ON THE COMPLEXITY OF TESTING IMPLICATIONS OF DATA DEPENDENCIES

by

C. Beeri and M.Y. Vardi*

Department of Computer Science

The Hebrew University of Jerusalem

Jerusalem, Israel

December 1980

---

* Research partially supported by Grant 1849/79 of the U.S.A.-Israel Binational Science Foundation

ABSTRACT

We show that a join dependency is implied by a set D of total tuple generating dependencies and functional dependencies if and only if it is implied by the set D^*, which is the result of replacing each functional dependency X->A in D by the multivalued dependency X->->A. It follows that testing whether a join dependency is implied by a join dependency and several multivalued dependencies is NP-hard. Using one basic reduction from the exact cover problem, we prove several NP-hardness results. E.g., testing whether a tuple generating dependency is implied by a join dependency, testing whether a join dependency is implied by a join dependency and a functional or multivalued dependency, testing whether a join dependency is implied by a total tuple generating dependency, and testing whether an equality generating dependency is implied by an equality generating dependency, are all NP-hard. We also show that testing whether a relation violates an equality generating dependency, and testing whether a general tuple generating dependency is trivial are NP-complete.

1. INTRODUCTION

One of the important issues in the design of relational database schemas is the specification of the constraints that the data must satisfy to model correctly the part of the world under consideration. These constraints determine which databases are considered meaningful.

Of particular interest are the constraints called _data dependencies_. The first dependencies to be studied were the _functional dependencies_ [Codd], which were followed by the _multivalued dependencies_ [Fag1,Zan]. Recently, a number of generalizations of these dependencies have appeared, culminating with the _tuple generating_ and _equality generating_ dependencies of [BV2,BV3][(1)]. Intuitively, the meaning of a dependency is that if some tuples, fulfilling certain conditions, exist in the database, then either some other tuples must also exist in the database, or some values in the given tuples must be equal.

A utilization of the above dependencies in the design of a relational database requires an algorithm for solving the _implication problem_, i.e., to decide whether a given set of dependencies logically implies another dependency. It is not known whether the problem is solvable for general tuple generating dependencies, however, if all dependencies in the given set are total, then the problem is solvable. A decision procedure for this case - the "chase" - is described in

---

(1) Similiar generalizations were done independently by [Fag2,JP,Pa,YP,SU].

[BV3]. This procedure has an exponential worst case runing time.

The results of [MSY] suggests that there is probably no efficient implication testing procedure. They have shown that testing whether a join dependency is implied by a join dependency and several functional dependencies is NP-complete. In this paper we extend their results, and show that testing implications is NP-hard even for very restricted cases.

The outline of the paper is as follows. In Section 2 we define the relational model, tableaux and dependencies. The "chase" is described as an implication testing procedure. In Section 3 we prove that a join dependency is implied by a set D of total tuple generating dependencies and functional dependencies if and only if it implied by the set D^*, which is the result of replacing each functional dependency X->A in D by the multivalued dependency X->->A. From the above mentioned result of [MSY] it immediately follows that testing whether a join dependency is implied by a join dependency and several multivalued dependencies is NP-hard.

The EXACT COVER problem of [Ka] is the base of the NP-hardness results of Section 4. After describing the basic reduction in Subsection 4.1, we present a list of NP-hard problems in Subsection 4.2. Specifically, testing whether:

a) a tuple generating dependency is implied by a join dependency,

b) a join dependency is implied by a join dependency and a functional dependency,

c) a join dependency is implied by a join dependency and a multivalued dependency,

d) a join dependency is implied by a total tuple generating dependency,

e) an equality generating dependency is implied by an equality generating dependency,

f) a join dependency is implied by an equality generating dependency, or

g) an equality generating dependency is implied by a join dependency and a functional dependency,

is NP-hard. In fact, apart from cases (c) and (d) all other cases are NP-complete, since testing whether a dependency is implied by a join dependency and several equality generating dependencies is in NP.

The same basic reduction enables us to show in Subsection 4.3 that testing whether a relation violates an equality generating dependency, and testing containment of tableaux is NP-complete. We conclude in Section 5 by showing that for a more general type of tuple generating dependency even testing triviality is NP-complete.

2. BASIC DEFINITIONS

2.1 Attributes and Relations

Attributes are symbols taken from a given finite set $U=\{A_1,\ldots,A_n\}$ called the universe. All sets of attributes are subsets of U. We use

the letters A,B,C,... to denote single attributes, and X,Y,... to denote sets of attributes. We do not distinguish between the attribute A and the set {A}. The union of X and Y is denoted by XY, and the complement of X in U is denoted by $\overline{X}$.

With each attribute A is associated an infinite set, called its domain, denoted DOM(A), such that DOM(A) $\cap$ DOM(B) = $\emptyset$ for A $\neq$ B. Let Dom = DOM(A_1) $\cup$ ... $\cup$ DOM(A_n). For a set X, an X-value is a mapping w: X -> Dom, such that w(A) $\in$ DOM(A) for all A $\in$ X. A tuple is a U-value. A relation is a finite set of tuples. We use the letters p,q,r,... to denote tuples, and I,J,... to denote relations.

For a tuple w and a set Y $\subseteq$ U, we denote the restriction of w to Y by w[Y]. We do not distinguish between w[A], which is an A-value, and w(A), which is an element of DOM(A). Let I be a relation. The set of X-values in I is I[X] = {w[X] | w $\in$ I}.

2.2 Tableaux

A valuation is a mapping h: Dom -> Dom, such that a $\in$ DOM(A) implies h(a) $\in$ DOM(A) for all a $\in$ Dom. h can be extended to tuples and relations as follows. Let w be an tuple, then h(w) = h$\cdot$w ($\cdot$ denotes composition). Let I be a relation, then h(I) = {h(w) | w $\in$ I}. Usually, we are interested only in a small subset of Dom, e.g., the set of values in a relation I. We let h be undefined for other values, and say that h is a valuation on I.

Let I be a relation, and let h be a valuation on I. An extension of h to another relation I`,is a valuation h` on I`, which agree with h on I. If, for all A $\in$ U and for all a $\in$ I[A], we have h(a) = a, then h is the identity on I.

A tableau [ASU] is a pair T = <w,I>, where w is an tuple and I is a finite relation, such that w[A] $\in$ I[A] for all A $\in$ U. T defines an operation on relations as follows: T(J) = {h(w) | h is a valuation s.t. h(I) $\subseteq$ J}. I.e., T(J) is the set of images of w under all valuations that map every tuple of I to some tuple of J. Observe that I $\subseteq$ T(I).

Let T_1 and T_2 be tableaux. We say that T_1 is contained in T_2, denoted $T_1 \leq T_2$, if $T_1(I) \subseteq T_2(I)$ for all relations I. T_1 and T_2 are equivalent, denoted $T_1 \equiv T_2$, if both $T_1 \leq T_2$ and $T_2 \leq T_1$. A necessary and sufficient condition for containment was proved by [ASU].

Lemma 2.1. <v,J> $\leq$ <u,I> iff there exists a valuation h on I, such that h(I) $\subseteq$ J and h(u) = v. <>

The values in a tableau serve as variables, and can always be consistently renamed.

Lemma 2.2. [ASU] Let <u,I> be a tableau, and let h be a one-to-one valuation on I, then <u,I> $\equiv$ <h(u),h(I)>. <>

2.3 Dependencies

For any given application, only a subset of all possible relations is

of interest. This subset is defined by constraints which are to be satisfied by the relations of interest. A class of constraints that was extensively studied is the class of dependencies.

An equality generating dependency (abbr. egd) says that if some tuples, fulfilling certain conditions, exist in the database, then some values in these tuples must be equal. Formally, an egd is a pair $\langle(a_1,a_2),I\rangle$, where a_1 and a_2 are A-values for some attribute A, and I is a relation, such that $a_1,a_2 \in I[A]$. A relation J satisfies $\langle(a_1,a_2),I\rangle$ if for any valuation h, such that $h(I) \subseteq J$, we have $h(a_1) = h(a_2)$.

A special case of egd's is the functional dependency (abbr. fd) [Codd]. An fd is an egd $\langle(a_1,a_2),I\rangle$, where $|I| = 2$ and $\{a_1,a_2\} = I[A]$. Usually, it is written X->A, where $X = \{B : |I[B]| = 1\}$. Thus a relation J satisfies X->A if for all tuples $u,v \in J$, if $u[X] = v[X]$ then $u[A] = v[A]$.

A tuple generating dependency (abbr. tgd) says that if some tuples, fulfilling certain conditions, exist in the database, then another tuple must also exist in the database. Formally, a tgd is a pair $\langle w,I\rangle$, where w is a tuple and I is a relation. A relation J satisfies $\langle w,I\rangle$ if for every valuation h, such that $h(I) \subseteq J$, there exists an extension h` of h to w, such that $h`(w) \in J$.

The attribute set of $\langle w,I\rangle$ is the set of "known" attributes in w, $ATTR(\langle w,I\rangle) = \{A \mid w[A] \in I[A]\}$. $\langle w,I\rangle$ is a total tgd (abbr. ttgd) if

it is also a tableau, i.e., if ATTR(<w,I>) = U.

A sepcial case of ttgd's is the join dependency (abbr. jd) [ABU,Riss]. A jd is a ttgd <w,I> such that for every tuple u ∈ I, if u[A] ≠ w[A], then u[A] has a unique occurence in I[2].

Lemma 2.3. [BMSU,Va2] Let T = <w,I> be a jd, and let J be a relation, then T(T(J)) = T(J)[3]. <>

Another special case of ttgd's is the multivalued dependency (abbr. mvd) [Fagl,Zan]. An mvd is a ttgd <w,I>, where |I| = 2. Usually, it is written as X->->Y, where X = {A : |I[A]| = 1} and Y = {A | w[A] = u[A]} for some u ∈ I. Thus, a relation J satisfies X->->Y if for all tuples u,v ∈ J, we have that if u[X] = v[X] then there exists a tuple t ∈ J such that t[XY] = u[XY] and t[Z] = v[Z], where Z = $\overline{XY}$. Note that an mvd is also a jd.

The values in a dependency serve as variables, thus they can be consistently renamed. The proof of the following lemma is left to the reader.

Lemma 2.4. Let h be a one-to-one valuation on I u {w}, and let J be a relation, then:

a) J satisfies <w,I> if and only if it satisfies <h(w),h(I)>.

b) J satisfies $<(a_1,a_2),I>$ if and only if it satisfies $<(h(a_1),h(a_2)),h(I)>$. <>

---

(2) The customary definition of jd's, e.g., in [Riss], is different from ours but is equivalent, as shown in [BV1,Va2].

(3) Note that T is both a dependency and a tablueax.

Example 1.1. Let U = {A,B,C,D}, DOM(A) = {a0,a1,...}, DOM(B) = {b0,b1,...}, etc.. Let I,J be the relations:

I:

A	B	C	D
a0	b0	c1	d0
a0	b1	c0	d1

J:

A	B	C	D
a0	b0	c0	d0
a1	b0	c0	d1

Let u be the tuple:

A	B	C	D
a0	b0	c0	d0

Let d_1 be the ttgd <u,I>. d_1 is equivalent to the mvd A->->C. Let d_2 be the egd <(a0,a1),J>. d_2 is equivalent to the fd BC->A. <>

In the sequel we use D to denote finite sets of ttgd's and egd's, and we use d to denote single dependencies.

2.4 Testing Implications of Dependencies

For a set of dependencies D we denote by SAT(D) the set of relations that satisfy all dependencies in D. D implies a dependency d, denoted D |= d, if SAT(D) $\subseteq$ SAT(d). That is, if d is satisfied by every relation which satisfies all dependencies in D. The implication problem is to decide for a given set of dependencies D and a dependency d whether D |= d. It is not known whether the problem is solvable, however, if all tgd's in D are total, as we assume is the case from now on, then the problem is solvable [BV2,BV3]. A decision procedure for this case - the "chase" - was developed in [BV3] generalizing [MMS].

Intuitively, to test whether D implies $\langle w,I\rangle$ (or $\langle(a_1,a_2),I\rangle$), we "chase" I by D into SAT(D), and then check if w is in I (or if a_1 and a_2 are identical in I). A chase of I by D is a sequence of relations $I_0,I_1,\ldots$ such that $I = I_0$ and I_{n+1} is obtained from I_n by an application of a chase rule. To each dependency in D there corresponds a chase rule; T-rules correspond to ttgd's and E-rules correspond to egd's.

T-rule (for a ttgd $\langle w,J\rangle$ in D): For some valuation h on J such that $h(J) \subseteq I_n$, I_{n+1} is $I_n \cup \{h(w)\}$.

E-rule (for a egd $\langle(a_1,a_2),J\rangle$ in D): Initially I_{n+1} is I_n. Now for some valuation h on J such that $h(J) \subseteq I_n$, identify $h(a_1)$ and $h(a_2)$ in I_{n+1}.

To make the E-rule nonambiguous, we assume that for all attributes $A \in U$, DOM(A) is totally ordered, and whenever two values are identified, the greater is identified with the smaller. Given $\langle w,I\rangle$, we take w(A) as the smallest value in DOM(A), for all $A \in U$. (We can always rename the values in w and I such that this is true). Similiarly, given $\langle(a_1,a_2),I\rangle$, we take a_1,a_2 as the smallest values in DOM(A) and $a_1 \leq a_2$. Thus, the values in w or a_1 do not change in the chase and a_2 can be identified only with a_1.

If I_n is not effected by a chase rule, i.e., if $I_n = I_{n+1}$, then we say that the rule is not applicable to I_n. We also say "apply a dependency" instead of "apply a chase rule for a dependency". A chase is finite if it is a finite sequence $I_0,\ldots,I_n$, and no dependency in D is applicable to I_n.

We describe now two special cases of the rules.

T-rule (for an mvd X->->Y in D): For some tuples u,v ∈ I such that u[X] = v[X], I_{n+1} is $I_n \cup \{t\}$, where t is defined by t[XY] = u[XY] and t[Z] = v[Z], where Z = $\overline{XY}$.

E-rule (for an fd X->A in D): Initially I_{n+1} is I_n. Now, for some tuples u,v ∈ I such that u[X] = v[X], identify u[A] and v[A] in I_{n+1}.

Actually, in [BV3] we gave a stonger T-rule.

T`-rule (for a ttgd <w,J> in D): $I_{n+1} = \langle w,J\rangle(I_n)$.

However, observe that a T`-rule application is equivalent to a sequence of T-rules application.

Since we do not specify an order for the application of the rules, many chases are possible. However:

Lemma 2.5. [BV3] All chases of I are finite, and has the same last relation, which is SAT(D). <>

Let $chase_D(I)$ denote the unique last relation in a chase of I by D.

Theorem 2.1. [BV3]

a) D |= <w,I> if and only if w[ATTR(<w,I>)] ∈ $chase_D(I)$[ATTR(<w,I>)].

b) D |= <(a_1,a_2),I> if and only if $a_2 \notin chase_D(I)$. <>

Example 2.1. Let U = {A,B,C,D}, DOM(A) = {a0,a1,...}, DOM(B) = {b0,b1,...}, etc.. Let I,J,K be the relations:

I:

A	B	C	D
a0	b0	c1	d0
a0	b1	c0	d1

J:

A	B	C	D
a0	b0	c0	d0
a1	b0	c0	d1

K:

A	B	C	D
a0	b0	c1	d1
a0	b1	c0	d2
a1	b0	c0	d0

Let u be the tuple:

A	B	C	D
a0	b0	c0	d0

Let d_1 be <u,I>. d_1 is the mvd A->->C. Let d_2 be <(a0,a1),J>. d_2 is the fd BC->A. Let D be $\{d_1,d_2\}$. Let d be <u,K>. d is a jd. We use the chase to show that D |= d. We start by applying d_1 to K, getting K_1:

	A	B	C	D
	a0	b0	c1	d1
	a0	b1	c0	d2
	a1	b0	c0	d0
new tuple:	a0	b0	c0	d1
new tuple:	a0	b1	c1	d2

Now we apply d_2 to K_1, getting K_2:

	A	B	C	D
	a0	b0	c1	d1
	a0	b1	c0	d2
new tuple:	a0	b0	c0	d0
	a0	b0	c0	d1
	a0	b1	c1	d2

The new tuple is exactly u. Thus u $\in$ $chase_D(K)$ and D |= <u,K>. <>

The chase has an exponential worst case running time. The following result of [MSY] indicate that there is probably no efficient implication testing procedure.

Theorem 2.2. Testing whether a jd is implied by a jd and several fd's

is NP-complete. <>

3. TESTING IMPLICATION OF JOIN DEPENDENCIES

Let D be a set of fd's anf ttgd's. Denote by D^* the result of replacing each fd X->A in D by the mvd X->->A.

Theorem 3.1. Let d be a jd, D |= d iff D^* |= d. <>

Corollary. Testing whether a jd is implied by one jd and several mvd's is NP-hard.

Proof. The claim follows from Theorem 2.2 by the above theorem. <>

In the next section we show that testing whether a jd is implied by a jd and an fd or an mvd is also NP-hard.

Theorem 3.1 will be proved in two steps. First we introduce a modified E-rule for fd's, and show that the new chase works as a test for implications of jd's. Then we show that in the new chase fd's can be replaced by mvd's.

F-rule: (for an fd X->A in D): Initially I_{n+1} is I_n. Now for some tuples u,v $\in$ I, such that u[X] = v[X], identify u[A] and v[A] in u and v in I_{n+1}.

The difference between the E-rule and the F-rule is that the E-rule identifies all occurences of the identified values, while the F-rule identifies only a specific occurence of these values. Identification is controled by the order relation defined on the underlying domain for both the E-rule and the F-rule.

define a chase` of I by D in a manner analogous to the definition of the chase in Section 2.4, with F-rules replacing the E-rules. Since we always identify the greater value with the smaller one, the finiteness of the chase` is assured. Let chase`$_D$(I) the last relation in a chase` of I by D[4].

Lemma 3.1 Let d = <w,I> be a jd, D |= d iff w ∈ chase`$_D$(I).

Proof.

(if) Suppose that w ∈ chase`$_D$(I). Let J be a relation in SAT(D), and let h be a valuation on I such that h(I) ⊆ J. Since the new chase does not introduce new values, h is a valuation on chase`$_D$(I). If we show that h(chase`$_D$(I)) ⊆ J, then we are done since w ∈ chase`$_D$(I) entails h(w) ∈ J. We show that h(chase`$_D$(I)) ⊆ J by induction on the length of the chase`.

Initially, h(I) ⊆ J. Assume now that h(I_{n-1}) ⊆ J. The n-th rule is either a T-rule or an F-rule. If it is a T-rule for a ttgd <u,K>, then I_n is I_{n-1} ∪ {g(u)}, for some valuation g such that g(K) ⊆ I_{n-1}. But then we have h•g(K) ⊆ J, and since J ∈ SAT(D), also h•g(u) ∈ J. That is, h(I_n) ⊆ J. If the n-th rule is an F-rule for an fd X->A, then I_n is obtained by identifying u[A] with v[A], for tuples u,v ∈ I_{n-1} such that u[X] = v[X]. But then h(u)[X] = h(v)[X], and since J ∈ SAT(D), h(u)[A] = h(v)[A]. That is, h(I_n) ⊆ J.

(only if) Suppose that w ∉ chase`$_D$(I). Clearly, chase`$_D$(I) ∈SAT(D), otherwise we can apply the appropriate rule for the violated dependency. Consider now the effect of the rules. In every rule

---

[4] We do not claim uniqueness.

application every value occurence a_i is replaced by another value a_j, with the possibility that i = j. This defines a correspondence between value occurences. Similiarly, we can define a correspondence between tuples. Taking this correspondence to its reflexive-transitive closure, we see that to every value occurence in I there corresponds a value in $chase`_D(I)$, and similiarly for tuples. Since the values in w are never modified, they correspond to themselves. Let h be a valuation, taking each value occurence to its corresponding value in $chase`_D(I)$. h is well defined, since it is the identity on w, and all other value occurences are distinct. Because of the correspondence between tuples, we also have $h(I) \subseteq chase`_D(I)$. But $h(w) = w \notin chase`_D(I)$, hence $chase`_D(I) \notin SAT(D)$. <>

Remark. Using argumnets similiar to that of [BV3,MMS] we can prove a stronger result: If <w,I> is a jd, then $chase`_D(I) = chase_D(I)$. Lemma 3.1 straightforwardly follows. <>

Lemma 3.2. $chase`_D(I) \subseteq chase`_D{}^*(I)$.

Proof. Let $I_0,\ldots,I_n$ be a chase` of I by D. We describe a chase` of I by D^*: $I`_o,\ldots,I`_n,\ldots,I`_m$, such that $I_j \subseteq I`_j$, for $1 \leq j \leq n$. $I`_0$ is $I_0$. Assume that $I_j \subseteq I`_j$. If I_{j+1} is obtained by applying a tgd <u,K> to I_j, then $I_{j+1} = I_j \cup h(u)$, for some valuation h such that $h(K) \subseteq I_j$. But then also $h(K) \subseteq I`_j$, and we can apply <u,K> to $I`_j$ and have $I`_{j+1} = I`_j \cup h(u)$, so $I_{j+1} \subseteq I`_{j+1}$. If $I_{j+1}$ is obtained by applying an fd X->A to $I_j$, then for some tuples $u,v \in I_j$, u[X] = v[X], and $I_{j+1} = I_j \cup \{w\} - \{v\}$, where w is defined by w[A] = u[A] and $w[\bar{A}] = v[\bar{A}]$. Since u[X] = v[X], we can apply X->->A to $I`_j$ and have

$I`_{j+1} = I`_j \cup \{w\}$, so $I_{j+1} \subseteq I`_{j+1}$. Finally, $I`_{n+1},\ldots,I`_m$ are obtained by applying ttgd's in $D^*$; thus, $I_n \subseteq I`_j$, for $n < j \leq m$. Hence, $chase`_D(I) \subseteq chase`_{D^*}(I)$. <>

Proof of Theorem 3.1.

(if) Since X->A |= X->->A [BFH], we have $D \models D^*$. The claim follows.

(only if) Let d be <w,I>. By Lemma 3.1, $w \in chase`_D(I)$. By Lemma 3.2, $w \in chase`_{D^*}(I)$. Thus, Lemma 3.1 entails that $D^* \models d$. <>

4. LOWER BOUNDS FOR IMPLICATION TESTING

By Theorem 3.1, testing implications of tgd's is NP-hard. In this section we show that testing implication of tgd's and egd's is NP-hard, even if D consist of one or two dependencies. In proof we reduce the NP-complete problem of EXACT COVER [Ka,GJ] to the implication problem. The reduction technique is similiar to that used in [SY] and [MSY].

4.1 The Reduction

An instance of EXACT COVER is a set $X = \{X_1,\ldots,X_n\}$ and a collection of proper subsets of X, $C = \{S_1,\ldots,S_m\}$, where $|S_i| = 3$, for $1 \leq i \leq m$. C contains an exact cover if there is a subset of C, $C` = \{S_{i_1},\ldots,S_{i_p}\}$, $p > 1$, which is a partition of X. Without loss of generality we can assume that $i \neq j$ implies $S_i \neq S_j$. We construct two relations that represent this instance. The universe has

m+n+1 attributes, $U = \{X_1,\ldots,X_n,S_1,\ldots,S_m,A\}$. For domains we take $DOM(B) = \{B\} \times \mathbf{N}$, for all attributes $B \in U$. E.g., $\langle X_1,1\rangle \in DOM(X_1)$ and $\langle A,2\rangle \in DOM(A)$. For clarity we usually omit the first element of the pair. Thus, $w[A] = 1$ actually means $w[A] = \langle A,1\rangle$.

The first relation I represents the n elements in X. It has n+1 tuples $r_1,\ldots,r_n,s$, where r_i corresponds to X_i. Let X_i belong to the sets $S_{i_1},\ldots,S_{i_k}$, $k \geq 1$, then $r_i[X_i] = 1$, $r_i[S_{i_j}] = 1$, $1 \leq j \leq k$, and $r_i[A] = 1$. For the last tuple s we have $s[X_i] = 1$, $1 \leq i \leq n$, $s[S_j] = 2$, $1 \leq j \leq m$, and $s[A] = 2$. All other values in I are distinct, i.e., each other value has a unique occurence in I.

The second relation J represents the m sets in C. It has m+1 tuples $u_1,\ldots,u_m,v$, where u_j corresponds to S_j. Let S_j be $\{X_{j_1},\ldots,X_{j_k}\}$, $k \geq 1$, then $u_j[X_{j_i}] = 1$, $1 \leq i \leq k$, $u_j[S_j] = 1$, $u_j[S_i] = 2$, $1 \leq i \leq m$ and $i \neq j$, and $u_j[A] = 1$. For the last tuple v we have $v[X_i] = 1$, $1 \leq i \leq n$, $v[S_j] = 2$, $1 \leq j \leq m$, and $v[A] = 2$. All other values in J are distinct.

Example 4.1. Let $X = \{X_1,X_2,X_3,X_4\}$ and $C = \{S_1,S_2,S_3\}$, where $S_1 = \{X_1,X_2,X_3\}$, $S_2 = \{X_1,X_3,X_4\}$ and $S_3 = \{X_1,X_2,X_4\}$. The relation I is described in Figure 1, and the relation J is described in Figure 2. The dots stand for distinct values. <>

Lemma 4.1. Let h be a valuation on I such that $h(I) \subseteq J$ and $|h(I)| > 1$, then $h(s) = v$, $h(I-\{s\}) \subseteq J-\{v\}$, and $h(r_i) = u_j$ only if $i \in S_j$, for all r_i.

	X_1	X_2	X_3	X_4	S_1	S_2	S_3	A
r_1	1	.	.	.	1	1	1	1
r_2	.	1	.	.	1	.	1	1
r_3	.	.	1	.	1	1	.	1
r_4	.	.	.	1	.	1	1	1
s	1	1	1	1	2	2	2	2

Figure 1

	X_1	X_2	X_3	X_4	S_1	S_2	S_3	A
u_1	1	1	1	.	1	2	2	1
u_2	1	.	1	1	2	1	2	1
u_3	1	1	.	1	2	2	1	1
v	1	1	1	1	2	2	2	2

Figure 2

Proof. Let h be a valuation on I such that $h(I) \subseteq J$ and $|h(I)| > 1$.

Claim 1. $h(I-\{s\}) \subseteq J-\{v\}$.

Assume that $h(r_i) = v$, for some r_i. Since $r_i[A] = r_k[A]$, for all r_k, and $v[A] = 2$ is a distinct value, it must be the case that $h(I-\{s\}) = v$. Since $r_k[X_k] = s[X_k]$ and $h(r_k)[X_k] = 1$, for all r_k, it must be the case that $h(s) = v$, because if $h(s) = u_j$ then $S_j = X$. Thus, $h(I) = v$ - contradicting the suposition that $|h(I)| > 1$.

Claim 2. If $X_k, X_l \in S_j$ and $h(r_k) = u_j$, then also $h(r_l) = u_j$.

If $X_k, X_l \in S_j$, then $r_k[S_j] = r_l[S_j]$. But $u_j[S_j]$ is a distinct value; hence $h(r_k) = u_j$ implies $h(r_l) = u_j$.

Claim 3. $h(s) = v$.

Assume that $h(s) = u_j$, for some u_j. For all $X_i \in X-S_j$, $s[X_i] = r_i[X_i]$ and $u_j[X_i]$ is a distinct value, hence, $h(r_i) = u_j$. Let $X_i \in S_j$ and $h(r_i) = u_k$. Since $r_i[X_i] = s[X_i]$ and $h(s)[X_i] = u_j[X_i] = 1$, it must be the case that $h(r_i)[X_i] = u_k[X_i] = 1$, that is, $X_i \in S_k$. By Claim 2,

for all $X_1 \in S_k$, $h(r_1) = u_k$. If $k = j$, then $h(I) = u_j$ - contradicting the supposition that $|h(I)| > 1$. Thus, $k \neq j$, and it must be the case that $S_k \subseteq S_j$, because for all $X_1 \in X\text{-}S_j$ we have $h(r_1) = u_j$, contradicting the assumption that $S_i \neq S_j$ and $|S_i| = |S_j| = 3$.

Claim 4. $h(r_i) = u_j$ only if $X_i \in S_j$. Assume that $X_i \notin S_j$. Since $r_i[X_i] = s[X_i]$ and $u_j[X_i]$ is a distinct value, it must be the case that $h(s) = u_j$ - contradicting Claim 3. <>

Lemma 4.2. Let h be a valuation on I such that $h(I) \subseteq J$, then $|h(I)| > 1$ if and only if C contains an exact cover of X.

Proof.

(only if) Let h be a valuation on I such that $h(I) \subseteq J$, and $|h(I)| > 1$. We claim that the subcollection $C' = \{S_j \mid u_j = h(r_i)$ for some $r_i\}$ is an exact cover of X. C' is a cover of X because, by Lemma 5.2, $h(I\text{-}\{s\}) \subseteq J\text{-}\{v\}$, and $h(r_i) = u_j$ only if $X_i \in S_j$. C' is exact because, by Claim 2 in the proof of Lemma 5.2, if $X_k, X_l \in S_j$ and $h(r_k) = u_j$, then also $h(r_l) = u_j$. Thus, if for $S_{j_1}, S_{j_2} \in C'$ we have $X_i \in S_{j_1} \cap S_{j_2}$, then $h(r_i) = u_{j_1}$ and $h(r_i) = u_{j_2}$, implying $j_1 = j_2$.

(if) Let $C' = \{S_{j_1}, .., S_{j_p}\}$, $p > 1$, be an exact cover of X, i.e., for each $X_i \in X$, there is a unique $S_{k_i} \in C'$ such that $X_i \in S_{k_i}$. Define a valuation h on I as follows. $h(\langle X_i,1\rangle) = \langle X_i,1\rangle$, $1 \leq i \leq n$, $h(\langle S_j,1\rangle) = \langle S_j,1\rangle$ if $S_j \in C'$, $h(\langle S_j,1\rangle) = \langle S_j,2\rangle$ if $S_j \notin C'$, $h(\langle S_j,2\rangle) = \langle S_j,2\rangle$, $1 \leq j \leq m$, $h(\langle A,1\rangle) = \langle A,1\rangle$ and $h(\langle A,2\rangle) = \langle A,2\rangle$. Clearly, we can extend h to the other (distinct) values in I so that $h(s) = v$ and $h(r_i) = u_{k_i}$, $1 \leq i \leq n$. <>

4.2 NP-hardness Results

Most of the NP-hardness results in this subsection are actually NP-completeness results due to the following lemma, which generalizes a result of [MSY].

Lemma 4.3. Testing whether a tgd or an egd is implied by a jd and several egd's is in NP.

Proof. Let d be <*,I> (* is either a tuple or a pair of values), and let D be a set consisting of one jd <u,J> and several egd's. Let n be the number of distinct values in I, and let m be $\max\{k \mid \langle a_1,a_2),K\rangle \in D \text{ and } |K| = k\}$.

To compute $\text{chase}_D(I)$ we execute the following until no rule is applicable:

a) apply a T`-rule for <u,J>, and

b) apply E-rules until no egd is applicable.

By Lemma 2.3, each time step (b) is executed (except for the last time) the number of distinct values in I is decreased at least by one. Thus, step (a) and (b) together are executed at most n times. In order for step (b) to decrease the number of values in I, it suffices that step (a) add some (at most) k tuples. Thus, by guessing the "right" applications, we can compute $\text{chase}_D(I)$ by executing the following at most n times:

a`) apply k T-rules for <u,J>, and

b`) apply an E-rule.

Thus, the computation requires at most n(k+1) rules applications. Each rule application can be done in polynomial time after the "right"

valuation is guessed. It follows that $chase_D(I)$ can be computed in nondeterministic polynomial time. The claim follows. <>

We use now the lemmas of the preceeding subsection to prove NP-hardness for several cases.

Theorem 4.1. Testing whether a tgd is implied by a jd is NP-complete.

Proof.

In NP: by Lemma 4.3.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct relations I and J as described above. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 1$, $1 \leq j \leq m$, and $p[A] = 1$. $\langle p,I \rangle$ is a jd. Let q be a tuple: $q[X_i] = 1$, $1 \leq i \leq n$, $q[S_j] = 3$, $1 \leq j \leq m$, and $q[A] = 1$. $\langle q,J \rangle$ is a tgd and $ATTR(\langle q,J \rangle) = XA$. C contains an exact cover of X iff $\langle p,I \rangle \models \langle q,J \rangle$: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$ and $h(p)[XA] = q[XA]$. Thus, $h(p) \in chase_{\{\langle p,I \rangle\}}(J)$ and $\langle p,I \rangle \models \langle q,J \rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Thus, $chase_{\{\langle p,I \rangle\}}(J) = J$ and $\langle p,I \rangle \not\models \langle q,J \rangle$. <>

We can not strenghen Theorem 4.1 to implications of ttgd's.

Theorem 4.2. Testing whether a ttgd is implied by a jd can be done in polynomial time.

Proof. Let $\langle p,I \rangle$ be a jd, and let $\langle q,J \rangle$ be a ttgd. By Lemma 2.4, we can assume without loss of generality that $p = q$. By Lemma 2.3, $chase_{\{\langle p,I \rangle\}}(J) = \langle p,I \rangle(J)$. Thus, $\langle p,I \rangle \models \langle p,J \rangle$ iff there exists a

valuation h such that h(p) = p and h(I) $\subseteq$ J. But for every tuple u $\in$ J, if u[A] $\neq$ p[A], then u[A] is a distinct value. Let S(u) = {A | u[A] = p[A]}. Obviously, there is a valuation h as above iff for all u $\in$ I there is a tuple v $\in$ J such that u[S(u)] = v[S(u)]. This can be checked in polynomial time. <>

By adding an fd to the jd we can get a lower bound for implication of jd's, thus improving upon Theorem 2.2.

Theorem 4.3. Testing whether a jd is implied by a jd and an fd is NP-complete.

Proof.

In NP: By Lemma 4.3.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct the relations I and J as described above. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 1$, $1 \leq j \leq m$, and p[A] = 1. <p,I> is a jd. Let D be {<p,I>,X->A}. Let q be a tuple: $q[X_i] = 1$, $1 \leq i \leq n$, $q[S_j] = 2$, $1 \leq j \leq m$, and q[A] = 1. <q,J> is a jd. C contains an exact cover of X iff D |= <q,J>: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get h(I) $\subseteq$ J, $h(p)[X_i] = 1$, $1 \leq i \leq n$, and P[A] = 1. To get q in $chase_D(J)$ we apply first <q,I> and get $J_1 = J \cup \{h(p)\}$. Now h(p)[X] = v[X], and we can apply X->A and identify v[A] with h(p)[A], to obtain $J_2 = J_1 \cup \{q\}-\{v\}$. Thus, D |= <q,J>. If C does not contain an exact cover of X, then h(I) $\subseteq$ J implies |h(I)| = 1. Also, no two tuples in J agree on X. Thus, $chase_D(J) = J$ and D $\not\models$ <q,J>. <>

Corollary. Testing whether a jd is implied by a jd and an mvd is NP-hard.

Proof. The claim follows from the above theorem by Theorem 3.1. <>

Note that a lower bound for testing implications of jd's by sets of mvd's is not known.

Testing implication of jd's by ttgd's is also probaly computationally intractable.

Theorem 4.4. Testing whether a jd is implied by a ttgd is NP-hard.

Proof. Let X,C be an instance of EXACT COVER, and construct the relations I and J as described above. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 2$, $1 \leq j \leq m$, and $p[A] = 1$. <p,I> is a ttgd and <p,J> is a jd. C contains an exact cover of X iff <p,I> |= <p,J>: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$ and h(p) = p. Thus, $p \in \text{chase}_{\{<p,I>\}}(J)$ and <p,I> |= <p,J>. If C does not contain an exact cover of X, the $h(I) \subseteq J$ implies |h(I)| = 1. Thus, $\text{chase}_{\{<p,I>\}}(J) = J$ and <p,I> $\not\models$ <p,J>. <>

Testing implication by egd's is also probably computationally intractable.

Theorem 4.5. Testing whether an egd is implied by an egd is NP-complete.

Proof.

In NP: By Lemma 4.3.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct the relations I and J as described above. Let a denote the pair (<A,1>,<A,2>). <a,I> and <a,J> are egd's. C contains an exact cover of X iff <a,I> |= <a,J>: If C contains an exact cover of X, then

defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$, $h(\langle A,1\rangle) = \langle A,1\rangle$ and $h(\langle A,2\rangle) = \langle A,2\rangle$. Thus, $\langle A,2\rangle \notin chase_{\{\langle a,I\rangle\}}(J)$ and $\langle a,I\rangle \models \langle a,J\rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Thus, $chase_{\{\langle a,I\rangle\}}(J) = J$ and $\langle a,I\rangle \not\models \langle a,J\rangle$. <>

Theorem 4.6. Testing whether a jd is implied by an egd is NP-complete.

Proof.

In NP: By Lemma 4.3.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct the relations I and J as described above. Let a denote the pair $(\langle A,1\rangle,\langle A,2\rangle)$. $\langle a,I\rangle$ is an egd. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 2$, $1 \leq j \leq m$, and $p[A] = 1$. $\langle p,J\rangle$ is a jd. C contains an exact cover of X iff $\langle a,I\rangle \models \langle p,J\rangle$: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$, $h(\langle A,1\rangle) = \langle A,1\rangle$ and $h(\langle A,2\rangle) = \langle A,2\rangle$. Thus, by applying $\langle a,I\rangle$, we identify $\langle A,2\rangle$ with $\langle A,1\rangle$ and get $p \in chase_{\{\langle a,I\rangle\}}(J)$ and $\langle a,I\rangle \models \langle p,J\rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Thus, $chase_{\{\langle a,I\rangle\}}(J) = J$ and $\langle a,I\rangle \not\models \langle p,J\rangle$. <>

A non-trivial egd can not be implied by ttgd's [BV3]. However:

Theorem 4.7. Testing whether an egd is implied by a jd and an fd is NP-complete.

Proof.

In NP: By Lemma 4.3.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct the relations I and J as described above. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 1$, $1 \leq j \leq m$, and $p[A] = 1$. $\langle p,I \rangle$ is a jd. Let D be $\{\langle p,I \rangle, X \rightarrow A\}$. Let a denote the pair $(\langle A,1 \rangle, \langle A,2 \rangle)$. $\langle a,J \rangle$ is an egd. C contains an exact cover of X iff $D \models \langle a,J \rangle$: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$, $h(p)[X] = v[X]$ and $h(p)[A] = 1$. To get that $\langle A,2 \rangle \notin chase_D(J)$ we apply first $\langle p,I \rangle$ and add h(p) to J, then we apply X->A and identify v[A] with h(p)[A]. Thus, $\langle A,2 \rangle \notin chase_D(J)$ and $D \models \langle a,J \rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Also, no two tuples in J agree on X. Thus, $chase_D(J) = J$ and $D \not\models \langle a,J \rangle$. <>

We conclude this subsection with the following observation. The EXACT COVER problem is shown in [GJ] to be NP-complete even if each element occurs in at most three sets. Thus our NP-hardness results holds also for dependencies $\langle *,I \rangle$ (* is either a tuple or a pair of values) where each value has at most three occurences in I. However, in more restricted cases we can find polynomial time implication tests. E.g., if we require $|I| = 2$, then we can test whether $\langle *,I \rangle$ is implied by a set of ttgd's and egd's in O(sn), where s is the size of the input and n is the number of attributes [BV3,BV4]. (See also [BB,Beer,MSY,Val] for efficient implication testing procedures for restricted cases).

4.3 Additional NP-completeness Results

From the reduction of Subsection 4.1 we can derive additional results.

Theorem 4.8. Testing whether a relation does not satisfy an egd is NP-complete.

Proof.

In NP: Let $e = \langle (a_1,a_2), I \rangle$ be an egd, and let J be a relation. J does not satisfy e iff for some valuation h on I we have $h(I) \subseteq J$ and $h(a_1) \neq h(a_2)$. Obviously this can tested in nondeterministic polynomial time.

Hard for NP: Let X,C be an instance of EXACT COVER, and construct I and J as described above. Let a denote the pair $(\langle A,1\rangle,\langle A,2\rangle)$. $\langle a,I\rangle$ is an egd. C contains an exact cover of X iff J does not satisfy $\langle a,I\rangle$: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$, $h(\langle A,1\rangle) = \langle A,1\rangle$ and $h(\langle A,2\rangle) = \langle A,2\rangle$. Thus, J does not satisfy $\langle a,I\rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Thus, $h(\langle A,1\rangle) = h(\langle A,2\rangle)$ and J satisfies $\langle a,I\rangle$. <>

Note that both the relation and the egd are considered as parts of the instance of the problem.

Theorem 4.9. Testing containment of tableaux is NP-complete.

Proof.

In NP: Let $\langle p,I\rangle$ and $\langle q,J\rangle$ be tableaux. By Lemma 2.1, $\langle p,I\rangle \geq \langle q,J\rangle$ iff there is a valuation h on I such that $h(I) \subseteq J$ and $h(p) = q$. Obviously, this can be tested in nondeterministic polynomial time.

Hard for NP: Let X,C be an instance of EXACT COVER, and constrcut the relations I and J as described above. Let p be a tuple: $p[X_i] = 1$, $1 \leq i \leq n$, $p[S_j] = 2$, $1 \leq j \leq m$, and $p[A] = 1$. $\langle p,I\rangle$ and $\langle p,J\rangle$ are

tableaux. C contains an exact cover of X iff $\langle p,I\rangle \geq \langle p,J\rangle$: If C contains an exact cover of X, then defining a valuation h on I as described in the proof of Lemma 5.2, we get $h(I) \subseteq J$ and $h(p) = p$. Thus $\langle p,I\rangle \geq \langle p,J\rangle$. If C does not contain an exact cover of X, then $h(I) \subseteq J$ implies $|h(I)| = 1$. Thus, either $h(I) = v$ and $h(\langle A,1\rangle) = \langle A,2\rangle$, or $h(I) = u_j$ and $h(\langle S_j,2\rangle) = \langle S_j,1\rangle$, for some $1 \leq j \leq m$. In either case $h(p) \neq p$, so $\langle p,I\rangle \not\geq \langle p,J\rangle$. <>

The last theorem sharpens a result of [ASU,SY], since their class of tableaux is more general.

5. TESTING THE TRIVIALITY OF GENERAL TGD'S

Generalizing tgd's, a general tuple generating dependency (abbr. gtgd) says that if some tuples, fulfilling certain conditions, exist in the database, then some other tuples (possibly with some unknown values), fulfilling certain conditions, must also exists in the database. Formally, a gtgd is a pair of relations $\langle I`,I\rangle$. It is satisfied by a relation J if for any valuation h, such that $h(I) \subseteq J$, there is an extension h` of h to I` so that $h`(I`) \subseteq J$.

A dependency d is trivial if it is satisfied in every relation, i.e., if it is implied by the empty set of dependencies.

Lemma 5.1. [BV3] A dependency d is trivial iff

a) d is an egd $\langle (a_1,a_2),I\rangle$ and $a_1 = a_2$, or

b) d is a tgd $\langle w,I\rangle$ and $w[ATTR(d)] \in I[ATTR(d)]$, or

c) d is a gtgd $\langle J,I\rangle$ and for some valuation h, which is the identity

on I, h(J) $\subseteq$ I. <>

While condition (a) and (b) of Lemma 5.1 can be checked by inspection, condition (c) is not so trivial to check.

Theorem 5.1. Testing the triviality of gtgd's is NP-complete.

Proof.

In NP: Nondeterministically choose a valuation h and check for condition (c) of Lemma 5.1.

Hard for NP: We reduce the tableaux containment problem of Theorem 4.9 to our problem. Given two tableaux <u,I> and <v,J>, we can assume, without loss of generality (by Lemma 2.2), that u = v and there are no other values common to I and J. <u,I> $\leq$ <u,J> iff <J,I> is trivial. If <u,I> $\leq$ <u,J>, then by Lemma 2.1 there is a valuation h, such that h(J) $\subseteq$ I and h(u) = u. But this means that h is the identity on I, hence, <J,I> is trivial by Lemma 5.1. If <J,I> is trivial, then by Lemma 5.1 there is a valuation h, which is the identity on I, such that h(J) $\subseteq$ I. But this means that h(u) = u, hence <u,I> $\leq$ <u,J> by Lemma 2.1. <>

REFERENCES

[ABU] Aho, A.V., Beeri, C., Ullman, J.D.: The theory of joins in relational databases. ACM Trans. on Database Systems 4:3(Sept. 1979), pp. 297-314

[ASU] Aho, A.V., Sagiv, Y., Ullman, J.D.: Equivalence among relational expressions. SIAM J. Comput. 8:2(1979), pp. 218-246.

[BB] Beeri, C., Bernstein P.A.: Computational problems related to the design of normal form relational schemas. ACM Trans. on Database Systems 4:1 (March 1979), pp. 30-59.

[Beer] Beeri, C.: On the membership problem for multivalued dependencies. ACM Trans. on Database Systems 5:3(Sept. 1980), pp. 241-259.

[BV1] Beeri, C., Vardi, M.Y.: On the properties of total join dependencies. Workshop on Formal Bases for Data Bases, Toulouse, Dec. 1979.

[BV2] Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. Preliminary Report, Dept. of Computer Science, The Hebrew Univ. of Jerusalem, May 1980.

[BV3] Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. Preliminary Report, Dept. of Computer Science, The Hebrew Univ. of Jerusalem, August 1980.

[BV4] Beeri, C., Vardi, M.Y.: Inferring multivalued dependencies from tuple and equality generating dependencies. in preparation.

[Codd] Codd, E.F.: Further normalization of the data base relational model. in Data Base Systems (R. Rustin, ed.), Prentice-Hall, N.J., 1972, pp. 33-64.

[Fag1] Fagin, R.: Multivalued dependencies and a new normal form for relational databases. ACM Trans. on Database Systems 2:3(Sept. 1977), pp. 262-278.

[Fag2] Fagin, R.: Horn clauses and database dependencies. Proc. 12th Ann. ACM Symp. on Theory of Comput., 1980, pp.123-134.

[GJ] Garey, R.M., Johnson, D.S.: Computers and intractability. Freeman, San Francisco, 1979.

[JP] Janssens, D., Paredaens, J.: General dependencies. Workshop on Formal Bases for Data Bases, Toulouse, Dec. 1979.

[Ka] Karp, R.M.: Reducibility among combinatorial problems. in Complexity of Computer Computation (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 85-104.

[MMS] Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of data dependencies. ACM Trans. on Database Systems 4:4(Dec. 1979), pp. 455-469.

[MSY] Maier, D., Sagiv, Y., Yannakakis, M.: On the complexity of testing implications of functional and join dependencies. to appear in J. ACM.

[Pa] Paredaens, J.: A universal formalism to express decomposition, functional dependencies and other constraints in a relational database. Research Report, Dept. of Mathematics, University of Antwerp, 1980.

[Riss] Rissanen, J.: Theory of relations for databases - a tutorial survey. Proc. 7th Symp. on Math. Found. of Computer Science, Poland, 1978, Lecture Notes in Computer Science 64, Springer-Verlag, pp. 537-551.

[SU] Sadri, P., Ullman J.D.: A complete axiomatization for a large class of dependencies in relational databases. Proc 12th Ann. ACM Symp. on Theory of Comput., 1980, pp.117-122.

[SY] Sagiv, Y., Yannakakis, M.: Equivalence among relational expressions with the union and difference operators. to appear in J. ACM.

[Val] Vardi, M.Y.: Inferring multivalued dependencies from functional and join dependencies. Research Report, Dept. of Applied Math.,

Weizmann Inst. of Science, March 1980.

[Va2] Vardi, M.Y.: Axiomatization of functional and join dependencies in the relational model. M.Sc. Thesis, Department of Applied Mathematics, Weizmann Institute of Science, April 1980.

[YP] Yannakakis, M., Papadimitriou, C.: Algebraic dependencies. 21st Ann. IEEE Sypm. on Found. of Computer Science, Syracuse, 1980, pp. 328-332.

[Zan] Zaniolo, C.: Analysis and design of relational schemata for database systems. Technical Report UCLA-ENG-7769, Department of Computer Science, UCLA, July 1976.