Functional Dependencies on Cyclic Database Schemes

Kent Laver
Alberto O. Mendelzon

Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A4

Marc H. Graham

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

ABSTRACT

We study how functional dependencies affect the cyclicity of a database scheme; in particular, when does a set of functional dependencies make a cyclic database scheme behave like an acyclic one.

A database scheme is *fd-acyclic* if every pairwise-consistent database state that satisfies the fd's is join-consistent. We give a simple characterization of fd-acyclicity over a restricted class of database schemes. We then give a tableau-based characterization for the general case that leads to an algorithm for testing fd-acyclicity. This algorithm actually solves the more general problem of query equivalence under functional dependencies and typed inclusion dependencies.

1. INTRODUCTION

Acyclic database schemes have been shown to have remarkable properties [B+,BG,FMU]. In this paper, we study how functional dependencies affect the cyclicity of a database scheme. In particular, since cyclic schemes seem to appear often in practice, we would like to know when functional dependencies have the effect of making a cyclic scheme behave like an acyclic one. This would let us exploit the desirable properties of acyclic schemes on a larger class of schemes provided we ensure the functional dependencies are satisfied.

A natural approach to this question is to consider those characterizations of acyclicity that are defined in terms of database states, say "every state that has property P has also property Q", and consider the modified version "every state that has property P and satisfies a set of functional dependencies F has also property Q". The simplest such characterization is "every pairwise consistent state is join-consistent", where pairwise consistency means that any two relations in the state agree on their common attributes and join-consistency means the state is a projection of a universal relation. It is well known (see e.g. [BFMY]) that this property is equivalent to acyclicity of the database scheme.

Let us now consider the modified version "every pairwise consistent state that satisfies a set of functional dependencies F is also join-consistent". The first question that arises is what do we mean by a state satisfying a set of functional dependencies. Several definitions are possible; we choose to use the one proposed by Honeyman [H] and Vassiliou [V] for reasons discussed in [H,V]. A state is said to satisfy a set of functional dependencies F if there exists a universal

relation, I, on the union of all the attributes such that I satisfies F and the projection of I on each relation scheme in the database is a superset of the corresponding relation in the state. I is called a *weak instance* for the state, and this notion usually called *weak instance satisfaction*. We will say that a database scheme is *fd-acyclic* with respect to the set of functional dependencies F if every pairwise consistent database state that has a weak instance under F is also join-consistent.

Example 1: Let our database scheme have three relation schemes, PS, ST, PT, where P stands for Professor, S for Student, and T for Topic. A ps tuple means professor p is supervising student s, an st tuple means student s is working on topic t, and a pt tuple means professor p is supervising a thesis on topic t. Suppose the functional dependencies $S \rightarrow P$ and $T \rightarrow P$ hold, that is, each student has a unique professor and there is no overlap of topics between professors. This database scheme is cyclic. We claim that it is fd-acyclic.

In proof, we need to show that every pairwise consistent state satisfying the functional dependencies is join-consistent. It suffices to show that if we join such a state and then project back onto PS, ST, and PT, we get exactly the same state back. Let ps be a PS-tuple. Since the state is pairwise-consistent, there exist tuples st in ST and pt' in PT. We know a weak instance for the state exists. This weak instance will have a tuple whose PS projection is ps and a tuple whose PT projection is pt'. Since the weak instance must satisfy $P \rightarrow T$, t and t' must be equal. That is, there exist tuples st in ST and pt in PT that join with ps, showing that ps appears in the join and thus it appears in the PS-projection of the join of the state. By similar arguments we can show that every tuple in every relation of the state appears in the appropriate projection of the join of the state, so the state is join-consistent. ■

After giving basic definitions in Section 2, in Section 3 we study the special class of cyclic schemes called *A-rings*, first considered by Goodman and Shmueli [GS]. We give a simple characterization of fd-acyclicity on A-rings and an efficient algorithm for testing it.

In Section 4, we consider an arbitrary database scheme and show that the question of fd-acyclicity can be reduced to a query containment problem. That is, given a database scheme and a set of dependencies we can easily construct two project-join queries Q_1 and Q_2 such that the database scheme is fd-acyclic if and only if $Q_1(D)$ contains $Q_2(D)$ for every pairwise consistent state D that satisfies the functional dependencies. Since Q_1 and Q_2 can be represented by *tableaux*, following [ASU], we show how to use the *chase* technique to provide a containment test where functional dependencies plus the pairwise consistency constraint are given. To do this we note that the pairwise consistency constraint can be viewed as a set of *inclusion dependencies*. These are statements of the form "$r[X]$ must be a subset of s[Y]", where r and s are relations in the state and X, Y are subsets of the appropriate relation schemes. Inclusion dependencies have aroused much interest lately; in particular, Johnson and Klug [JK] have given an exponential time algorithm for the query containment problem under a set of functional and inclusion dependencies with the "key-based" property.

In Section 5, we show that there exists an effective procedure to test the tableaux containment problem under a set of *typed* inclusion dependencies and arbitrary functional dependencies. A corollary of this result is that there exists an effective procedure to test for fd-acyclicity.

2. DEFINITIONS

2.1. BASICS

U is a finite set of *attributes*. A *database scheme* H is a set of non-empty subsets of U. We say a database scheme is *connected* if the associated hypergraph (see [B+]) is connected. For the purposes of this paper all database schemes are connected. Elements of H are called *relation schemes*. Associated with each attribute $A \in U$ is a (countable) set of constants called the *domain* of A or $dom(A)$. We assume $dom(A) \cap dom(B) = \phi$ for $A \neq B$. A *tuple* t over a relation scheme $R = \{A_1, \ldots, A_k\} \in H$ is an element in $dom(A_1) \times \cdots \times dom(A_k)$. A *relation* r for R is a (countable) set of tuples over R. A *database state* for a given scheme, H, is an assignment D of relations to the elements of H. If t is a tuple over the scheme R and X is a subset of the attributes of R, $t[X]$ is the

restriction of t to the attributes of X. If, in addition, r is a relation over R, the *projection* of r onto X is

$$\pi_X(r) = \{t[X] \mid t \in r\}.$$

If r and s are relations over R and S, the *join* of r and s is

$$r*s = \{t \mid t[R] \in r \text{ and } t[S] \in s\}.$$

Σ is a set of *constraints*. The constraints of interest here are the *functional dependencies* (fd's). Let X and Y be sets of attributes and R a relation scheme such that $R \supseteq XY$. The fd $X \rightarrow Y$ is said to hold in a relation r over R if for any two tuples t,u in r $t[X] = u[X]$ implies $t[Y] = u[Y]$. Given a set of fd's, $\{X_i \rightarrow Y\}_{i=1}^{n}$, we will write $X_1 / \ldots / X_n \rightarrow Y$. To avoid repetition, whenever we write of a database scheme H, we will be referring to both the relation schemes in H and the set of constraints Σ.

A state D satisfies a set of fd's Σ, or is in $SAT(\Sigma)$, if there exists a relation I over U which satisfies Σ such that, for each R_i, $\pi_{R_i}(I) \supseteq D(R_i)$. When there exists a relation I over U which satisfies Σ such that, for each R_i, $\pi_{R_i}(I) = D(R_i)$, D is said to be *join-consistent* (JC). D is *pairwise-consistent* (PC), if for every pair of relation schemes, R and S, and their respective relations, r and s,

$$\Pi_{R \cap S}(r) = \Pi_{R \cap S}(s).$$

A database scheme H is *acyclic* if pairwise-consistency is equivalent to join-consistency, otherwise it is cyclic. For further discussion as well as equivalent definitions of acyclicity see [B+,BFMY,BG,FMU].

2.2. TABLEAUX, VALUATIONS

A *tableau* , T, is a tabular representation of a mapping from states or tableaux to relations. Each column of a tableau corresponds to an attribute in the universe. The tableau domain of the i-th column, corresponding to some attribute A, consists of, the *distinguished variable* (dv), a_i, *non-distinguished variables* (ndv's), b_i's, and *constants* taken from the domain of A. The tableau domains of two distinct columns are disjoint.

The first row of the tableau, called the summary, may only contain distinguished variables, constants, and blanks. The attributes which correspond to non-blank columns of the summary define the *target relation scheme* of the tableau. In a tableau each row, except the summary is *tagged* by some relation scheme in H. (Note that in presenting examples of tableaux non-distinguished variables which occur only once and are not in columns represented in the tuple's tag will be replaced by blanks.)

The image of the mapping represented by the tableau is a relation over the target relation scheme. The value of this mapping is determined by the set of *valuation functions*, which we now define.

A valuation function, ν, is a mapping on the set of symbols that appear in a tableau, T. For each column in T it maps the tableau domain of that column into itself such that it sends a ndv to a ndv or the dv or a constant, sends the dv to itself or a constant, and sends a constant to itself. Valuations also map tags to tags and a *tag-preserving* valuation is the identity on tags.

ν will also refer to the set and tuple-wise extension of ν, ie., $\nu(t) = <\nu(t[A_1]), \ldots, \nu(t[A_n])>$ and $\nu(\{t_1, \ldots, t_k\}) = \{\nu(t_1), \ldots, \nu(t_k)\}$.

Consider some row, w, with tag R_i, in a tableau T. We say that ν is a valuation from T into a database state D if for each such row w the R_i-projection of $\nu(w)$ is in the relation over R_i in D. Let ν be a tag-preserving valuation function. Then ν is a valuation from a tableau T into a tableau T' if $\nu(T) \subseteq T'$.

The value of a tableau T on a state or tableau σ is

$$T(\sigma) = \{\nu(s) \mid s \text{ is the summary row for } T \text{ and}$$
$$\nu: T \rightarrow \sigma \text{ is a valuation function}\}$$

For notational convenience, we allow the empty tableau, denoted ϕ_T, whose body is the empty set. Evaluation of the empty tableau is the special case,

$$\phi_T(\sigma) = \phi \text{ for any } \sigma.$$

We can alleviate the confusion of having two kinds of valuation functions by noticing that any state D gives rise to a unique (up to renaming of non-distinguished variables) tableau T_D to which, for the purpose of tableau evaluation, it is identical. T_D is

defined as follows:

For every relation $D(R_i)$ and every tuple $t \in D(R_i)$ there is a row u of T_D with

- $u[R_i] = t$
- $u[B] \in Ndv(B)$ appearing nowhere else in T_D for $B \in U - R_i$.
- the tag of u being R_i

The summary row of T_D is undefined; i.e., the target relation scheme is empty. It is obvious that $T(D) = T(T_D)$ for any tableau T and state D.

2.3. CONTAINMENT

Fact 1: For every expression E over select, project and join there is a tableau T_E such that for every state D, $T_E(D) = E(D)$.

Proof: see [ASU] Theorem 1. ■

An expression E_1 is said to *contain* an expression E_2, written $E_1 \supseteq E_2$, if for every state D, $E_1(D) \supseteq E_2(D)$. Similarly for tableaux $T_1 \supseteq T_2$ if for every σ (state or tableau) $T_1(\sigma) \supseteq T_2(\sigma)$. Equivalence is containment in both directions; i.e., $E_1 \equiv E_2$ ($T_1 \equiv T_2$) abbreviates $E_1 \supseteq E_2$ and $E_2 \supseteq E_1$ ($T_1 \supseteq T_2$ and $T_2 \supseteq T_1$). Consider some set of constraints Σ. $E_1 \supseteq_\Sigma E_2$ ($E_1 \equiv_\Sigma E_2$), if for every state $D \in SAT(\Sigma)$, $E_1(D) \supseteq E_2(D)$ ($E_1(D) \equiv E_2(D)$).

A tag preserving valuation $\nu: T_1 \rightarrow T_2$ is a *containment mapping* if $\nu(s)$ (s the summary of T_1) is the summary of T_2.

Fact 2: $T_1 \supseteq T_2$ iff there exists $\nu: T_1 \rightarrow T_2$, ν a containment mapping. Thus $E_1 \supseteq E_2$ iff $T_{E_1} \supseteq T_{E_2}$ iff there exists $\eta: T_{E_1} \rightarrow T_{E_2}$, η a containment mapping.

Proof: [ASU] Theorem 2. ■

2.4. THE CHASE

The *chase* is a process for converting a tableau into one which satisfies a given set of constraints. The chase of a tableau T with respect to a set of fd's Σ, called $chase_\Sigma(T)$, is the result of exhaustively applying the following transformation rule ,the fd-rule, to T.

Assume $X \rightarrow A$ is an fd in Σ. Assume that there are two rows, t and s, in T, such that, $t[X] = s[X]$ and $t[A] \neq s[A]$. In this case we say that the fd-rule is *enabled* and identify the symbols $t[A]$ and $s[A]$. We do this identification as follows: if one of $t[A]$ or $s[A]$ is a ndv (assume $t[A]$ is) then the symbol $t[A]$ is replaced by $s[A]$; if $t[A]$ is a dv then the symbol $t[A]$ is replaced by $s[A]$ (note that $s[A]$ must be a constant); otherwise the tableau which results from applying the fd-rule is ϕ_T. This last case will occur exactly when $t[A]$ and $s[A]$ are constants.

Fact 3: A non-empty state D satisfies a set of fd's Σ iff $chase_\Sigma(T_D) \neq \phi_T$.

Proof: [H] Theorem 1. ■

2.5. FD-ACYCLICITY

In this section we formally define fd-acyclicity, in both its finite and infinite forms.

Definition: A database scheme is *(finite) functional dependency-acyclic* (fd-acyclic) with respect to a set of fd's if every (finite) satisfying pairwise-consistent state is join-consistent.

We shall say that if a database scheme is not (finite) fd-acyclic then it is (finite) fd-cyclic.

All real world database states are finite. Therefore such states are our ultimate concern. However, the definition of fd-acyclicity considers all countable states because, as will become clear, such states provide a natural framework in which to analyze fd-acyclicity.

Example 1: Consider the database scheme $\{R_i\}_{i=1}^{4}$ where $R_1 = \{AB\}$, $R_2 = \{BC\}$, $R_3 = \{CD\}$, $R_4 = \{AD\}$, together with the constraints $\Sigma = \{B/C \rightarrow AD\}^*$. Let D be the following state,

A	B
0	0
1	1

B	C
0	0
1	1

C	D
0	0
1	1

A	D
0	1
1	0

* recall that $\{B/C \rightarrow AD\}$ is our shorthand for $\{B \rightarrow AD, C \rightarrow AD\}$.

D is finite and pairwise consistent by observation and not join consistent (since the join over all the relations is empty). T_D^* is shown below.

A	B	C	D	Tag
0	0		0	AB
1	1		1	AB
0	0	0	0	BC
1	1	1	1	BC
0		0	0	CD
1		1	1	CD
0			1	AD
1			0	AD

Since no contradiction was uncovered in the chase ($T_D^* \neq \phi_T$) D is a satisfying state and this scheme is (finite) fd-cyclic.

Let $\Sigma' = \{B/C/D \rightarrow A\}$. Keeping the same database scheme as above and letting Σ' be the set of constraints the reader can verify that $T_D^* = \phi_T$. It will be shown later in this section that this database scheme is fd-acyclic. ■.

It follows from the definitions above that any acyclic database scheme is fd-acyclic. Also if there are no fd's then acyclicity, fd-acyclicity and finite fd-acyclicity all are equivalent.

Because fd-acyclicity is defined in terms of all countable states, finite or infinite, it follows that any fd-acyclic database scheme is finite fd-acyclic. However, a database scheme, H, might be such that there would exist some infinite, satisfying, PC and non-JC state, although any finite, satisfying and PC state would be JC. In these circumstances, H would be fd-cyclic without being finite fd-cyclic.

3. A CHARACTERIZATION OF FD-ACYCLICITY ON A SIMPLE CLASS OF SCHEMES

In this section we will present a characterization of fd-acyclicity on a restricted class of database schemes. and show that on this class the finite and infinite forms of fd-acyclicity agree. We will then present a polynomial time test for fd-acyclicity on this class.

Definition: Let $H = \{R_i\}_{i=1}^{n}$ be a database scheme. H is a *Berge-cycle* if: 1) $m \geq 2$; 2) $\exists x_1, \ldots, x_m$ distinct; 3) $\exists R_1, \ldots, R_m$ distinct; 4) $x_i \in R_i \cap R_{i+1} = I_{i+1}$; (where the I_i's are *intersection sets*, and for notational convenience we define R_{n+1} and I_{n+1} to be R_1 and I_1 respectively). An attribute, A, is *pinned* if $\forall j$ $I_j \rightarrow A$. An intersection set is *pinned* if each attribute in it is pinned. A Berge-cycle is *pinned* if at least one of its intersection sets is pinned.

Berge-cycles have been previously considered in [BFMY].

Example 1: Consider the database scheme H and the two sets of constraints Σ and Σ' presented in Example 1 of Section 2. H is a Berge-cycle in which the intersection sets are $\{A\}, \{B\}, \{C\}$ and $\{D\}$. Under the constraints Σ H is not pinned while under Σ' it is, since A is pinned. ■

Definition: Let $U = \{A_i\}_{i=1}^{n}$, for some $n \geq 3$. A Berge-cycle $H = \{R_i\}_{i=1}^{n}$ is an *A-ring* if $\forall 1 \leq j \leq n-1$, $R_j = \{A_j, A_{j+1}\}$ and $R_n = \{A_1, A_n\}$.

A-rings were originally defined by Goodman and Shmueli [GS]. They are one of two classes of database schemes to which any cyclic scheme can be reduced. Because of this property they were used in the study of natural join queries and cyclic schemes.

We will now characterize fd-acyclicity on A-rings.

Theorem 1: An A-ring H is finite fd-acyclic iff it is pinned.

Proof: (if) See [L].

(only if) The proof that every fd-acyclic A-ring is pinned will be by counter-example. That is, we will present a state, $D = \{r_i\}$, where r_i is the relation over R_i, such that D is finite, pairwise-consistent, satisfying and not join-consistent.

$$r_i = \{t_i^1, t_i^0\} \quad i = 1, \ldots, n$$

where $t_i^1|_{A_i} = 1$, $t_i^0|_{A_i} = 0 \quad \forall 1 \leq i \leq n$

$$t_i^1|_{A_{i+1}} = 1, \; t_i^0|_{A_{i+1}} = 0 \quad \forall 1 \leq i \leq n-1$$

$$t_n^1|_{A_1} = 0, \; t_n^0|_{A_1} = 1.$$

This state is clearly finite and PC. To see that it

is not join-consistent note that the join is empty. For the proof that D is satisfying see [L]. ■

The database scheme presented in Example 1 of Section 2 is an A-ring and the state, D, presented there is one example of the class of states described in Lemma 2.

Corollary 2: Fd-acyclicity and finite fd-acyclicity are equivalent on A-rings.

Proof: Since any fd-acyclic scheme is finite fd-acyclic we need only show that an fd-cyclic A-ring is finite fd-cyclic. But the counter-example presented in Theorem 1 is finite. Thus the two concepts are equivalent on A-rings. ■

The fact that fd-acyclicity can be expressed as a property of the fd's, i.e. that the A-ring is pinned, leads to the following test and complexity bound for fd-acyclicity.

Corollary 3: Let H be an A-ring. Testing if H is fd-acyclic can be done in time polynomial in the number of fd's and the size of the database scheme.

Proof: The procedure is as follows: 1) for each intersection set, I_i, calculate its closure, I_i^+; 2) calculate $I = \bigcap_{j=1}^{n} I_j^+$; 3) determine if there exists some intersection set I_i such that $I_i \subseteq I$. Such an intersection set exists iff H is fd-acyclic.

Each of these steps can be done in time polynomial in the number of relation schemes in H and the size of a cover of the set of fd's. ■

The characterization of (finite) fd-acyclicity just presented is a particular case of a characterization of fd-acyclicity on a more general class of database schemes. The class is described and the characterization presented in [L].

4. FD-ACYCLICITY AND TABLEAUX

In the previous section a characterization of (finite) fd-acyclicity on a restricted class of database schemes was presented. The characterization involved an explicit condition, pinning the A-ring. In this section we will characterize fd-acyclicity over all database schemes. This characterization will be based on evaluating expressions and testing tableaux containment.

Definition: Consider any $R \in H$. Let T_R and J_R be tableaux with the target relation scheme R and summary rows containing only dv's. In addition let T_R contain a single row with tag R and distinguished variables on the attributes in R and non-distinguished variables elsewhere. Let J_R be the tableau corresponding to the join over the entire state projected onto R.

Example 1: Let H be the database scheme presented in Example 1 of Section 2. T_{R_1} is then

A	B	C	D	Tag
a_1	a_2			
a_1	a_2			AB

and J_{R_1} is

A	B	C	D	Tag
a_1	a_2			
a_1	a_2			AB
	a_2	b_1		BC
		b_1	b_2	CD
a_1			b_2	AD

■

Theorem 1: Let Σ be a set of fd's. A database scheme H is (finite) fd-acyclic iff

$$\forall R \in H, \text{ for every (finite) PC state } D$$

$$\text{such that } D \in SAT(\Sigma),$$

$$T_R(T_D) = J_R(T_D).$$

Proof: Let $H = \{R_i\}$. Let $D = \{r_i\}$ be any (finite) satisfying state on H. For the state to be join- consistent requires that for each R_i

$$\pi_{R_i}(\bowtie_j r_j) = r_i$$

Therefore by the definition of the tableaux J_{R_i} and T_{R_i}, $J_{R_i}(T_D) = T_{R_i}(T_D)$. ■

Given Theorem 1, and results such as Fact 3 in Section 2, it is reasonable to consider the implications of chasing T_R under Σ together with the predicate PC. To do this we will view the predicate PC as a collection

of predicates, inclusion dependencies, each of which involves a pair of relation schemes.

The next section deals with defining the chase under these constraints. We will define inclusion dependencies and the chase of a tableau under these dependencies. Then a constraint on tableaux which is closely related to inclusion dependencies will be introduced. The subsequent section will present a graphical representation of the chase followed by a characterization of fd-acyclicity in terms of chased tableaux.

4.1. EXTENDING THE CHASE

4.1.1. INCLUSION DEPENDENCIES

Definition: Consider a database scheme H and a state D, with $R_1, R_2 \in H$, $S \subseteq R_1 \cap R_2$ and $r_1, r_2 \in D$ the relations on R_1 and R_2 respectively. D satisfies the (typed) *inclusion dependency* (id) $[R_1, R_2, S]$ if $\pi_S(r_1) \subseteq \pi_S(r_2)$.

Therefore, D is PC iff it satisfies each of the id's $[R_1, R_2, R_1 \cap R_2]$ $\forall R_1, R_2 \in H$.

We will define a chase rule for id's which corresponds to the chase rule for fd's. Let T be a tagged tableau and consider some id $[R_1, R_2, S]$. Assume that there is a row w in T with tag R_1 and that there is no row with tag R_2 in T which agrees on S with w. Given this situation the *id-rule*, consisting of the row w and the id $[R_1, R_2, S]$, is enabled and it can be applied to add a row with tag R_2 to T where the row agrees on S with w and has new non-distinguished variables on $U-S$. We will say that T satisfies a set of dependencies, Σ, (fd's and id's) if no chase-rule based on a dependency in Σ is enabled.

Let Σ be any set of fd's and id's. For any tableau, T, chasing under Σ consists of repeated applications of any enabled fd or id rule subject to the following restriction. Let T_1 be some intermediate tableau formed by the application of some sequence of enabled chase-rules to T. If there is a chase rule, τ, in Σ that is enabled on T_1 then after some finite number of chase-rule applications τ will no longer be enabled. Any tableau which results from this process (i.e. a tableau in which no transformations remain enabled) will be referred to as $chase_\Sigma(T)$.

Theorem 2: Let Σ be any set of fd's and id's. Let T be any finite tableau. Then $chase_\Sigma(T)$ exists and is in $SAT(\Sigma)$.

Proof: See [L]. ■

When Σ contains fd's and id's the tableau $chase_\Sigma(T)$ will also be referred to as T^*. Two things should be noted at this point. First, that $chase_\Sigma(T)$ can in general be infinite, and so will not be calculatable. Second, that $chase_\Sigma(T)$ is not uniquely defined. This is because at any point in the chase if some set of transformations is enabled, the resulting tableau will be a function of the order in which the transformations are applied. In the example below two tableaux are presented. Each of them is the result of chasing the same initial tableau under the specified id's. Because of this property we will use $chase_\Sigma(T)$ to refer to both some particular tableau, and the set of tableaux which can be obtained by chasing T under Σ. The meaning should be clear from the context.

Example 2: Let H be the database scheme $\{R_i\}_{i=1}^{3}$ where $R_1 = \{ABC\}$, $R_2 = \{BD\}$, $R_3 = \{BCE\}$ and Σ is the set of id's implied by pairwise-consistency. Consider $chase_\Sigma(T_{R_1})$. We present two distinct tableaux, T^1 and T^2, where both are in $chase_\Sigma(T_{R_1})$. T^1 is

A	B	C	D	E	Tag
a_1	a_2	a_3			
a_1	a_2	a_3			ABC
	a_2	a_3		b_1	BCE
	a_2		b_2		BD

T^2 is

A	B	C	D	E	Tag
a_1	a_2	a_3			
a_1	a_2	a_3			ABC
	a_2		b_1		BD
	a_2	b_2		b_3	BCE
b_4	a_2	b_2			ABC
	a_2	a_3		b_5	BCE

In T_{R_1} there are two enabled chase-rules. They are based on the two id's $[R_1, R_2, B]$ and $[R_1, R_3, BC]$.

T^1 was formed by first applying the id-rule based on the id $[R_1,R_3,BC]$. In T^2 the first id-rule applied was based on $[R_1,R_2,B]$. ■

Although $chase_\Sigma(T)$ is not uniquely defined we can show that any two tableaux in it are equivalent.

Lemma 3: Let Σ be any set of id's and fd's. Let T be any finite tableau. Let T^1 and T^2 be any pair of tableaux in $chase_\Sigma(T)$. Then $T^1 \equiv T^2$.
Proof: See [L]. ■

Within any tableau that can be obtained by applying chase-rules to T, each row can be assigned a number, called its level. This is done as follows;

1) any row in T is at level zero,
2) a row added to the tableau by the application of an id-rule to a row at level n, is at level n+1.

We define the function *level* which, for any tuple in the chase returns that tuple's level.

4.1.2. EXTENDED INCLUSION DEPENDENCIES

In this section we introduce a constraint on tableaux that will aid us in analyzing the chase of a tableau. We will first define the constraint and then present some results.

Functional and inclusion dependencies are defined in terms of states. The semantic question of whether or not a state satisfies these dependencies is replaced by a syntactic question on tableaux. Tableaux are used for testing satisfaction under a set of dependencies or for testing expression equivalence. When a tableau does not satisfy one of these dependencies it can be transformed by a chase-rule. In an id $[R_i,R_j,S]$, S is defined such that $S \subseteq R_i \cap R_j$. In tableaux though two rows may agree on any set $S \subseteq U$. Thus we define an *extended-inclusion dependency* (ex-id) $[R_i,R_j,S]$, in the same manner as id's except that the only restriction on S is that $S \subseteq U$. A tableau T satisfies this ex-id if for every row $t_i \in T$ with tag R_i there is a row t_j with tag R_j such that $t_i[S] = t_j[S]$. The chase-rule for ex-id's, the ex-id-rule, is the obvious extension of the id-rule.

Ex-id's would seem to have no obvious meaning on states since in an ex-id the set S need have no relationship to the relation scheme H. We shall use them to increase our understanding of the chase process and in determining the properties of $chase_\Sigma(T)$. Because of this all of the following discussion on ex-id's will be in terms of tableaux.

Our use for ex-id's is to avoid chasing under fd's. This is possible because ex-id's, in terms of chasing a tableau under some set of fd's and id's, can fulfill the role played by fd's.

Consider any id $[R_1,R_2,S]$ and any set of fd's. For our purposes this id will be replaced by the ex-id $[R_1,R_2,S^+]$. Given a set of fd's and id's Σ we will refer to the associated set of ex-id's as Σ^e. Also, let Σ^f be the set of all the fd's in Σ.

Example 3: Consider the database scheme $\{R_i\}_{i=1}^3$, where $R_1 = \{ABC\}$, $R_2 = \{BCD\}$ and $R_3 = \{DE\}$, with constraints $\Sigma = \{[R_1,R_2,BC],\ [R_2,R_3,D],\ BC \rightarrow E,\ D \rightarrow C\}$.

The set of ex-id's is then $\{[R_1,R_2,BCE],[R_2,R_3,CD]\}$, and $chase_{\Sigma^e}(T_{R_1})$ is

A	B	C	D	E	Tag
a_1	a_2	a_3			
a_1	a_2	a_3		b_1	ABC
	a_2	a_3	b_2	b_1	BCD
		a_3	b_2	b_3	DE

■

The following simple results describe the relationship between fd's, id's and ex-id's.

Lemma 4: Let $R_1,R_2 \in H$ and $S \subseteq R_1 \cap R_2$. Let Σ be a set of fd's.

a) $\Sigma \cup \{[R_1,R_2,S]\} \vdash [R_1,R_2,S^+]$

b) $\{[R_1,R_2,S^+]\} \vdash [R_1,R_2,S]$.

Proof: Both parts are trivial. ■

This shows that any tableau chased under fd's and id's Σ, will satisfy the set of associated ex-id's Σ^e. It also shows that a tableau chased under the set of ex-id's Σ^e, will satisfy the id's in Σ. The next Lemma will show when the resulting tableau will also satisfy the fd's.

Lemma 5: Let Σ, Σ^e, and Σ^f be defined as above. Let

T be any finite tableau that satisfies Σ^f. Then $chase_{\Sigma^e}(T) \in SAT(\Sigma^f)$.

Proof: See [L]. ■

Therefore

$$chase_\Sigma(T) = chase_{\Sigma^e}\left[chase_{\Sigma^f}(T)\right].$$

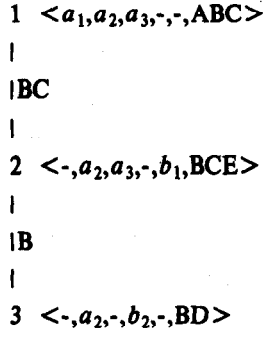
In other words, our reason for introducing ex-id's is that once an initial tableau has been completely chased under fd's then only the ex-id chase rule need be applied to the tableau to complete the chase.

4.2. THE CHASE AS A GRAPH

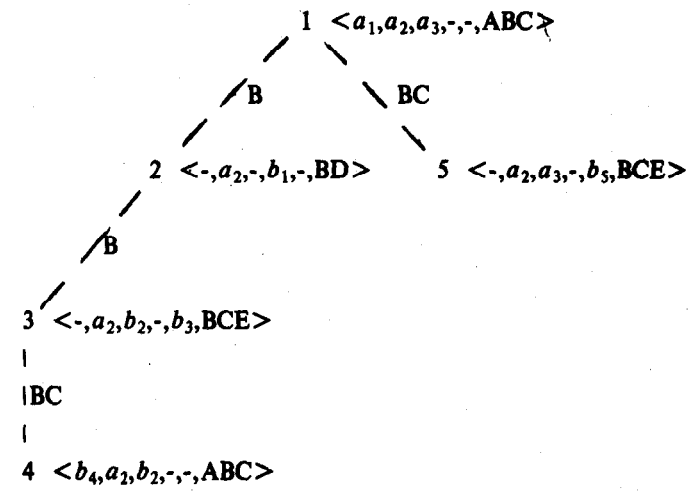
Consider some tableau T. One of the most helpful ways to view $chase_\Sigma(T)$ is as a labeled directed graph. In our graphical representation of T each tuple t in T, except the summary row, will be associated with a unique node with label t and there will be no edges initially. Rather than referring to the tag R of the label of a node n we shall say that the node n has tag R. Whenever some id $[R_i,R_j,S]$ is applied to a tuple w_i in T, thereby adding the tuple w_j to T, then add a node with label w_j to G and an edge between the nodes with labels w_i and w_j where the edge is labeled with the set S. Let G be the graph representing some completed chase. G is a set of trees.

Example 4: In Example 2 of this section two tableaux were presented. Both were members of $chase_\Sigma(T_{R_1})$ (where R_1 was a relation scheme in the database scheme H). Below are the two graphs corresponding to these tableaux.

Graph for T^1.



Graph for T^2.



■

The following Lemma illustrates the important role played by fd's in such graphs G.

Lemma 6: Let Σ be any set of fd's and id's. Let T be some finite tableau. Let $\hat{T}$ be any tableau in $chase_\Sigma(T)$. Let G be the graph for $\hat{T}$. Consider any two nodes in some tree in G. For any set of attributes $A \subseteq U$ the projection onto the set A of the labels of two nodes will be the same iff on the path between the the nodes the closure of each edge label contains A.

Proof: By induction on the number of dependencies applied. ■

Lemma 6 considers two nodes from any one tree in G. Although G is a forest, the property illustrated in the Lemma would obviously hold for any two nodes in G provided that there were appropriate edges between the root nodes of the trees.

4.3. TESTING FD-ACYCLICITY

In this section we use the results of the previous two sections to provide a characterization of fd-acyclicity.

It has been shown that if Σ is some set of fd's and id's and T some tableau then the chase of T under Σ can be defined, and although $chase_\Sigma(T)$ is not uniquely defined, any two tableaux in the set are equivalent.

Before these results can be used, however, we present a result shown in [L], extending work done by

Graham and Mendelzon [GM].

This result displays the relationship between equivalent expressions and tagged tableaux when the constraints of interest are fd's and id's. (The original result was proven for the more general case of equality generating dependencies, tuple generating dependencies, and typed id's.)

Theorem 7: $E \equiv_\Sigma E'$ iff $T_E^* \equiv T_{E'}^*$.
Proof: See [L]. ■

Using this theorem, Theorem 1 from this section can be expanded upon.

Theorem 8: Let Σ be a set of fd's together with the id's implied by pairwise-consistency. A database scheme H is fd-acyclic iff

$$\forall R \in H, \quad \forall D \in SAT(\Sigma), \quad T_R(T_D) = J_R(T_D),$$

iff

$$\forall R \in H, \quad T_R^* \equiv J_R^*,$$

iff

$$\forall R \in H, \quad T_R^* \subseteq J_R.$$

Proof: The first equivalence is Theorem 1 of this section. The second equivalence follows from Theorem 7.

To show the third equivalence notice that J_R represents the join taken over all of the edges. Thus, for each attribute A there is a unique distinguished variable v_A such that for any row $w \in J_R$ with tag R_1, if $A \in R_1$ then $w|_A = v_A$. So T_R can be embedded into J_R, and therefore into J_R^*, by a valuation which maps the summary of T_R to the summary of J_R^*. Thus there is a containment mapping of T_R into J_R^*. ■

The above Theorem provides a potential method of determining if a database scheme, H, is fd-acyclic; testing tableau containment for each relation scheme in H. The next result will show that containment only has to be tested for one of these relation schemes.

Theorem 9: If

$$\exists R \in H \text{ such that } \quad T_R^* \subseteq J_R$$

then

$$\forall R \in H, \quad T_R^* \subseteq J_R.$$

Proof: See [L]. ■

Corollary 10: A database scheme H is fd-acyclic iff there exists some relation scheme R in H for which there is a containment mapping of J_R into T_R^*.
Proof: By above. ■

Example 5: Let H be the database scheme previously considered in Example 1 of Section 2. Let the set of constraints be $\Sigma = \{B/C/D \rightarrow A\}$. J_{R_1} has been described in Example 1 of this section. We now present a subset of $T_{R_1}^*$.

A	B	C	D	Tag
a_1	a_2			
a_1	a_2			AB
a_1	a_2	b_1		BC
a_1		b_1	b_2	CD
a_1			b_2	AD

There is a trivial containment mapping of J_{R_1} into $T_{R_1}^*$. H is therefore fd-acyclic. ■

An immediate consequence of this result is that we need no longer distinguish between any of the edges in H, nor need we differentiate between distinguished and non-distinguished variables, except in recognizing the root tuple in T_R^*. Therefore the tableau J_R will be referred to as J.

5. TESTING TABLEAU CONTAINMENT

As was shown above a database scheme H is fd-acyclic iff J can be embedded into some T_R^* by a containment mapping. Since T_R^* contains an infinite number of rows though, there may not exist an effective procedure for testing this condition. This problem is a specific example of the following question, "If T_1 and T_2 are two tableaux which represent select, project, join expressions, and if Σ is a set of id's and fd's, does there exist an effective procedure to test if there is a containment mapping of T_1 into $chase_\Sigma(T_2)$?". In this section we will show the existence of such a procedure and present it.

D. S. Johnson and A. Klug [JK] have considered the related question of the existence of effective procedures to test tableau containment under a set of

"key-based" dependencies. Our problem differs from theirs in that they only considered key-based dependencies and they dealt with untyped id's and locally satisfying fd's. We however, are concerned with the chase under id's and weakly satisfying fd's or equivalently, as has been shown above, the chase under a set of ex-id's. As noted earlier the difference between an id and an ex-id is that in an id the attributes on which two adjacent nodes in the graph may agree must be contained in both of the associated edges (the tags of the rows). An examination of the proofs of the results of Johnson and Klug shows however that this property of id's was not used by them. Therefore the proofs, and the results, apply here.

Given two tableaux, T_1 and T_2, representing select, project, join expressions, Johnson and Klug showed that if there is a containment mapping from T_1 into $chase_\Sigma(T_2)$, then there will be one whose image is contained in the finite subset of $chase_\Sigma(T_2)$ where each node has level no greater than $|C||\Sigma|(N_\Sigma+1)^{N_\Sigma}$ (where C is the image of the containment mapping and N_Σ is the maximum number of attributes involved in an id in Σ). It is interesting to note that this bound can be improved on considerably when only typed id's (or ex-id's) are considered.

Johnson and Klug proved three Lemmas which they used to establish the result. We shall rephrase them in our context, including the tighter depth bound.

Let E be some select, project, join expression. Let T be the tableau associated with E. Let Σ be any set of ex-id's. We will follow the terminology of Johnson and Klug in so far as we shall not distinguish between a tableau, T, and the graph corresponding to T.

Lemma 1: Suppose c_1 and c_2 are rows in $chase_\Sigma(T)$ such that c_2 is in the subtree of c_1 (That is, in the tree which contains c_2 in $chase_\Sigma(T)$, the path from c_2 to the node of level zero contains c_1.) Let the length of the path from c_1 to c_2 be L. Suppose further that c_3 is a row in $chase_\Sigma(T)$ such that $Tag(c_1) = Tag(c_3)$. Suppose further that there is a containment mapping from $\{c_1\}$ to $chase_\Sigma(T)$ that sends c_1 to c_3. Then there is a containment mapping h of $\{c_1,c_2\}$ to $chase_\Sigma(T)$ such that $h(c_1) = c_3$ and

$$level(h(c_2)) \leqslant level(c_3) + L$$

Proof: See [JK]. ■

Lemma 2: Let c_1, c_2, and c_3 be as in Lemma 1. Let N_Σ be the maximum number of attributes involved on one side of an ex-id from Σ. Then there is a containment mapping, h, from $\{c_1,c_2\}$ to $chase_\Sigma(T)$ such that $h(c_1) = c_3$ and it satisfies

$$level(h(c_2)) \leqslant level(c_3) + |\Sigma| N_\Sigma.$$

Proof: Two rows, c and c', on the path from c_1 to c_2 will be said to be *equivalent* if, they were added to the tableau by chase-rules based on the same ex-id, $[R,R',I]$, and for every attribute $A \in I$, if $c_1[A] = c[A]$ then $c_1[A] = c'[A]$. The importance of equivalent rows is that if the path from c_1 to c_2, is longer than $|\Sigma| N_\Sigma$ then there must be two equivalent rows on it.

From this point on the proof is as in [JK]. ■

Lemma 3: If C is a set of rows in $chase_\Sigma(T)$ then there is a containment mapping of C to $chase_\Sigma(T)$ that preserves the summary row of $chase_\Sigma(T)$ and such that no row in the image has level exceeding

$$|C||\Sigma| N_\Sigma.$$

Proof: See [JK]. ■

Theorem 4: Let Σ be any set of fd's and typed id's. Let T_1 and T_2 be any two tableaux representing select, project, join expressions. If there is a containment mapping of T_1 into T_2, then there is a containment mapping of T_1 into the first $|T_1||\Sigma| N_\Sigma$ levels of T_2.

Proof: Let the assumed containment mapping of T_1 into T_2 be h_1. Assume that there is some row of level greater than $|T_1||\Sigma| N_\Sigma$ in $h_1(T_1)$. Let h_2 be the mapping which exists by Lemma 3. Then h_2 composed with h_1 is the desired containment mapping. ■

Corollary 5: There is an effective procedure to test for fd-acyclicity.

Proof: Choose some $R \in H$. Chase T_R to a depth of $|H||\Sigma| N_\Sigma$. Test to see if there is a containment mapping of J into this partially chased tableau. By Theorem 4, such a containment mapping will exist iff H is fd-acyclic. ■

Johnson and Klug were able to show that under key-based dependencies query containment is the same over finite or infinite states. They were primarily concerned with determining the complexity of testing tableau containment under this restrictive class of constraints. Based on their work we can show the corresponding result for an arbitrary set of constraints consisting of fd's and typed id's, however the result holds only over all countable states.

Theorem 6: Let Σ be a set of fd's and typed id's. The problem, "Given select, project, join expressions E, E', is $E \subseteq_\Sigma E'$?" is in NP.

Proof: By the above, $chase_\Sigma(T_E)$ need only be chased to a depth which is bounded by a polynomial which is a function of Σ. A nondeterministic polynomial time algorithm for testing tableaux containment is: 1) guess the image of $T_{E'}$; 2) guess enough of $chase_\Sigma(T_E)$ to show that the guessed image is in $chase_\Sigma(T_E)$; 3) verify that there is a containment mapping from E' to the guessed image. ■

Johnson and Klug were interested in conjunctive queries. Although we are concerned with select, project, join expressions we note that the results of this section extend to typed conjunctive queries.

6. CONCLUSION AND OPEN PROBLEMS

The notion of fd-acyclicity is an attempt to exploit the presence of functional dependencies to make cyclic schemes behave like acyclic ones.

As an application, consider query processing in a distributed database. Bernstein and Goodman [BG] have shown that semijoin tactics, which are useful in minimizing data transmission costs, may be applied successfully in the acyclic case. The property on which their results depend is that pairwise-consistency is equivalent to join-consistency and this is the property we use to define fd-acyclicity (Katsumo, [K], has taken a different approach to the same goal, based on an alternative characterization of acyclicity). If, on an fd-acyclic scheme, we can ensure that the data satisfies the fd's, this property will hold and thus the scheme will be amenable to the use of semijoin tactics for query processing.

It must be noted that the need to maintain satisfaction of the fd's after each update may entail additional data transmission costs. However, these costs are already present if we wish to support fd's. We might as well take advantage of this available information in query processing. Furthermore, research is being done on database schemes for which local satisfaction of the fd's guarantees satisfaction of the fd's as a whole[GY,CM]. It will be interesting to examine how this requirement interacts with fd-acyclicity.

It is intriguing to examine our results, and those of Johnson and Klug, in contrast to the limitations of the chase procedure pointed out by Goodman and Shmueli [GS2]. Their paper shows, among other things, that a finite chase-like procedure can not be used to determine whether pairwise-consistency implies join-consistency. This is because applying pairwise-consistency constraints to a tableau can be a non-terminating process. However, our work shows that the (possibly infinite) result of this process is well defined and that only a finite portion of the chase needs to be computed in order to test tableaux containment. This containment though is over all countable states. As a result we have shown that there is a chase-like procedure to test if pairwise-consistency implies join-consistency, although not over just the class of finite states.

Therefore the main problem this paper leaves open is whether the finite and infinite versions of fd-acyclicity are the same over arbitrary database schemes. This is an instance of a more general question, whether the notion of query equivalence under typed inclusion dependencies and fd's changes when we consider only finite database states.

In our work, and the work of Katsumo, two characterizations of acyclicity have been used as a basis for the proper definition of fd-acyclicity. It would be interesting to consider still others, for example the existence of a join forest under satisfying states, with the goal of determining which are equivalent.

7. References

[ASU] Aho, A., Sagiv, Y., Ullman, J. , "Equivalences Among Relational Expressions", SIAM J. Comp. 8,2 (1979), pp. 218-246.

[B+] Beeri, C., Fagin, R., Maier, D., Mendelzon, A.O., Ullman, J.D., Yannakakis, M., "Properties of Acyclic Database Schemes", Proc. 13th Ann. ACM Symp. Theory of Comp., (1981).

[BFMY] Beeri, C., Fagin, R., Maier, D., Yannakakis, M., "On The Desirable Properties of Acyclic Database Schemas", IBM Report RJ3131, San Jose, Calif., 1981.

[BG] Bernstein, P.A., Goodman, N., "The Power of Natural Semi-Joins", Siam J. of Comp., 10,4, (1981).

[CFP] Casanova, M.A., Fagin, R., Papadimitriou, C.H., "Inclusion Dependencies and Their Interaction with Functional Dependencies", Proc. First ACM SIGACT-SIGMOD Conf. on Principles of Database Systems (1982), pp. 171-176.

[CM] Chan, E., Mendelzon, A.O., "Separable and Independent Database Schemes", to appear, Second ACM SIGACT-SIGMOD Conf. on Principles of Database Systems, 1983.

[F1] Fagin, R., "Horn Clauses and Data Dependencies", JACM 29, 4 (1982), pp. 952-985.

[F2] Fagin, R., "Multivalued Dependencies and a New Normal Norm for Relational Databases", ACM Trans. Database Syst., 2,3 (1977), pp. 262-278.

[FMU] Fagin, R., Mendelzon, A.O., Ullman, J.D., "A Simplified Universal Relation Assumption and its Properties", ACM Trans. Database Syst., 7,3 (1982), pp. 343-360.

[G] Graham, M.H., "Satisfying Database States", Ph.D. thesis, University of Toronto, Dept. of Computer Science, 1981.

[GM] Graham, M., Mendelzon, A.O., "Strong Equivalence of Relational Expressions Under Dependencies", Infor. Proc. Letters, 14,2, (1982), pp. 57-62.

[GS] Goodman, N., Shmueli, O., "Tree Queries: A Simple Class of Relational Queries", ACM Trans. Database Syst., 7,4 (1982), pp. 653-677.

[GS2] Goodman, N., Shmueli, O., "Limitations of the Chase", Infor. Proc. Letters, 13,4 (1982), pp. 154-157.

[GY] Graham, M., Yannakakis, M., "Independent Database Schemes", Proc. First ACM SIGACT-SIGMOD Conf. on Principles of Database Systems (1982), pp. 199-205.

[H] Honeyman, P., "Testing Satisfaction of Functional Dependencies", JACM 29,3 (1982), pp. 668-677.

[JK] Johnson, D.S., Klug,A., "Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies", Proc. First ACM SIGACT-SIGMOD Conf. on Principles of Database Systems (1982), pp. 164-169.

[K] Katsumo, H., "A Desirable Class of Sets Of FD's and MVD's", NTT Technical Report, Japan, (1982).

[L] Laver, K., Forthcoming Ph.D dissertation, Univ. of Toronto.

[MMS] Maier, D., Mendelzon, A.O., Sagiv, Y., "Testing Implications of Data Dependencies", ACM Trans. Database Syst., 4,4 (1979), pp. 455-469.

[S] Sagiv, Y., "Can we use the universal instance assumption without using nulls?", Proc. ACM-SIGMOD 1981 International Conf. on Management of Data (1981), pp. 108-120.

[U] Ullman, J.D., "Principles of Database Systems", Computer Science Press, 2nd ed., (1982).

[V] Vassiliou, Y., "A Formal Treatment of Imperfect Information in Database Management", Ph.D thesis, Univ. of Toronto, (1980).