (1982).

16. G. E. Weddell, Selection of indices to memory-resident entities for semantic data models, *to appear in IEEE Transactions on Knowledge and Data Engineering*, ().

17. G. E. Weddell, Physical design and query compilation for a semantic data model (assuming memory residence), Technical Report 198, Computer Systems Research Institute, University of Toronto (1987).

18. C. Zaniolo, The database language GEM, *Proc. ACM SIGMOD Conference on Management of Data*, pp. 207-218 (May 1983).