A PROOF PROCEDURE FOR DATA DEPENDENCIES

Preliminary Report

by

C. Beeri and M.Y. Vardi*

Department of Computer Science

The Hebrew University of Jerusalem

Jerusalem, Israel

August 1980

---

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

ABSTRACT

A class of dependencies - tuple and equality generating dependencies - is defined, and the "chase" process of [MMS] is generalized to deal with these dependencies. For total dependencies the chase is an exponential time decision procedure for the implication problem, and in some restricted cases it can be modified to run in polynomial time. For partial dependencies the chase is only a proof procedure, however, we show several cases for which it is a decision procedure. It is shown that equality is redundant for deciding implication of tuple generating dependencies, and is "almost redundant" for deciding implication of equality generating dependencies. By expressing dependencies in the language of first order logic, we draw a correspondence between the chase and refutation by resolution and paramodulation.

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 was followed by the multivalued dependencies [Fag1,Zan]. Recently, a number of generalizations of these dependencies have appeared: mutual dependencies [Nico1], join dependencies [ABU,Riss], transitive dependencies [Pa1], general dependencies [JP],and subset dependencies [SW]. In this paper we generalize all these dependencies, and define tuple generating dependencies and equality generating dependencies. 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.

The decision problem for dependencies is to decide whether a given set of dependencies logically implies another dependency. A decision procedure[1] for the implication problem for functional and join

---

[1] We distinguish between a decision procedure, which always halts, and a proof procedure, which may run forever if the answer to the decision problem is negative.

dependencies called the chase was developed by [ABU,MMS]. This procedure, generalized to tuple generating dependencies and equality generating dependencies, is a proof procedure for the implication problem. In several cases, however, the chase is also a decision procedure, e.g., if all dependencies are total; in this case we also show how the chase may lead to an efficient decision procedure.

The formalism we use is that of tableaux [ASU]. An alternative formalism is that of first order logic with identity [Nico2]. By describing the chase in the formalism of first order logic, we show that it is in fact a refinement of a well-known proof procedure - refutation by resolution and paramodulation [Ro,RW].

The outline of the paper is as follows. In Section 2 we define the relational model, tableaux and dependencies. The implication problem and the chase as a decision procedure for implication of total dependencies is described in Section 3. We study the complexity of the chase, and investigate the role of equality in the chase. It is shown that the result of the chase has a closure property which enables us to devise efficient tests for the implication of multivalued dependencies. In Section 4 we define partial dependencies, for which the chase is a proof procedure. We point several classes of partial dependencies for which the chase is a decision procedure. In Section 5 we express dependencies in the language of first order logic, and describe the correspondence between the chase and refutation by resolution and paramodulation. We conclude the paper by pointing out several possible generalization and their limitation in Section 6.

It should be noted that the generalizations of dependencies to tuple generating and equality generating dependencies was also done independently by several researchers [Fag3,GJ,Pa2,YP,SU], though the degree of generality differs from one to the other. The generalization of the chase to tgd's was done also by [SU] and to a certain extent by [Pa3].

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 $Dom = DOM(A_1) \cup \ldots \cup DOM(A_n)$. For a set X, an X-value is a mapping $w: X \rightarrow Dom$, such that $w(A) \in DOM(A)$ for all $A \in X$. A tuple is a U-value. A relation is a set (not necessarily finite) of tuples. We use the letters t,u,w,... to denote tuples, and I,J,... to denote relations.

For an tuple w and a set Y $\subseteq$ U, we denote the restriction of w to Y by w[Y]. With each attribute A we associate the set of all possible A-values, VAL(A) = {w[A] | w is a tuple}. Let I be a relation. The set of A-values in I is I[A] = {a | w $\in$ I and w[A] = a}.

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 with 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}. Observe that I $\subseteq$ T(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.

As an example consider functional dependencies (abbr. fd) [Codd], and multivalued dependencies (abbr. mvd) [Fag1,Zan]. An fd is a statement X->Y. It is satisfied by a relation I if for all tuples $t_1,t_2 \in I$, we have that if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. An mvd is a statement X->->Y. It is satisfied by a relation I if for all $t_1,t_2 \in I$, we have that if $t_1[X] = t_2[X]$ then there exists a tuple $t \in I$ such that $t[XY] = t_1[XY]$ and $t[XZ] = t_2[XZ]$, where $Z = \overline{XY}$.

Generalizing fd's, an equality generating dependency (abbr. egd) says that if some tuples, fulfilling certain conditions, exist in the database, then some values in the database 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 $h(a_1) = h(a_2)$.

Generalizing mvd's, a total tuple generating dependency [2] (abbr. ttgd) says that if some tuples, fulfilling certain conditions, exist

---

[2] The reason for this terminology will be clarified in Section 4.

in the database, then another tuple must also exist in the database. Formally, a ttgd is a tableau T = <w,I>. A relation J satisfies T if for any valuation h, such that h(I) $\subseteq$ J, we have that w $\in$ J. That is, J satisfies T if T(J) = J. Observe that we use T both as an operator on relation and as a dependency. The meaning should be clear fron the context.

3. A DECISION PROCEDURE FOR TOTAL DEPENDENCIES

3.1 The Implication Problem

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 $\models$ 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 $\models$ d. Algorithms which decide implication for some families of dependencies were invesigated by [BB,Beer,MMS,MSY,Va1].

In the sequel D denotes a set of dependencies, and d denotes a dependency.

3.2 The Chase

Let D be a set of ttgd's, and let T=<w,I> be a ttgd. To decide whether

$D \models d$ we add to I all tuples as necessitated by the dependencies in D, and then look for w. Such a process is called a chase, since inconsistencies in I are "chased out". The chase is performed by applying the following T-rules to I, until no change is effected.

T-rule: For some $T \in D$, add T(I) to I.

Since we do not require any specific order of T-rules applications, many chases are possible.

Lemma 1. All chases of I terminate and yield a unique result, denoted $chase_D(I)$, such that $chase_D(I) \in SAT(D)$. <>

The chase can be employed as a decision procedure.

Theorem 1. $D \models \langle w,I \rangle$ iff $w \in chase_D(I)$. <>

Intuitively, "chasing" with egd's consist of identifying values. If D contains also egd's then the chase is performed by applying E-rules and T-rules, until no more change is effected.

E-rule: For some egd $\langle (a_1,a_2),J \rangle \in D$ and a valuation h such that $H(J) \subseteq I$, identify $h(a_1)$ and $h(a_2)$ in I.

We assume that for all attributes $A \in U$, VAL(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] to be the smallest value in VAL(A), for all $A \in U$. Given $\langle (a_1,a_2),I \rangle$, we take a_1,a_2 as the smallest values in VAL(A) and $a_1 < a_2$. Thus, the values in w or a_1,a_2 do not change in the computation of $chase_D(I)$. It follows that Lemma 1 and Theorem 1 still hold.

To decide implication of egd's we use the following theorem.

Theorem 2. $D \models \langle (a_1,a_2),I\rangle$ iff $a_2 \notin chase_D(I)$. <>

A dependency is trivial if $\models d$, i.e., d is satisfied by every relation. Obviously a trivial dependency is a meaningless constraint.

Lemma 2. A dependency d is trivial iff

a) d is $\langle w,I\rangle$ and $w \in I$, or

b) d is $\langle (a_1,a_2),I\rangle$ and $a_1=a_2$. <>

3.3 Complexity Analysis

We study here the complexity of deciding implication for ttgd's. The analysis for ttgd's and egd's is similiar. Let d be $\langle w,I\rangle$, where $|I| = m$. Thus, $|chase_D(I)| \leq m^n$. Let the number of attribute values in D be k, then there are m^k possible valuations. Let the number of tuples in D be l, then checking each valuation takes $O(nl \cdot m^n)$ time.

Theorem 3. Deciding whether $D \models d$ can be done in $O(s \cdot m^{2n+k})$ time, where s is the number of symbols in D and d. <>

3.4 The Role of Egd's

Given D and d, we can decide whether $D \models d$ "almost" without considering egd's.

Consider first the case that d is a ttgd. Let e be an egd in D, $e = \langle (a_1,a_2),I\rangle$. Let w_1 be a tuple such that $w_1[A] = a_1$, and for all attributes $B \neq A$, $w_1[B] \notin I[B]$. Let w_2 be a tuple such that

$w_2[\bar{A}] = w_1[\bar{A}]$ and $w_2[A] = a_2$. We associate with e two tgd's. e_1 is $\langle w_1, I \cup \{w_2\}\rangle$, and e_2 is $\langle w_2, I \cup \{w_1\}\rangle$. Let D^* be the result of relacing each egd e ∈ D by e_1 and e_2

Lemma 3. e ⊨ e_1 and e ⊨ e_2. ⟨⟩

Theorem 4. Let d be a ttgd, D ⊨ d iff D^* ⊨ d. ⟨⟩

Consider now the case that d is an egd, then D must contain some egd's, otherwise:

Lemma 4. Let D be a set of ttgd's, and let d be an egd, then D ⊨ d iff d is trivial. ⟨⟩

Though if d is a non-trivial egd then egd's in D are necessary, their role can be minimized.

Theorem 5. D ⊨ $\langle (a_1,a_2),I\rangle$ iff for some egd $\langle (a_i,a_j),J\rangle$ ∈ D and a valuation h, h(J) ⊆ $chase_D^*(I)$, $h(a_i) = a_1$ and $h(a_j) = a_2$. ⟨⟩

3.5 Tuple Closure

Let D_1 and D_2 be sets of dependencies, and let d be a dependency. Obviously, if D_1 ⊨ D_2, then D_1 ⊨ d iff $D_1 \cup D_2$ ⊨ d.

Lemma 5. D_1 ⊨ D_2 implies $chase_{D_1}(I) = chase_{D_1 \cup D_2}(I)$. ⟨⟩

Let I be a relation. For all w ∈ $chase_D(I)$, we have D ⊨ $\langle w,I\rangle$. Thus, the tuples of the chase represent ttgd's implied by D. Combining this with Lemma 5 we get the tuple closure property.

Theorem 6. For all w ∈ $chase_D(I)$, we have D ⊨ $\langle w,I\rangle$ and $\langle w,I\rangle(chase_D(I)) = chase_D(I)$. ⟨⟩

Using the closure property, we can design in some cases a plynomial time decision procedure. Consider the case that $I = \{w_1, w_2\}$, and let $X = \{A \mid w_1[A] = w_2[A]\}$. Obviously, for all $w \in chase_D(I)$, we have $w[X] = w_1[X] = w_2[X]$. We can characterize each tuple $w \in chase_D(I)$ by the set $FIRST(w) = \{A \in \overline{X} \mid w[A] = w_1[A]\}$, since $w[\overline{X}\text{-}FIRST(w)] = w_2[\overline{X}\text{-}FIRST(w)]$. By definition, $FIRST(w_1) = \overline{X}$ and $FIRST(w_2) = \emptyset$.

Applying Theorem 6 we get a Boolean closure property.

Lemma 6. Let $I = \{w_1, w_2\}$. For all $w', w'' \in chase_D(I)$, there exist tuples $t_1, \ldots, t_4 \in chase_D(I)$, such that:

a) $FIRST(t_1) = FIRST(w') \cup FIRST(w'')$.

b) $FIRST(t_2) = FIRST(w') \cap FIRST(w'')$.

c) $FIRST(t_3) = FIRST(w') - FIRST(w'')$.

d) $FIRST(t_4) = FIRST(w'') - FIRST(w')$. <>

Theorem 7. Let $I = \{w_1, w_2\}$. There exist a partition of $\overline{X}$ to disjoint nonempty sets $W_1, \ldots, W_m$, $1 \leq m$, such that $w \in chase_D(I)$ iff for all i, $1 \leq i \leq m$, either $W_i \subseteq FIRST(w)$ or $W_i \cap FIRST(w) = \emptyset$. <>

To decide if $D \models \langle w, I \rangle$, one has to construct $W_1, \ldots, W_m$, and check for the condition of the theorem. This can be done in polynomial time. In fact, if $|I| = 2$ then $\langle w, I \rangle$ corresponds to the multivalued dependency $X \rightarrow\rightarrow FIRST(w)$, and Theorem 7 is the key to the efficient algorithms for inferences of multivalued dependencies of [Beer,BV4,MSY,Va1].

4. A PROOF PROCEDURE FOR DEPENDENCIES

4.1 Tuple Generating Dependencies

Generalizing ttgd's, a tuple generating dependency (abbr. tgd) 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 tgd 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 $h'(I') \subseteq J$.

If for all attributes A we have $I'[A] \subseteq I[A]$, then $\langle I', I\rangle$ is total, and is partial otherwise. Viewing a ttgd $\langle w, I\rangle$ as a tgd $\langle \{w\}, I\rangle$, we see that a ttgd is a specific case of a tgd. By the following lemma there is no loss of generality in assuming that every total tgd is of the form $\langle w, I\rangle$.

Lemma 7. Let $\langle I', I\rangle$ be a total tgd, then $\langle I', I\rangle \models\models \{\langle w, I\rangle \mid w \in I'\}$. $\langle\rangle$

Corolarry. Let $\langle I', I\rangle$ be a total tgd, we can effectively construct a ttgd $\langle w, J\rangle$, such that $\langle I', I\rangle \models\models \langle w, J\rangle$. $\langle\rangle$

This justifies our definition of ttgd in Section 3.

The following condition for triviality generalizes Lemma 2(a).

Lemma 8. A tgd $\langle I', I\rangle$ is trivial iff there exist a valuation h, which is the identity on I, such that $h(I') \subseteq I$. $\langle\rangle$

The chase enable us to decide implication of tgd's by ttgd's and egd's.

Theorem 8. Let D be a set of ttgd's and egd's, then $D \models \langle I', I\rangle$ iff $\langle I', chase_D^*(I)\rangle$ is trivial. <>

Corolarry. $D \models \langle I', I\rangle$ iff there exist ttgd's $\langle w_1, I\rangle, \ldots, \langle w_k, I\rangle$, $k > 0$, such that $D \models \{\langle w_i, I\rangle \mid 1 \leq i \leq k\}$, and for some valuation h which is the identity on I, $h(I') \subseteq \{w_1, \ldots, w_k\}$. <>

4.2 Non-terminating Chase

Trying to generalize our T-rule to tgd's we encounter difficulties, because the new tuples, whose existence in the database is implied by the existence of some other tuples, are only partly known. The solution is to replace each unknown value by a new distinct value. let $\langle I', I\rangle$ be a tgd and h a valuation. A distinct extension of h to I' is an extension h' of h to I', where for all attributes A, if $a \in I'[A] - I[A]$ than h'(a) ia a new distinct value. Our generalized chase rule is now:

T-rule: For some $\langle J', J\rangle \in D$ and a valuation h such that $h(J) \subseteq I$, but for no extension h' of h we have $h'(J') \subseteq I$, add h'(J') to I for some distinct extension h' of h to J'.

Since this rule introduces new values, termination of the chase is not assured any more. If D is a set of tgd's, then we take $chase_D(I)$ to be any (possibly infinite) relation resulting from the chase, and we have $chase_D(I) \in SAT(D)^{(3)}$.

Theorem 9.

a) $D \models \langle I', I\rangle$ iff $\langle I', chase_D^*(I)\rangle$ is trivial.

b) $D \models \langle (a_1, a_2), I\rangle$ iff for some egd $\langle (a_i, a_j), J\rangle \in D$ and a valuation h, $h(J) \subseteq chase_D^*(I)$, $h(a_i) = a_1$ and $h(a_j) = a_2$. $\langle\rangle$

Obviously the condition of Theorem 9 are not practical if $chase_D^*(I)$ is infinite. In practice, we execute repeatedly, until a positive answer is obtained, the following step:

apply T-rules a finite number of times, and test for the conditions of the theorem.

If $D \models d$, then we are bound to get a positive answer. But if $D \not\models d$, then we may chase forever. Thus, in general the chase constitutes a proof procedure and not a decision procedure. Obviously, the set $\{\langle D,d\rangle \mid D \models d\}$ is recursively enumerable. It is an open problem whether it is recursive or not.

4.3 Solvable cases

Though the recursive solvability of the implication problem in general is still open, we can show that it is solvable for several cases, the simplest of which is of course the case where D consists of ttgd's and egd's.

In some cases solvability follows from the fact that the answer to the implication problem is trivially negative.

Lemma 9. In the following cases $D \models d$ iff d is trivial:

---

(3) That is, we take the (possibly infinite) union of all intermediate relation obtained during the chase.

a) D is a set of tgd's and d is an egd.

b) D is a set of partial tgd's and d is a ttgd. <>

That is, an egd can not be equivalent to a tgd, and a partial tgd can not be equivalent to a ttgd.

Though in general the chase does not terminate, in some cases does terminate. An attribute A is partial in a tgd $\langle I', I\rangle$ if $I'[A] \not\subseteq I[A]$. It is partial in a set D of dependencies if it is partial in some tgd in D.

Theorem 10. Let D be a set of dependencies such that at most one attribute A is partial in D, then the chase terminates.<>

This case includes of course the case that all tgd's in D are total.

In some cases we can force termination of the chase by applying the rules in a specific order. We assume that all tgd's are of the form $\langle\{w\},I\rangle$. For each tgd $d = \langle\{w\},I\rangle$ we define an attribute set $TOTAL(d) = \{A \mid w[A] \in I[A]\}$. Let d be a tgd $\langle\{w\},I\rangle$, $X = TOTAL(d)$, and let Y be an attribute set $Y \subseteq X$. Define w_Y as a tuple such that $w_Y[Y] = w[Y]$, and all other values of w_Y are new values; and define w_Y' as a tuple such that $w_Y'[X] = w[X]$, and $w_Y'[\overline{X}] = w_Y[\overline{X}]$. We associate with d a ttgd $d_Y = \langle w_Y', I \cup \{w_Y\}\rangle$.

Theorem 11. Let D be a set of tgd's of the form $\langle\{w\},I\rangle$, and let Y be an attribute set, such that for all $d \in D$, we have $Y \subseteq TOTAL(d)$ and $D \models d_Y$, then there is a chase that terminates. <>

It seems that this theorem is useless because its condition $D \models d_Y$ may itself be recursively unsolvable. However, in some practical cases it can be decided easily, enabling us employ the chases as a decision

procedure.

We have observed that the set $\{\langle D,d\rangle \mid D \models d\}$ is recursively enumerable. Thus, to show that it is recursive, it suffices to show that its complement $\{\langle D,d\rangle \mid D \not\models d\}$ is also recursively enumerable. An attribute A is <u>non-repeating</u> in a relation I if $|I[A]| = |I|$. For a set of tgd's D and a tgd $\langle I', I\rangle$ we define the following property:

(*) for all attributes A, if A is partial in D then A is non-repeating in I', and for all $\langle J', J\rangle \in D$, A is non-repeating in J.

<u>Theorem 12</u>. The set $\{\langle D,d\rangle \mid D \not\models d$ and $\langle D,d\rangle$ satisfies (*)$\}$ is recursively enumerable. <>

Theorem 12 can be slightly modified to include egd's as well.

A tgd $\langle\{w\},\{w_1,w_2\}\rangle$ is called a <u>subset dependency</u> (abbr. sd) [SW]. Let Z be an attribute set. A Z-sd is an sd $\langle\{w\},I\rangle$, such that for some $w' \in I$, we have $Z = \{A \mid w[A] = w'[A]\}$.

<u>Theorem 13</u>. [SW] For the class of Z-sd's, the set $\{\langle D,d\rangle \mid D \not\models d\}$ is recursively enumerable. <>

5. CHASE AND REFUTATION

The chase as described in the preceeding sections is a combinatorial search for a counterexample to the implication $D \models d$. However by describing the chase in the formalism of first order logic we show that it is in fact a refinement of a well-known theorem proving procedure - resolution with paramodulation [Ro,RW]. It is interesting

to note that the use of resolution with paramodulation for deciding implication of dependencies has already been advocated by [Nico2].

For simplicity we deal only with ttgd's and egd's.

6.1 First Order Dependencies

Let us fix some specific order $A_1,\ldots,A_n$ for the attributes of U. We can take a tuple on U to be an n-tuple $\langle a_1,\ldots,a_n\rangle$ such that $a_i \in DOM(A_i)$, and a relation on U is a subset of the Cartesian product $DOM(A_1)X\ldots XDOM(A_n)$.

The language we are going to use is that of first order logic with equality with no function symbols and with one n-ary predicate symbol, say R. Indexed y's are used as universally quantified variable symbols, and indexed x's are used as existentially quantified variable synbols. Indexed v's are syntactic variables ranging over variable synbols. An atomic formula $R(v_1,\ldots,v_n)$ is a predicate formula, and an atomic formula $v_i = v_j$ is an equality formula.

To express a dependency d = <*,I> (* is either a tuple or a pair of values) as a first order dependency observe that:

a) A relation J satisfies d if for any valuation h on I a certain implication is satisfied, where I serves as the antecedent, and * serves as the consequent. Thus, all values are essentially universally quantified variables.

b) All values in * occur in I.

Thus, d can be expressed by the sentence $\forall y_1 \ldots \forall y_k (A_1 /\backslash \ldots /\backslash A_p \rightarrow B)$, where

a) $k,p \geq 1$.

b) The A's are predicate formulas, and the set of variables occuring in them is exactly $\{y_1,\ldots,y_k\}$.

c) B is either a predicate formula or an equality formula.

d) A variable does not occur in two different argumet position of R, and only variables which occur in the same argument position occur as arguments of equalities.

Sentences of this kind are a special case of strict universal Horn sentences.

6.2 Refutation by Resolution and Paramodulation

We assume familiarity with theorem proving terminology as described in [CL].

To prove $D \models d$, we refute $D \cup \{\neg d\}$, i.e., we show that $D \cup \{\neg d\}$ is unsatisfiable. Let d be the sentence $\forall y_1 \ldots \forall y_k (A_1 /\backslash \ldots /\backslash A_p \rightarrow B)$, then $\neg d$ is the sentence $\exists x_1 \ldots \exists x_k (A_1 /\backslash \ldots /\backslash A_p /\backslash \neg B)$, where the y_i's are replaced by x_i's. To refute $D \cup \{\neg d\}$ we must transform it to a clausal form. The clause corresponding to a dependency $\forall y_1 \ldots \forall y_m (C_1 /\backslash \ldots /\backslash C_r \rightarrow D)$ is $\neg C_1 \backslash/ \ldots \backslash/ \neg C_r \backslash/ D$. To transform $\neg d$ to a clausal form we first Skolemize it and replace each universally quantified variable x_i by an individual constant a_i. Thus, $\neg d$ is transformed to a set of ground unit clauses $\{A_1,\ldots,A_p,\neg B\}$. Now we have to refute a set of clauses $S = \{d_1,\ldots,d_l,A_1,\ldots,A_p,\neg B\}$, where

the clauses $d_1,\ldots,d_l$ corresponds to the dependencies in D. S is called a dependency clause set. To refute S we generate additional clauses by resolution and paramodulation until the empty clause $\square$ is generated. If $\square$ is not generated then S is satisfiable.

A positive unit hyperresolution is a hyperresolution in which the electrons and the resolvent are positive literals. A strict unit paramodulation is a paramodulation in which both parent clauses are unit clauses.

Lemma 10. Let S be an unsatisfiable dependency clause set, then S can be refuted by positive unit hyperresolution and strict unit paramodulation.

Proof. The claim follows from a similiar result of [HW] for Horn sets. <>

6.3 Chase vs Refutation

We describe now the correspondence between the chase as a decision procedure for $D \models d$ (by Theorems 4 and 5) and the refutation of S (which is the dependency clause set for $D \cup \{\neg d\}$) by positive unit hyperresolution and strict unit paramodulation (by Lemma 10). In fact the chase can be adapted to be a decision procedure for the solvable class of Bernays-Schonfinkel sentences, filling a gap in [Jo].

Let d be <*,I>. The tuples of I corresponds to the positive unit clauses of S, and * corresponds to the negative clause of S (either a predicate literal or an equality literal). The chase is performed by

adding tuples to I either by a T-rule for a ttgd in D, or by a T-rule for an egd in D. (By "a T-rule for an egd" we mean a T-rule for the two ttgd's which replace the egd in D^*, as described in Subsection 3.4). The chase terminates when the condition for * (Theorem 4 and 5) is satisfied, or when no more change can be effected. In the first case $D \models d$, and in the last case $D \not\models d$.

Claim 1. A T-rule application for a ttgd corresponds to a positive unit hyperresolution.

Indeed, let $d' = \langle u,J \rangle$ be a ttgd in D. To apply a T-rule for d', we find a valuation h on J such that $h(J) \subseteq I$, and add h(u) to I. The clause corresponding to d' is $\neg C_1 \vee \ldots \vee \neg C_r \vee D$, where the C's and D are predicate formulas. To apply positive unit hyperresolution, we find positive unit clauses $A_{i_1},\ldots,A_{i_r}$ (corresponding to tuples of I), and a most general unifier σ such that $C_j\sigma = A_{i_j}\sigma$, $1 \leq j \leq r$, to get the resolvent $D\sigma$. Thus, the valuation h corresponds to the most general unifier σ, and the added tuple h(u) corresponds to the resolvent D .

Claim 2. A T-rule application for an egd corresponds to a positive unit hyperresolution followed by a sequence of strict unit paramodulation.

Indeed, let $e = \langle (a_i,a_j),J \rangle$ be an egd in D. To apply a T-rule for e, we replace e by two ttgd's e_1 and e_2 (see Subsection 3.4), and apply a T-rule for e_1 and e_2. That is, we find a valuation h on J such that $h(J) \subseteq I$, and for any tuple $t \in I$, with $t[A] = h(a_i)$, we add to I another tuple t' such that $t'[A] = h(a_j)$ and $t'[\overline{A}] = t[\overline{A}]$, and for any tuple $t \in I$, with $t[A] = h(a_j)$, we add to I another tuple t', such that $t'[A] = h(a_i)$ and $t'[\overline{A}] = t[\overline{A}]$. The clause corresponding to e is

$\sim C_1 \backslash/ \ldots \backslash/ \sim C_r \backslash/ D$, where the C's are predicate formulas and D is an equality formula $y_i = y_j$. By positive unit hyperresolution we generate a resolvent $y_i \sigma = y_j \sigma$, for some most general unifier σ, and tnen by strict unit paramodulation we generate modulants by interchanging $y_i \sigma$ and $y_j \sigma$, where ever they occur in the positive unit clauses of S. This corresponds to the addition of tuples to I be interchanging $h(a_i)$ and $h(a_j)$.

Consider the case that d is $\langle w, I \rangle$. If $D \not\models d$, then eventually the chase will add the tuple w to I. Correspondingly, the clause B will be generated, and B and $\sim B$ resolve to $\square$.

Consider the case that d is $\langle (a_1, a_2), I \rangle$. If $D \not\models d$, then eventually, for some egd $\langle (a_i, a_j), J \rangle$ in D and a valuation h on J such that $h(J) \subseteq I$, we have $h(a_i) = a_1$ and $h(a_j) = a_2$. Correspondingly, by hyeprresolution we generate a positive literal $a_1 = a_2$, which is complementary to $\sim a_1 = a_2$, and the two resolve to $\square$.

In general, a tgd is expressed as a strict universal-existential Horn sentence. Thus, Skolemization introduces function symbols. The new values generated by the chase corresponds to higher order terms generated by resolution.

6. CONCLUDING REMARKS

The framework established in the preceeding sections enables us to

study several possible directions.

6.1 Non-disjoint Domains

In Section 2 we required that $DOM(A) \cap DOM(B) = \emptyset$ if $A \neq B$. In practice, it may be that $A \neq B$ but $DOM(A) = DOM(B)$, e.g., consider the attributes EMPLOYEE# and MANAGER#. It turns out that almost all our results (except for solvability results) carry over to the unrestricted case with minor modification. However, note that in [BV2] it is proven that the implication problem for tgd's, with $U = \{A,B\}$ and $DOM(A) = DOM(B)$ is unsolvable.

6.2 Several Relations

Most of the research in dependency theory is done under the assumption that the database consists of a single relation - the universal relation assumption [BBG]. Recently, this assumtion has been assailed as highly impractical [HLY,Ke]. By adjoining to each tuple a relation name, our dependencies can be both interrelational and intrarelational. However, even for a universe $U = \{A,B\}$ with disjoint domains, the existence of two relations on U yields unsolvability of the implication problem [BV2].

6.3 Constants

The values in our formalism are uninterpreted in the sense that they serve as variables. Thus, while we can express the constraint "a

borrower can borrow at most one book" by the functional dependency BORROWER#->BOOK#, we can not express the constraint "borrower #12345 can borrow at most one book". To this end, we associate with each attribute A a set of constants CON(A) $\subseteq$ DOM(A). For any valuation h and a constant a $\in$ DOM(A), we require h(a) = a. While dependencies without constants have the property that for any relation I and a set of dependencies D, I can be "chased" by D so that $chase_D(I) \in SAT(D)$, this is not the case for dependencies with constants, since it may happen that an application of an E-rule requires the identification of two constants. Also note that the introduction of constants entails the unsolvability of the implication problem [BV2].

6.4 Bounded Domains

In practice some domains may be bounded, not only practically but also conceptually, e.g., |DOM(SEX)| = 2. When dealing with egd's and ttgd's this poses no problem. However, our T-rule of Section 4 presupposes that in each domain there is an infinite supply of values. To account for the boundedness of a domain we should be careful not to introduce more values than available. (Clearly, if all domains are bounded then the implication problem is solvable).

Most of the formalisms for dependencies do not refer directly to the domains. If some domain is bounded, then it is preferable to use our formalism, which refers directly to the available values. Implication of join dependencies with bounded domains is treated in [Fag2].

6.5 Disjunctive Dependencies

Recently, it has been suggested that dependencies with disjunctive conclusion may be of interest [AD,SPDF]. For example, a *disjunctive functional dependency* is a statement X->Y+Z. It is satisfied by a relation I, if for all tuple $t_1, t_2 \in I$, if $t_1[X] = t_2[X]$, then either $t_1[Y] = t_2[Y]$ or $t_1[Z] = t_2[Z]$. Our formalism can be generalized to include disjunctive dependencies. For example, a disjunctive tgd is a pair $\langle I', I \rangle_{dis}$. 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 for some tuple $w \in I'$ we have $h'(w) \subseteq J$. We can also generalize the chase to disjunctive dependencies by treating it as a tree search rather than as a straight line search.

6.6 Finite Implication

Since a database is inherently finite, there is a strong justification to define a relation as a finite set of tuples. We say the a set of dependencies D *finitely implies* a dependency d, denoted $D \models_f d$, if d is satisfied by every finite relation which satisfies all dependencies in D. The finite implication problem is to decide, for a given D and d, whether $D \models_f d$. The solvability of this problem is also an open question. It is even more intracteble than the implication problem, since the is not even e proof procedure for finite implication. In fact, existence of a proof procedure for the finite implication problem entails its solvability [BV2].

6.7 Formal Systems

A lot of effort has been devoted to the developement of formal systems for dependencies [AD,Arm,BFH,Bisk1,Bisk2,BV,Mend,Pa1,Pa2,Sc,Va2,Va3]. Formal systems enjoy several advantages over the chase. A formal system allows us to infer new dependencies from given dependencies; wheras the chase only allows us to check if a dependencies is implied by given dependencies, but it does not tells us how to generate new dependencies. It turns out that the chase is very useful in developing sound and comlete formal systems. In [BV3] we develope several formal systems for tgd's and egd's.

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

[AD] Armstrong, W.W., Delobel, C.: Decomposition and functional dependencies in relations. Publication #271, Department D'Informatique et de Recherche Operationalle, Universite de Montreal, revised Sept. 1979.

[Ar] Armstrong, W.W.: Dependency structure in data base relationships. Proc. IFIP, North-Holland, 1974, pp. 580-583.

[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.

[BBG] Beeri, C., Bernstein, P.A., Goodman, N.:A sophisticate's introduction to database normalization theory. Proc. 4th Intl. Conf. on Very Large Data Bases, Berlin, 1978, pp.113-124.

[Beer] Beeri, C.: On the membership problem for multivalued dependencies. to appear in ACM Trans. on Database Systems.

[BFH] Beeri, C., Fagin, R., Howard, J.H.: A complete axiomatization for functional and multivalued dependencies in database relations. Proc. 3rd ACM-SIGMOD Int'l Conf. on Management of Data, Toronto, August 1977, pp. 47-61.

[Bisk1] Biskup, J.: On the complementation rule for multivalued dependencies in data base relations. Acta Informatica 10:3(1978), pp. 297-305.

[Bisk2] Biskup, J.: Inferences of multivalued dependencies in fixed and undetermined universe. Theoretical Computer Science 10(1980), pp. 93-105.

[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.: Axiomatization of tuple and equality generating dependencies. in preparation.

[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.: A normal form for relational databases that is based on domains and keys. IBM Research Report RJ2520, May 1979.

[Fag3] Fagin, R.: Horn clauses and database dependencies. Proc. 12th Ann. ACM Symp. on Theory of Comput., 1980, pp.123-134.

[GJ] Grant, J., Jacobs, B.E.: On generalized dependencies. to appear.

[HLY] Honeyman, P., Ladner, R.E., Yannakakis, M.: Testing the universal instance assumption. Info. Proc. Lett. 10:1(1980), pp. 14-19.

[HW] Henschen, L., Wos, L.: Unit refutation and Horn sets. J. ACM 21:4(1974), pp. 590-605.

[Jo] Joyner, W.H.: Resolution strategies as decision procedures. J. ACM 23:3(1976), pp. 398-417.

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

[Ke] kent, W.: Consequences of assuming a universal relation. Research Report, IBM General Product Division, Santa Teresa, California, Dec. 1979.

[Mend] Mendelzon, A.O.: On axiomatizing multivalued dependencies in relational databases. J. ACM 26:1(Jan 1979), pp. 37-44.

[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

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.: A normal form for relational databases that is based on domains and keys. IBM Research Report RJ2520, May 1979.

[Fag3] Fagin, R.: Horn clauses and database dependencies. Proc. 12th Ann. ACM Symp. on Theory of Comput., 1980, pp.123-134.

[GJ] Grant, J., Jacobs, B.E.: On generalized dependencies. to appear.

[HLY] Honeyman, P., Ladner, R.E., Yannakakis, M.: Testing the universal instance assumption. Info. Proc. Lett. 10:1(1980), pp. 14-19.

[HW] Henschen, L., Wos, L.: Unit refutation and Horn sets. J. ACM 21:4(1974), pp. 590-605.

[Jo] Joyner, W.H.: Resolution strategies as decision procedures. J. ACM 23:3(1976), pp. 398-417.

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

[Ke] kent, W.: Consequences of assuming a universal relation. Research Report, IBM General Product Division, Santa Teresa, California, Dec. 1979.

[Mend] Mendelzon, A.O.: On axiomatizing multivalued dependencies in relational databases. J. ACM 26:1(Jan 1979), pp. 37-44.

[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 be published.

[Nico1] Nicolas, J.M.: Mutual dependencies and some results on undecomposable relations. Proc. 4th Int'l Conf. on VLDB, 1978, pp.360-367.

[Nico2] Nicolas, J.M.:First order logic formalization for functional, multivalued and mutual dependencies. Proc. ACM-SIGMOD Int'l Conf. on Management of Data, 1978, pp.40-46.

[Pa1] Paredaens, J.: Transitive dependencies in a database scheme. Report R387, MBLE Research Lab, Brussels, Feb. 1979.

[Pa2] 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 Foundations of Computer Science, Poland, 1978, Lecture Notes in Computer Science 64, Springer-Verlag, pp. 537-551.

[Ro] Robinson, J.A.: A machine oriented logic based on the resolution principle. J. ACM 12:1(1965), pp.23-41.

[RW] Robinson, G.A., Wos, L.: Paramodulation and theorem proving in first order theories with equality. in Machine Intelligence, vol 4. (B. Meltzer and D. Michie, eds.), American Elsevier, New York,1969, pp. 135-150.

[Sc] Sciore, E.: A complete axiomatization of full join dependencies. TR #279, Department of EECS, Princeton University, July 1979.

[SPDF] Sagiv, Y., Parker, D.S., Delobel, C., Fagin, R.: An equivalence

between relational database depdencies and a subclass of propositional calculus. to appear in J. ACM.

[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.

[SW] Sagiv, Y., Walecka, S.: Subset dependencies as an alternative to embedded multivalued dependencies. Technical Report UIUCDCS-R-79-980, University of Illinois at Urbana-Champaign, 1979.

[Va1] 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.

[Va3] Vardi, M.Y.: Axiomatization of functional and multivalued dependencies - a comprehensive study. in preparation.

[YP] Yannakakis, M., Papadimitriou, C.: Algebraic dependencies. Bell Labs Report, 1980.

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