DEPENDENCY SATISFACTION IN DATABASES WITH INCOMPLETE INFORMATION

Gösta Grahne

University of Helsinki, Department of Computer Science
Tukholmankatu 2, SF-00250 Helsinki 25, Finland

Abstract. Two of the major problems raised by information incompleteness in databases are how to evaluate queries and how to take data dependencies into account. We give a unified solution of these two intermingled problems for the relational model. Formal criteria for the correctness of the relational algebra and dependency satisfaction are presented. We give a correct redefinition of the complete relational algebra and present a method, called a chase, for enforcing a set of functional and full join dependencies on a relation with null-values of type "value exists, but is presently unknown". This novel chase can also be regarded as a generalization of previously known chase methods. The title of the paper reflects the emphasis of its contribution.

1. INTRODUCTION

The research in the field of information incompleteness has mainly followed three paths. The vast majority of the papers on the topic have sought a way to adapt the query language to handle null-values of different kinds (a.o. [Bis1, Bis2, IL1, IL2, Vas1]), while only a few papers (i.e. [IL3, Lie, Vas2]) have looked at the problems pointed at by the handling of data dependencies under information incompleteness. Even less papers have orientated towards a solution to the problem of updating databases with incomplete information (i.e. [FUV, Sci]). All these directions are however only different aspects of the same problem: how to interpret and accordingly handle incomplete information. (For an overview of the field, see [Gra1] or [Lip].)

This paper offers a unified treatment of incomplete information of the type "value exists, but is presently unknown". Explicit solutions are given for the query problem and the data dependencies in the context of the relational model. Updates will be treated elsewhere [Gra2].

Following [IL2] and [Bis2] we consider a relation containing null-values of the aforementioned type as representing a set of relations of the ordinary kind (i.e. without nulls), one of which corresponds to the state of the real world. Since our information is incomplete we only know the set of possible states, and we store its representative. The query language should then be designed so that only information that holds in all of the possible states is inferred. In [IL2] it is shown how to extend a relational algebra consisting of projection, positive selection, union and join so that it is correct in this sence. A selection is positive if its formula only includes atomic expressions of the form A=a combined with conjunction and disjunction (A is an attribute name, a some domain value). This algebra is quite restrictive, since it would for example be impossible to ask for a list of all em-

This work was supported by the Academy of Finland

ployees earning more than 20000$ in some enterprise database. The authors of [IL2] show that the handling of the complete relational algebra requires a more complicated representative than just putting null-values in ordinary relations. Such a representative, called a c-table, and a correct redefinition of the complete relational algebra for these tables is given in [IL2].

Some authors have tried to solve the problem of data dependencies by incorporating the null-values in the definitions of the data dependencies. The dependencies are however not statements about the incomplete database, but statements about the real world. Thus their definitions remain unchanged, and the question of satisfaction is determined by the set of possible relations, in which each individual relation should satisfy the dependencies. But since we only store the representative relation with null-values, we need some syntactic method for checking dependency satisfaction. This syntactic satisfaction will, in absence of better names, be called "satisfaction".

The argumentation above also shows that the data dependencies affect query evaluation: the incomplete relation determines a set of possible relations, and this set is further restricted by the data dependencies. Since query evaluation is based on the set of possible relations, we must, if we want to maintain the correctness of the query language, restrict the set of possible relations to those relations that satisfy the data dependencies. In [IL3] it is shown how to take into account data dependencies in the system where only positive selections are allowed. Since we feel that this system is too restrictive, we present a system that supports the complete relational algebra and functional and full join dependencies. Chapter 2 of this paper contains the necessary preliminaries and our model that is based on so called tables, and Chapter 3 contains a formal criteria for the correctness of a relational algebra on tables, along with the correct redefinition of the complete relational algebra. In Chapter 4 we give a formal criteria for dependency satisfaction in tables, and we present a method for transforming a table so that the corresponding set of relations will be restricted to those satifying a given set of functional and full join dependencies. The transformation procedure is called a chase, and it can also be regarded as a generalization of previously known chase methods.

2. PRELIMINARIES

For the definitions of the basics of the relational model we refer to [Mai, Ull]. Relation schemes will be denoted by R_1, R_2, ..., and their instances or relations by r_1, r_2, ..., where r_i is a relation over R_i. The set of all relations over R_i is Rel(R_i). For notational convenience we assume one (infinite) common attribute domain D. A tuple is denoted by t, possibly indexed or primed, and t(X) is the restriction of t to the attributes X. The projection operator is denoted by π and the join operator by *. By a relational expression f we mean a well formed expression built up from projection, selection, union, join and difference. A PJ-expression is a relational expression involving only projection and join, for instance $\pi_{AB}(R_1)*\pi_{BC}(R_2)$. In general, if f is performed on relations r_1, r_2, ..., r_n we can write f(r_1, r_2, ..., r_n). The result of performing f is always a single relation of appropriate type.

X->A denotes that X functionally determines A, where A is a single attribute. The join dependency for the set $\mathcal{R}$ = {R_1, R_2, ..., R_p} is denoted by *[$\mathcal{R}$] or *[R_1, R_2, ..., R_p]. The join dependency *[$\mathcal{R}$] is full for the scheme R, if $\bigcup_{i=1}^{p} R_i =$ R. A set of functional dependencies (FDs) and full join dependencies (JDs) is denoted by Σ. The set of JDs in Σ is denoted by Σ_{JD}. It is assumed that Σ is defined for some scheme R. Sat(Σ) is then the set of relations over R that satisfy the dependencies in Σ.

The device for representing a set of relations is called a table. Our tables are essentially extensions of so called c-tables of [IL2]. A table T consists of two parts, a set of c-tuples and a set of general conditions, denoted T_C and T_G. For building up a table we need a set G of all expressions with atoms of the form (x=a), (x=y), true and false combined by ∧, ∨, ¬ and ⇒ (conjunction, disjunction, negation and implication). In the atoms x and y are variables from a countably infinite set V, and a is a value from the domain D. The domain values will be called constants. The variables are used to express the null-values, and we assume that V∩D=∅. A c-tuple t over R is a mapping from R∪{Con} to V∪D∪G, such that t(A)∈V or t(A)∈D for A∈R, and t(Con)∈G. A c-table T_C over R is a finite set of c-tuples over R. T_G is a finite set of general conditions {g|g∈G}. By definition, true is always a member of T_G. The set of all tables over R is denoted by Tab(R). A multitable T is a sequence <T_1, T_2, ..., T_n> of tables T_i∈Tab(R_i), such that all tables have the same set of general conditions T_G. Tab(R) is the set of all multitables over R.

A valuation v is a mapping from V∪D to D, such that for x∈V, v(x) = a for some a∈D, and v(a) = a for all a∈D. Valuations are extended to c-tuples by defining (v(t))(A) = v(t(A)) for A∈R and (v(t))(Con) = true if t(Con) comes to true when all variables x in t(Con) are substituted by v(x). Else (v(t))(Con) is false. The valuation v applies to the general conditions in T_G in the same way, and v(T_G) = true if v(g) = true for all g∈T_G.

The set of relations that a table T represents is denoted by Rep(T) and it is defined in the following way:

Definition 1. For T∈Tab(R), Rep(T) = {r | r∈Rel(R) and there exists a valuation v such that r = v(T_C) and v(T_G) = true}, where v(T_C) = {v(t(R)) | t∈T_C and v(t(Con)) = true}. For a multitable T = <T_1, ..., T_n>, T_i∈Tab(R_i) Rep(T) = {<r_1, ...,r_n> | r_i∈Rel(R_i) and there exists a valuation v such that r_i = v(T_{i_C}) and v(T_G) = true}.

Choosing r = v(T_C) means that we make the closed world assumption [Rei]. The open world assumption would correspond to r ⊇ v(T_C).

The next example shows a table and two of the relations that it represents. Note that the set Rep(T) in this and most other cases is infinite.

Example 1. T∈Tab(SUPPLIER PART PROJECT)

T_C(SUPPLIER PART PROJECT Con)

Jones	x	steel	x=bolt∨x=nut
Smith	y	z	true

T_G = {true, x=y, z=concrete}

{r_1, r_2} ⊆ Rep(T)

r_1(SUPPLIER PART PROJECT)

Jones	bolt	steel
Smith	bolt	concrete

r_2(SUPPLIER PART PROJECT)

Smith	nail	concrete

Given two tables T_1 and T_2 over the same scheme, we say that T_2 contains T_1, den. $T_1 \sqsubseteq T_2$, if Rep(T_1) ⊆ Rep(T_2), and that T_1 is equivalent to T_2, den. $T_1 \equiv T_2$, if $T_1 \sqsubseteq T_2$ and $T_2 \sqsubseteq T_1$. The equivalence of two tables means that they define the same set of relations, since we clearly have $T_1 \equiv T_2$ if Rep(T_1) = Rep(T_2). Also, v(T_{1_C}) = v(T_{2_C}) and v(T_{1_G}) = v(T_{2_G}) for all valuations v necessitates $T_1 \equiv T_2$.

A condition g (or a set of conditions T_G) implies a condition g', den. g ⇒ g', if v(g) =

true implies $v(g') =$ true (or $v(T_G) =$ true implies $v(g') =$ true) for all valuations v. Mutual implication is denoted by $\simeq$ (equivalence). The following normalization rules can now be applied to tables.

(1) If for some $t \in T_C$, $\neg(T_G \wedge t(Con))$, then T_C is replaced by $T_C - \{t\}$.

(2) If there exists c-tuples $t_1, \ldots, t_k \in T_C$, $T \in Tab(R)$, such that $t_1(R) = \ldots = t_k(R)$, then T_C is replaced by $T_C - \{t_1, \ldots, t_k\} \cup \{t\}$, where $t(R) = t_1(R)$ and $t(Con) = \bigvee_{i=1}^{k} t_i(Con)$.

A table T is said to be normalized, den. T^o, if none of these normalization rules can be applied to T. (There are also other normalization rules, but these two are sufficient for our purposes.) Normalization of T does not affect Rep(T). We have $T \equiv T^o$. Replacing conditions in c-tuples or in T_G by equivalent ones also preserves equivalence.

For some purposes we need a form of set inclusion for tables. This modified inclusion is defined followingly:

Definition 2. Let T_1 and T_2 be two tables from Tab(R). $T_1 \underset{\mp}{\subseteq} T_2$, T_2 m-includes T_1 if

(1) for each $g \in T_{1_G}$ there exists a $g' \in T_{2_G}$ such that $g \simeq g'$, and

(2) for each $t_1 \in T_{1_C}$ there exists a $t_2 \in T_{2_C}$ such that $t_1(R) = t_2(R)$ and $t_1(Con) \simeq t_2(Con)$.

If both $T_1 \underset{\mp}{\subseteq} T_2$ and $T_2 \underset{\mp}{\subseteq} T_1$, we say that T_1 is m-equal to T_2 and denote it $T_1 \mp T_2$. Clearly m-equality implies equivalence, i.e. $T_1 \mp T_2$ implies $T_1 \equiv T_2$, but note that $T_1 \underset{\mp}{\subseteq} T_2$ does not imply $T_1 \sqsubseteq T_2$.

3. EXTENSION OF THE RELATIONAL ALGEBRA

As mentioned in the introduction the correctness criteria for an extension of the algebra to a table T is that we only conclude information that holds in every possible state of the real world, i.e. in every relation in Rep(T). If we denote the extension of a relational expression f by $\hat{f}$ we can formalize the correctness criteria as

$$f(Rep(\mathcal{T})) = Rep(\hat{f}(\mathcal{T}))$$

for all relational expressions f and multitables $\mathcal{T}$. The lefthand side of the equality stands for $\{f(r_1, \ldots, r_n) \mid <r_1, \ldots, r_n> \in Rep(\mathcal{T})\}$, and the multitable $\mathcal{T}$ is taken as $<T_1, \ldots, T_n>$ of the tables T_i that $\hat{f}$ is applied on. The result $\hat{f}(\mathcal{T})$ is a table of appropriate type. Our notation reveals that we perform non-unary operations only between tables with the same set of general conditions, i.e. between individual tables of a multitable.

Imielinski and Lipski [IL2] have given a correct extension of the relational algebra for c-tables. The same extension can with a slight modification be used for tables also. Since we in the sequel only will need PJ-expressions we will be contended with giving only the definitions for the extension of the project and join operators.

Definition 3. For a table $T \in Tab(R)$ the projection $\hat{\pi}_X(T)$ on a set $X \subseteq R$ is a table $T'^o \in Tab(X)$ such that $T'_C = \{t(X \cup Con) \mid t \in T_C\}$ and $T'_G = T_G$.

Definition 4. For tables $T_1 \in Tab(R_1)$ and $T_2 \in Tab(R_2)$ with the same set of general conditions the join $T_1 \hat{*} T_2$ of T_1 and T_2 is a table $T'^o \in Tab(R_1 \cup R_2)$ such that $T'_G = T_{1_G}$ and $T'_C = \{t_1 \hat{*} t_2 \mid t_1 \in T_{1_C}$ and $t_2 \in T_{2_C}\}$, where $t_1 \hat{*} t_2$ is the c-tuple over $R_1 \cup R_2$ with

$$(t_1 \hat{*} t_2)(A) = \begin{array}{ll} t_1(A), & \text{if } A \in R_1 \\ t_2(A), & \text{if } A \in R_2 - R_1 \end{array}$$

$$(t_1 \hat{*} t_2)(\mathrm{Con}) = t_1(\mathrm{Con}) \wedge t_2(\mathrm{Con}) \wedge \bigwedge_{A \in R_1 \cap R_2} (t_1(A)=t_2(A)).$$

As an illustration, consider

Example 2. T_{1_C} $\begin{array}{cccl} A & B & C & \mathrm{Con} \\ \hline a & x & c & \neg(x=b) \\ a' & b & y & \underline{true} \end{array}$ T_{2_C} $\begin{array}{cccl} C & D & E & \mathrm{Con} \\ \hline c & d & e & \underline{true} \\ z & d' & e' & \overline{\neg(z=c'')} \\ c' & d' & e & \underline{true} \end{array}$

$$T_{1_G} = T_{2_G} = \{\underline{true}, \neg(y=c)\}$$

$$(T_1 \hat{*} T_2)_C \begin{array}{cccccl} A & B & C & D & E & \mathrm{Con} \\ \hline a & x & c & d & e & \neg(x=b) \wedge \underline{true} \wedge c=c \\ a & x & c & d' & e' & \neg(x=b) \wedge \overline{\neg(z=c'')} \wedge c=z \\ a' & b & y & d' & e' & \underline{true} \wedge \neg(z=c'') \wedge y=z \\ a' & b & y & d' & e & \underline{true} \wedge \underline{true} \wedge y=c' \end{array}$$

$$(T_1 \hat{*} T_2)_G = \{\underline{true}, \neg(y=c)\}$$

The conditions of the first and last c-tuples of $(T_1 \hat{*} T_2)_C$ can be replaced by $\neg(x=b)$ and $y=c$. Also, the un-normalized join of T_1 and T_2 contains the c-tuple $\langle a', b, y, d, e, \underline{true} \wedge \underline{true} \wedge y=c \rangle$, but since $(\neg(y=c) \wedge y=c) \simeq \underline{false}$ the tuple is removed. For a similar reason the c-tuple $\langle a, x, c, d', e, \neg(x=b) \wedge \underline{true} \wedge c=c' \rangle$ does not belong to the normalized result.

The rest of the relational operators can be extended to operate on tables along the same lines as the project and join operators, and the following theorem follows easily from Theorem 9.2 of [IL2].

Theorem 1. For any well formed expression f built up from projection, selection, union, join and difference, and multitables $\mathcal{T}$, we have $f(\mathrm{Rep}(\mathcal{T})) = \mathrm{Rep}(\hat{f}(\mathcal{T}))$.□

In the ordinary relational algebra a relation r is always included in the result of certain PJ-expressions on r. A similar property will be needed for the extended algebra.

Lemma 1. Let $\mathcal{R} = R_1, \ldots, R_p$, $\bigcup_{i=1}^{p} R_i = R$ and $T \in \mathrm{Tab}(R)$. Then $T \underset{\mp}{\subset} \hat{\pi}_{R_1}(T) \hat{*} \ldots \hat{*} \hat{\pi}_{R_p}(T)$.

Proof. Property (1) of Definition 2 is immediate since T_G is not changed. For property (2), take an arbitrary $t \in T_C$. By Definition 3 $t(R_i \cup \mathrm{Con}) \in \hat{\pi}_{R_i}(T)$, for $i=1,\ldots,p$. By inductive use of Definition 4 $\hat{\pi}_{R_1}(T) \hat{*} \ldots \hat{*} \hat{\pi}_{R_p}(T)$ will include a c-tuple t' with $t'(R) = t(R)$ and $t'(\mathrm{Con}) =$
$(t(\mathrm{Con}) \wedge t(\mathrm{Con}) \wedge \bigwedge_{A \in R_1 \cap R_2} (t(A)=t(A)) \wedge \ldots \wedge t(\mathrm{Con}) \wedge \bigwedge_{A \in R_{p-1} \cap R_p} t(A)=t(A)))$. Now $t'(\mathrm{Con})$ is clearly equivalent to $t(\mathrm{Con})$, so the result follows.□

4. EXTENSION OF DEPENDENCY SATISFACTION

A set Σ of dependencies restricts the relations that model the real world to those in $\mathrm{Sat}(\Sigma)$. For a table T we make the natural interpretation that T "satisfies" Σ if

$$\mathrm{Rep}(T) \subseteq \mathrm{Sat}(\Sigma).$$

Thus we need a method for transforming T in order to cut down Rep(T) to $\mathrm{Rep}(T) \cap \mathrm{Sat}(\Sigma)$ for any table T and set Σ of FDs and JDs. This cutting down is necessary, since the extended algebra operates on the basis of Rep(T), and the set Σ implies that not necessarily all members of Rep(T) are possible states of the real world.

The method is called a chase, and its idea will be clarifyed by concidering the algebraic counterparts of FDs and JDs. These counterparts are special cases of so called algebraic dependencies of [YP] (see also [Abi]). Stated with a momentary simplification, an algebraic dependency is a statement of the form $f(R) \subseteq R$, where f is a PJ-expression. A relation r over R satisfies such a dependency if $f(r) \subseteq r$. Since we demand the inclusion to hold for all members of Rep(T), where $T \in \mathrm{Tab}(R)$, we can using the definition of containment and Theorem 1 write the following deduction chain:

T "satisfies" $f(R) \subseteq R$ if and only if
$f(Rep(T)) \subseteq Rep(T)$ if and only if
$Rep(\hat{f}(T)) \subseteq Rep(T)$ if and only if
(*) $\hat{f}(T) \sqsubseteq T$

Our chase is essentially a method for enforcing the containment (*) by applying the expression $\hat{f}$ on T. The enforcement will always be possible, even if $Rep(T) \cap Sat(\Sigma) = \emptyset$. There are two reasons for the intersection to be empty. First, it might be that every relation in Rep(T) violates a FD in Σ. Then we must make Rep(T) equal to the empty set. Second, the relations in Rep(T) might violate some JDs of Σ. In this case we will add the necessary c-tuples to T, and thus we have in fact done more than just cutting down Rep(T) to $Rep(T) \cap Sat(\Sigma)$. We shall be more precise after introducing some necessary definitions.

First we concider completions of relations: Let $r \in Rel(R)$. The completion of r with respect to Σ is $Comp_\Sigma(r) = s$, such that $s \supseteq r$ and $s \in Sat(\Sigma_{JD})$, and there is no relation $s' \in Sat(\Sigma_{JD})$ with $s \supsetneq s' \supseteq r$. $Comp_\Sigma(Rep(T)) = \{Comp_\Sigma(r) \mid r \in Rep(T)\}$. The completion means that the tuples that are necessary for making r satisfy Σ_{JD} are added in s. The completion always exists and it is unique (see [Mai, exercise 8.41]). We are also free to choose any cover for Σ, i.e. if $Sat(\Sigma) = Sat(\Sigma')$ then $Comp_\Sigma(Rep(T)) = Comp_{\Sigma'}(Rep(T))$.

For the definition of algebraic dependencies we need the notions of extended schemes and relations. An extended relation scheme $\overline{R}$ of R has two copies of every attribute. For instance, if R = ABCD then $\overline{R}$ = ABCDABCD. An extended relation $\overline{r}$ of r over R is accordingly $\{<t,t> \mid t \in r\}$. The two different copies of an attribute will be distinguished by the subscripts 1 and 2. We make the same kind of extensions to tables also, except that the conditions are not repeated. Thus an extended table for $T \in Tab(R)$ is a table $\overline{T} \in Tab(\overline{R})$ with $\overline{T}_C = \{<t(R),t(R),t(Con)> \mid t \in T_C\}$ and $\overline{T}_G = T_G$. An example will clarify the point.

Example 3. T_C(A B C Con) $T_G = \{$true, $\neg(y=c)\}$

A	B	C	Con
a	x	c	$\neg(x=b)$
a	b	y	true

$\overline{T}_C(A_1B_1C_1A_2B_2C_2Con)$ $\overline{T}_G = \{$true, $\neg(y=c)\}$

A_1	B_1	C_1	A_2	B_2	C_2	Con
a	x	c	a	x	c	$\neg(x=b)$
a	b	y	a	b	y	true

An algebraic dependency [YP] is a statement $f(\overline{R}) \subseteq f'(\overline{R})$, where f and f' are PJ-expressions. We will however only need the counterparts of FDs and JDs. The FD X->A for relations over R corresponds to the algebraic dependency

$$\pi_{A_1A_2}(\pi_{X_1A_1}(\overline{R}) * \pi_{X_1A_2}(\overline{R})) \subseteq \pi_{A_1A_2}(\overline{R}).$$

The JD $*[R_1,\ldots,R_p]$, $\bigcup_{i=1}^{p} R_i = R$, corresponds to to the algebraic dependency

$$\pi_{R_1}(R) * \ldots * \pi_{R_p}(R) \subseteq R.$$

Here correspondence means that the sets Σ of FDs and JDs and Σ' of the corresponding algebraic dependencies express the same constraints, i.e. that $Sat(\Sigma) = Sat(\Sigma')$. The algebraic counterpart of a JD should not raise anyones eyebrows. For the counterpart of a FD we give the following example of a relation that does not satisfy the dependency A->B.

Example 4. r(A B) $\overline{r}(A_1B_1A_2B_2)$

A	B
a	b
a	b'

A_1	B_1	A_2	B_2
a	b	a	b
a	b'	a	b'

$\pi_{B_1B_2}(\overline{r})(B_1B_2)$

B_1	B_2
b	b
b'	b'

$\pi_{B_1B_2}(\pi_{A_1B_1}(\overline{r}) * \pi_{A_1B_2}(\overline{r}))(B_1B_2)$

B_1	B_2
b	b
b	b'
b'	b'
b'	b

The chase procedure can now be described with the following algorithm.

```
Algorithm Chase
Input:  A Table $T \in Tab(R)$ and a set $\Sigma$ of FDs and
        JDs for R
Output: A table T' such that
        $Rep(T') = Comp_{\Sigma}(Rep(T)) \cap Sat(\Sigma)$
Method: $T' \leftarrow T$
        repeat foreach $*[R_1,\ldots,R_p] \in \Sigma$ do
                $T' \leftarrow \hat{\pi}_{R_1}(T') \hat{*} \ldots \hat{*} \hat{\pi}_{R_p}(T')$
        until  the number of c-tuples in $T'_C$
                no longer increases
        foreach $X \rightarrow A \in \Sigma$ do begin
                $T'' \leftarrow \hat{\pi}_{A_1A_2}(\hat{\pi}_{X_1A_1}(\overline{T'}) \hat{*} \hat{\pi}_{X_1A_2}(\overline{T'}))$
                foreach $t \in T''_C$ do
                if $t(A_1) \neq t(A_2)$ then
                    if $t(A_1)$ and $t(A_2)$ both
                        are constants
                    then $T'_G \leftarrow T'_G \cup \{\neg t(Con)\}$
                    else $T'_G \leftarrow T'_G \cup \{t(Con) \Rightarrow$
                                        $t(A_1) = t(A_2)\}$
        end
        end of algorithm chase.
```

Some examples of the chase can be found in the end of this chapter. We now proceed to prove the correctness and some properties of the chase.

Theorem 2. For a given set of FDs and JDs and a table T, the chase algorithm only requires a finite number of steps and the resulting table T' is finite.

Proof. The algorithm does not introduce new variables or constants, so the resulting table T' is finite. Furthermore, each time we perform the *repeat-until* loop the number of c-tuples in T'_C increases with at least one (except of course during the last loop). The loop is thus performed only a finite number of times. Since each FD is applied only once, the whole algorithm terminates after a finite number of steps.$\square$

Different choices of the order for applying the JDs will result in different tables. In the full version of this paper we prove that these tables are m-equal, so the order is of no significance. Usually we will let T' denote the result of chasing T with some arbitrary order.

The next lemma gives the relationship between T and T'.

Lemma 2. Let T' be the result of chasing a table T with a set of FDs and JDs. Then $T \mathrel{\underset{\cdot}{\subseteq}} T'$.

Proof. The *repeat-until* loop is a repetitive application of PJ-expressions fullfilling the conditions for Lemma 1. Thus the result holds at this stage. The FDs only cause adding of conditions to T_G, and hence $T_G \subseteq T'_G$ and consequently $T \mathrel{\underset{\cdot}{\subseteq}} T'$.$\square$

The main theorem states that the resulting table T' has the desired property. The proof of the theorem is given in the full version of this paper.

Theorem 3. Let T' be the result of performing the chase on a table $T \in Tab(R)$ with a set Σ of FDs and JDs. Then $Rep(T')=Comp_{\Sigma}(Rep(T)) \cap Sat(\Sigma)$.$\square$

In the chase of [MMS] any cover for the set of dependencies can be used. The results will be identical. In our chase we will get equivalent tables, which is sufficient for our purposes.

Corollary. If T' is the result of performing the chase with a set Σ of FDs and JDs on a table T, and T'' is the result when using a set Σ', where $Sat(\Sigma) = Sat(\Sigma')$, then $T' \equiv T''$.

Proof. By Theorem 3 and the definition of completions $Rep(T') = Comp_{\Sigma}(Rep(T)) \cap Sat(\Sigma) = Comp_{\Sigma'}(Rep(T)) \cap Sat(\Sigma') = Rep(T'')$.$\square$

Before closing this section we will give two small examples of the chase. These examples show the kind of information that has not been deducable by previously known chase methods (i.e. [IL3, MMS, Vas2]). T' will as usual denote the result of chasing T with Σ.

Example 5. $\Sigma = \{A \to B\}$

T_C

A	B	Con
a	b	true
x	b'	true

$T_G = \{\text{true}\}$

T'_C

A	B	Con
a	b	true
x	b'	true

$T'_G = \{\text{true}, \neg(x{=}a)\}$

Example 6. $\Sigma = \{*[AB, AC]\}$

T_C

A	B	C	Con
a	b	c	true
x	b'	c'	true

$T_G = \{\text{true}\}$

T'_C

A	B	C	Con
a	b	c	true
x	b'	c'	true
x	b	c'	$x{=}a$
x	b'	c	$x{=}a$

$T'_G = \{\text{true}\}$

In Example 5 we are able to express the fact that x cannot equal a. Example 6 is perhaps more interesting. Any relation in $\text{Comp}_\Sigma(\text{Rep}(T)) \cap \text{Sat}(\Sigma)$ should include two or four tuples, depending on the value of x. The table T' expresses exactly this and the content and conditions of the extra tuples. The conclusion is that the expressive power of our tables is required for properly handling data dependencies. These two examples can in fact be used to show that the usual device for representing null-values, in [IL2, IL3] called v-tables (ordinary relations with variables), is not capable of fully supporting data dependencies. That is, there are v-tables T and dependencies Σ such that there exists no v-table T' for which $\text{Rep}(T') = \text{Comp}_\Sigma(\text{Rep}(T)) \cap \text{Sat}(\Sigma)$. The problem is also noted by [IL3].

5. CONCLUSIONS

We have presented a relational system for handling null-values of type "value exists, but is unknown". The key idea is that a relation with null-values, here modelled by a so called table, represents a set of relations, one of which corresponds to the incompletely known state of the real world. The formal criteria for the correctness of an algebra that operates on tables is that only information that holds in every relation in the represented set is inferred. We have correctly extended the complete relational algebra to operate on tables.

The main contribution of this paper however lies in the capability of the system to support functional and full join dependencies. The formal criteria for a table to "satisfy" a set of dependencies is that the dependencies are satisfied in every relation in the set that the table represents. This strong form of satisfaction is required for maintaining the correctness of the algebra in the precence of dependencies. We have then presented a transformation algorithm, called a chase, that enforces a given set of dependencies on a table in such a way that the dependencies are satisfied in every relation in the set represented by the transformed table. This chase algorithm can also be regarded as a generalization of previously known chase methods, and we have given some examples of the kind of information that only our novel chase is able to deduce. Our results also show that the expressive power of our tables is required for fully supporting data dependencies in databases with incomplete information.

REFERENCES

Abi S. Abiteboul, Algebraic analogues to fundamental notions of query and dependency theory. Rapports de Recherche 201. Institut National de Recherche en Informatique et en Automatique. Roquencourt, April 1983.

Bis1 J. Biskup, A formal approach to null values in database relations. In: Advances in Data Base Theory - Vol. 1 J. Minker & J. M. Nicolas (eds.) Plenum Press, New York London 1981, 299-341.

Bis2 J. Biskup, A foundation of Codd's relational maybe operations. ACM Transactions on Database Systems 8,4 (Dec. 1983), 608-636.

FUV R. Fagin, J. D. Ullman & M. Y. Vardi, On the semantics of updates in databases. Proc. of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems 1983, 352-365.

Gra1 G. Grahne, Information incompleteness in databases. In: Proc. of the Winter School on Theoretical Computer Science. R. Back et al (eds.). Finnish Society of Information Processing Science, Lammi, January 1984, 65-98.

Gra2 G. Grahne, Updates in databases with incomplete information. In preparation.

IL1 T. Imielinski & W. Lipski, On representing incomplete information in a relational database. Proc. of the Sixth International Conference on Very Large Data Bases 1981, 388-397.

IL2 T. Imielinski & W. Lipski, Incomplete information in relational databases. To appear in J. ACM.

IL3 T. Imielinski & W. Lipski, Incomplete information and dependencies in relational databases. Proc. of the ACM SIGMOD Conference on Management of Data 1983, 178-184.

Lie E. Lien, Multivalued dependencies with null values in relational databases. Proc. of the Fifth International Conference on Very Large Data Bases 1979, 61-66.

Lip W. Lipski, Logical problems related to incomplete information in databases. Rapport de Recherche 138. Laboratoire de Recherche en Informatique, Universite de Paris-Sud, September 1983.

Mai D. Maier, The Theory of Relational Databases. Pitman, London 1983.

MMS D. Maier, A. O. Mendelson & Y. Sagiv, Testing implications of data dependencies. ACM Transactions on Database Systems 4,4 (Dec. 1979), 455-469.

Rei R. Reiter, On closed world databases. In: Logic and Databases, H. Gallaire & J. Minker (eds.). Plenum Press, New York 1978, 55-76.

Sci E. Sciore. The universal instance and database design. Ph.D. Thesis. Princeton University 1980.

Ull J. D. Ullman, Principles of Database Systems (Second Edition). Computer Science Press, Potomac, Md. 1982.

Vas1 Y. Vassiliou, Null values in database management: a denotational semantics approach. Proc. ACM SIGMOD 1979 Conference on Management of Data, 162-169.

Vas2 Y. Vassiliou, Functional dependencies and incomplete information. Proc. of the Sixth International Conference on Very Large Data Bases 1981, 260-269.

YP M. Yannakakis & C. H. Papadimitriou, Algebraic dependencies. Journal of Comp. and Syst. Sci. 25,1 (1982), 2-41.