A Characterization of Globally Consistent Databases and Their Correct Access Paths

YEHOSHUA SAGIV
University of Illinois at Urbana-Champaign

The representative instance is proposed as a representation of the data stored in a database whose relations are not the projections of a universal instance. Database schemes are characterized for which local consistency implies global consistency. (Local consistency means that each relation satisfies its own functional dependencies; global consistency means that the representative instance satisfies all the functional dependencies.) A method of efficiently computing projections of the representative instance is given, provided that local consistency implies global consistency. Throughout, it is assumed that a cover of the functional dependencies is embodied in the database scheme in the form of keys.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; H.2.1 [**Database Management**]: Logical Design—*normal forms, schema and subschema*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation*

General Terms: Algorithms, Design, Languages, Theory

Additional Key Words and Phrases: Chase, extension join, functional dependency, null value, relational algebra, relational database, representative instance, universal relation scheme

1. INTRODUCTION

The universal instance assumption is essential to many papers in design theory for relational databases. As pointed out in [11], two different concepts are included in this assumption. The most basic concept is the *universal relation scheme assumption* (also known as the *uniqueness assumption* [6]). It asserts that each attribute has a unique role, that is, for any subset of attributes X, there is (at most) one relationship among the attributes of X. This assumption is made explicitly or implicitly in many papers in design theory for relational databases. In particular, it is made in papers dealing with the axiomatization of dependencies and with synthesis and decomposition of relation schemes.

The second and more controversial concept is the *universal instance assumption*, that is, the assumption that the relations of a database are the projections of a single relation over the set of all the attributes. This assumption is needed in order to define lossless joins [1]. There are two versions of this assumption.

This work was supported in part by the National Science Foundaton under Grant MCS-80-03308.
Author's present address: Institute of Mathematics and Computer Science, The Hebrew University of Jerusalem, Givat Ram 91904, Jerusalem, Israel.

According to the first, this assumption has to be made only in order to determine whether a join is lossless, that is, it is only a tool that is used when a database is designed and when queries are evaluated [11]. The second version (the *pure* universal instance assumption) states that the relations of a database must always be the projections of a universal instance, and null values have to be used in order to satisfy this requirement [14, 15, 17, 18, 24].

When a universal instance is assumed, users can formulate queries having in mind the universal instance rather than the actual relations of the database [16]. If a given query refers to a set of attributes X, then the first step in evaluating this query is to compute the projection of the universal instance onto X [21]. When the pure universal instance assumption is made, it is sufficient to take any lossless join over a set of attributes that contains X. However, if the relations of the database are not the projections of a universal instance, different lossless joins might give different results [21]. In [11] this problem is solved by requiring that the join dependency consisting of all the relation schemes be acyclic. In this paper we propose an alternative solution. We believe that our solution reflects the properties of functional dependencies better than the solution given in [11]. In particular, some of the problems left open in [11] are handled in our case without explicitly defining maximal objects [20].

In this paper we assume a universal relation scheme, but not a pure universal instance. Instead we define the *representative instance* of a database (see Section 3). The representative instance has been used to determine whether the database satisfies a set of functional dependencies [13, 26]. We believe that the representative instance correctly describes the information stored in the database even when the relations are not the projections of a universal instance. As a first step toward evaluating queries with respect to the representative instance, we address the following problems.

(1) Under what conditions is the database globally consistent if each relation is locally consistent? That is, under what conditions does the representative instance satisfy all the functional dependencies if each relation satisfies its own functional dependencies?

(2) How can we efficiently compute projections of the representative instance?

Throughout this paper we assume that a cover of the functional dependencies is embodied in the database scheme in the form of keys (as in [6]). In Section 3 we define the *uniqueness condition*, and in Section 4 we prove that local consistency implies global consistency if and only if the database scheme satisfies the uniqueness condition. In Section 5 we show how to efficiently compute projections of the representative instance provided that the database scheme satisfies the uniqueness condition.

2. PRELIMINARIES

2.1 Basic Definitions

We view a *relation* (cf., [9]) as a finite table with columns, labeled *attributes*, and rows, called *tuples*, that represent mappings from the attributes to their associated *domains*. Let μ be a tuple of a relation labeled by a set of attributes R. If $A \in R$, then $\mu(A)$ is the value of μ in column A; if $S \subseteq R$, then $\mu[S]$ denotes the values of μ for the attributes in S. We say that r is a relation *over* a set of attributes R if the columns of r are labeled by the attributes of R.

We use letters from the beginning of the alphabet (A, B, C, ...) to denote attributes, and letters from the end of the alphabet (..., X, Y, Z) to denote sets of attributes. A string of attributes (e.g., $ABCD$) denotes the set containing these attributes, and the union of two sets X and Y is written XY.

A *relational database scheme* over a set of attributes U is a set of ordered pairs $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ such that $R_i \subseteq U$ and K_i is a set of (explicit) *keys* of R_i. Each R_i is a set of attributes labeling the columns of a relation, and we use it as the name of the relation (i.e., there are no distinct relations over the same set of attributes). We call each R_i a *relation scheme.* The set of FDs (functional dependencies) that are *embodied* in R_i is

$$F_i = \{X \rightarrow R_i - X \mid X \in K_i\}.$$

We impose the following two conditions on the set of keys K_i for R_i. First, each R_i has at least one key (which may be R_i itself). Second, no K_i can have two distinct keys X and Y such that $X \subseteq Y$.

A *database* is a set of relations $r_1, \ldots, r_n$ over $R_1, \ldots, R_n$, respectively, such that each r_i satisfies F_i. That is, a database is the "current value" of the database scheme. We assume that $F = \bigcup_{i=1}^{n} F_i$ is a cover of all the FDs imposed on the database by the user. In other words, a cover of the FDs is embodied in the database scheme (in the form of (explicit) keys) as in [6].

The assumption that the FDs are embodied as keys is well justified. It is easier to enforce an FD that follows from a key than an FD whose left side is not a key. Consequently, we prefer a design in which all the FDs follow from the keys of the relation schemes. A priori, we do not require that F_i be a cover of all the FDs of F^+ (i.e., the closure of F) that are defined over the attributes of R_i. Hence, the relation schemes are not necessarily in any normal form, and we can always synthesize a database scheme that embodies a cover of the given FDs [5]. However, the database schemes that satisfy the uniqueness condition are also in Boyce-Codd normal form.

2.2 Relations with Null Values

In many cases there is a need to represent partial information in the database. If we have a relation over the attributes Manager and Department, and Jones is a manager without a department, then the tuple (Jones, δ) is inserted into this relation. The value δ is a special value, called a *null value,* and it denotes unknown information. Suppose that there are two managers without a department, for example, Jones and Smith. There is no reason to assume that they manage the same (unknown) department. In order to distinguish the null value in the tuple (Jones, δ) from the null value in the tuple (Smith, δ), we will mark each null value with a unique subscript and store the tuples (Jones, δ_1) and (Smith, δ_2). Null values with distinguishing subscripts are called *marked nulls* [15, 18] and are used exclusively in this paper. Two null values are equal only if they have the same subscript. We say that tuples μ_1 and μ_2 *agree* on column A, written $\mu_1(A) = \mu_2(A)$, if either both $\mu_1(A)$ and $\mu_2(A)$ are not null and equal or both are null and equal.

2.3 The Chase Process

Suppose that a relation r (with null values) is required to satisfy an FD $X \rightarrow Y$. It is not necessarily correct to argue that r violates $X \rightarrow Y$ if it has two tuples that agree on X and disagree on some columns of Y. Instead, we associate with $X \rightarrow Y$ the following FD rule for equating symbols[1] of r.

FD rule (for $X \rightarrow Y$). Suppose that r has tuples μ_1 and μ_2 that agree on all the columns for X but disagree on some columns of Y. Then for all columns A in Y such that $\mu_1(A) \neq \mu_2(A)$,

(1) if μ_1 has δ_i in column A and μ_2 has δ_j in column A, then replace all occurrences of δ_j in r with δ_i, and
(2) if μ_1 has a nonnull value c in column A and μ_2 has a null value δ_j in that column, then replace all occurrences of δ_j with the nonnull value c.

Suppose that the relation r is required to satisfy a set F of FDs. We can apply the FD rules for F to r until no more symbols of r can be equated. The relation obtained in this way is called the *chase* of r with respect to F, written $\text{chase}_F(r)$, and it satisfies an FD $X \rightarrow Y$ of F if and only if there is no pair of tuples that agree on X and disagree on some columns of Y. We say that the relation r *satisfies* F if and only if $\text{chase}_F(r)$ satisfies F. If r satisfies F, then $\text{chase}_F(r)$ is unique up to renaming of null values [19].

Example 1: *Part A*. Suppose that $F = \{A \rightarrow C, A \rightarrow D, B \rightarrow C, CD \rightarrow B\}$ and let r be the following relation:

A	B	C	D
1	1	δ_1	δ_2
δ_3	2	1	δ_4
1	δ_5	δ_6	2
2	1	δ_7	1

We apply the FD rule for $A \rightarrow C$ to the first and third tuples to replace δ_6 with δ_1. Then all occurrences of δ_1 are replaced with δ_7 by applying the FD rule for $B \rightarrow C$ to the fourth and first tuples. The result is the following relation:

A	B	C	D
1	1	δ_7	δ_2
δ_3	2	1	δ_4
1	δ_5	δ_7	2
2	1	δ_7	1

By applying the FD rule for $A \rightarrow D$ to the first and third tuples, δ_2 is replaced with 2. Now δ_5 is replaced with 1 by applying the FD rule for $CD \rightarrow B$ to the first

[1] Occasionally, we refer to null and nonnull values as symbols.

and third tuples. Since no more applications are possible, $\text{chase}_F(r)$ is

A	B	C	D
1	1	δ_7	2
δ_3	2	1	δ_4
1	1	δ_7	2
2	1	δ_7	1

Clearly, $\text{chase}_F(r)$ (and hence r) satisfies F.

Example 1: *Part B.* Suppose that we add the *FD* $C \rightarrow D$ to F in Part A. Then $\text{chase}_F(r)$ is no longer unique, since δ_2 can be replaced with either 1 or 2, and $C \rightarrow D$ is not satisfied by $\text{chase}_F(r)$ (and r).

If r satisfies F, then r also satisfies additional FDs that can be inferred by Armstrong's axioms [4]. The *closure* of a set of attributes X with respect to a set of FDs F, written X_F^+, is the set of all attributes A such that $X \rightarrow A$ can be derived from F by Armstrong's axioms. We can compute X_F^+ in linear time [5]. If F denotes a cover of all the FDs imposed on the database (i.e., the embodied FDs), then we usually write X^+ instead of X_F^+.

2.4 Relational Expressions and Extension Joins

In this paper we consider relational expressions over the operators *project*, (natural) *join*, and *union* (denoted by π, $\bowtie$, and $\cup$, respectively). The operands are the relation schemes $R_1, \ldots, R_n$. Let α be a set of relations $r_1, \ldots, r_n$ for the relation schemes $R_1, \ldots, R_n$, respectively, such that each r_i satisfies F_i. The value of E for α, written $v_\alpha(E)$, is computed by substituting the relations $r_1, \ldots, r_n$ for the relation schemes $R_1, \ldots, R_n$, and applying the operators according to the usual definitions. (When the join is applied, two tuples are joined on a given column only if they agree in this column.) Two expressions E_1 and E_2 are *equivalent*,[2] written $E_1 \equiv E_2$, if for all sets α, $v_\alpha(E_1) = v_\alpha(E_2)$. The expression E_2 is *contained* in E_1, written $E_2 \subseteq E_1$, if for all sets α, $v_\alpha(E_2) \subseteq v_\alpha(E_1)$. An expression E has a unique value for the current database; therefore by a slight abuse of notation we denote this value by E (rather than $v_\alpha(E)$, where α is the set of current relations). In particular, if μ is a tuple in the value of E for the current database, then we write $\mu \in$ E.

The expression $\bowtie_{j=1}^{m} R_{i_j}$ is an *extension join* [12] of R_{i_1} if for all $1 \leq j < m$, the set $R_{i_1} \ldots R_{i_j}$ contains a key of $R_{i_{j+1}}$. We say that the extension join $\bowtie_{j=1}^{m} R_{i_j}$ is *over* the set of attributes $R_{i_1} \ldots R_{i_m}$. Extension joins are a special case of lossless joins [1].

Example 2. Let $\langle ABC, \{A\}\rangle$, $\langle BD, \{B\}\rangle$, $\langle CDE, \{CD\}\rangle$ be a database scheme. $ABC \bowtie BD$ is an extension join of ABC. Similarly, $ABC \bowtie BD \bowtie CDE$ is an extension join of ABC, since ABC contains the key B of BD, and $ABCD$ contains the key CD of CDE. $ABC \bowtie CDE$ is not an extension join, since ABC

[2] This notion of equivalence is called *strong equivalence* in [3].

does not contain any key of CDE (i.e., ABC does not contain CD which is the only key of CDE).

3. The Representative Instance

The ultimate goal of designing a database scheme has always been a collection of relation schemes $R_1, \ldots, R_n$ that are independent, that is, relation schemes that allow the user to update each relation in the database without having to change the contents of the other relations. Of course, there might be some semantically meaningful constraints (e.g., as in [10]) that do not allow every possible update. But these constraints should be as limited as possible. We feel that enforcing the universal instance assumption is too restrictive. Clearly, this assumption can always be enforced by using marked nulls [16, 18]. However, this can be done only at the expense of applying the chase process to the universal instance whenever updates are performed on the database. Furthermore, we have to store many null values that do not provide any information. These null values are needed only to satisfy the universal instance assumption.

Efficiency is not the only issue. Most relational database systems are not designed to use the chase process. Therefore, there is a need to develop a theory for determining correct access paths when the universal instance assumption is not satisfied. The simplified universal instance assumption of [11] is one possible solution. Only acyclic database schemes are considered by [11] and it argues that queries can be evaluated by applying tableau optimization [2] as if there were a universal instance. We require that database schemes satisfy the uniqueness condition (defined later), and we show that queries should be evaluated by performing the union of several lossless joins rather than just one lossless join. The class of acyclic database schemes and the class of database schemes satisfying the uniqueness condition are not comparable; that is, there are acyclic database schemes that do not satisfy the uniqueness condition, and some database schemes that satisfy the uniqueness condition are cyclic. Whether a database scheme is acyclic depends only on the set of attributes of each relation scheme. In contrast, the definition of the uniqueness condition considers both the set of attributes and the set of FDs of each relation scheme.

In this section we define the *representative instance* [13, 25] of a database $r_1, \ldots, r_n$. The representative instance is defined for every $r_1, \ldots, r_n$ (even if the r_i's are not the projections of a single relation). If $r_1, \ldots, r_n$ are the projections of a universal instance r and the database scheme has the lossless join property, then r is also the representative instance. In [13, 26] the representative instance is used to determine whether the database satisfies a set of FDs. We believe that the representative instance is also a correct representation of all the information stored in the database, and queries posed about the contents of the database should be answered with respect to the representative instance.

Let U be the set of all the attributes. A relation r_i can be viewed as a relation over U by adding columns for the attributes in $U - R_i$ that contain distinct null values. Formally, the *augmentation* of a relation r_i over R_i to a relation over U, written $\alpha_U(r_i)$, is a minimal relation satisfying the following definition: $\{\mu \mid \mu$ agrees with some tuple of r_i on R_i, and has distinct null values (that do not appear in any other tuple) for the attributes of $U - R_i\}$.

Example 3. Let r be the relation

A	C
c_1	c_2
c_3	c_4

A relation satisfying the definition of $\alpha_{ABCD}(r)$ is

A	B	C	D
c_1	δ_1	c_2	δ_2
c_3	δ_3	c_4	δ_4

where $\delta_1 \notin \{c_1, c_2, c_3, c_4\}$ for every i.

Consider a database $r_1, \ldots, r_n$ over relation schemes $R_1, \ldots, R_n$, and let $r' = \bigcup_{i=1}^{n} \alpha_U(r_i)$. If there are no dependencies, then r' is the *representative instance* of the database $r_1, \ldots, r_n$. When dependencies are present, the chase process should be applied to r'. Thus, if the only dependencies are those in the set of FDs F, then the representative instance is $\text{chase}_F(r')$. The database $r_1, \ldots, r_n$ *satisfies* the set of FDs F if the representative instance satisfies F [13, 26].

Example 4: *Part A*. Consider the database scheme $\langle ABCD, \{A\}\rangle$, $\langle CGDEF, \{CG\}\rangle$, $\langle DEFB, \{DEF\}\rangle$, $\langle BCF, \{BC\}\rangle$. Note that this database scheme is in Boyce-Codd normal form. Suppose that the relation for $ABCD$ is $\{1112\}$, the relation for $CGDEF$ is $\{11111\}$, the relation for $DEFB$ is $\{1111\}$, and the relation for BCF is empty. To obtain the representative instance we have to compute the chase of the following relation:

A	B	C	D	E	F	G
1	1	1	2	δ_1	δ_2	δ_3
δ_4	δ_5	1	1	1	1	1
δ_6	1	δ_7	1	1	1	δ_8

The null value δ_5 can be replaced with 1 by applying the FD rule for $DEF \rightarrow B$ to the second and third tuples, and then δ_2 is replaced with 1 by applying the FD rule for $BC \rightarrow F$ to the first and second tuples. No FD rule can be applied after that, and so the representative instance is

A	B	C	D	E	F	G
1	1	1	2	δ_1	1	δ_3
δ_4	1	1	1	1	1	1
δ_6	1	δ_7	1	1	1	δ_8

This representative instance (and hence also the database) satisfies the FDs that are embodied in the relation schemes.

Example 4: *Part B.* Suppose that the relation for BCF in Part A were $\{112\}$ instead of $\varnothing$. Then the tuple

$$(\delta_9, 1, 1, \delta_{10}, \delta_{11}, 2, \delta_{12})$$

would be added to the representative instance, and the representative instance would violate the FD $BC \rightarrow F$.

In both parts of the above example, the projection of the representative instance onto BCF contains the tuple 111. It may be argued that the tuple 111 over the attributes BCF does not represent correct information, since the relation for the relation scheme BCF does not contain this tuple. However, if this argument is accepted, then it follows that two distinct relationships between the attributes B, C, and F are stored in the database. One relationship is stored in the relation for the relation scheme BCF, and the other relationship is obtained by the extension join $CGDEF \bowtie DEFB$. But this is contrary to the universal relation scheme assumption, and without this assumption, many basic results (e.g., the axioms for functional and multivalued dependencies, and the various synthesis and decomposition algorithms) cannot be used. Therefore, we believe that the representative instance correctly represents the information stored in the database.

3.1 Computing Total Projections of the Representative Instance

In the remainder of this paper we consider a database scheme $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ and a corresponding database $r_1, \ldots, r_n$ with a representative instance r. For simplicity's sake, we assume that the relations $r_1, \ldots, r_n$ do not contain null values. (Databases with nulls are considered in the Appendix.) The user formulates queries having in mind the representative instance r rather than the individual relations $r_1, \ldots, r_n$. Suppose that the user is posing a query that refers to a set of attributes X. The first step in evaluating this query is to compute the projection of r onto X. We assume that if the user is referring to the attributes in X, then he is interested only in tuples of r that have nonnull values for all the attributes in X. Therefore, the problem addressed in this paper is how to compute the *X-total projection* of r, that is,

$$\{\mu \mid \mu \text{ is a tuple in } \pi_X(r) \text{ without any null value}\}.$$

For example, given the database of Example 4, Part A, the ACF-total projection of the representative instance is $\{111\}$.

Our approach of computing the X-total projection of the representative instance in response to a query over X can also be supported by the following argument. A universal instance I (without nulls) is a *containing instance* [13] of the database $r_1, \ldots, r_n$ if

(1) I satisfies F, and
(2) for all i we have $r_i \subseteq \pi_{R_i}(I)$.

Thus, a containing instance is a consistent global state that contains all the information in the database (and, possibly, more). The information stored in the database can be viewed as that portion which is common to all the containing instances of the database. But this common portion is exactly what we compute.

Figure 1

```
    begin
(1) $Y := X$;
(2) while there is a key $V \in K_j$ such that $V \subseteq Y$ and $R_j \not\subseteq Y$ do
(3)     $Y := YR_j$
    end
```

Formally, if we project each containing instance onto X and then take the intersection of all these projections, the result is exactly the X-total projection of the representative instance (provided that the representative instance satisfies F).

Example 4 illustrates two surprising facts. First, Part B shows that the representative instance does not necessarily satisfy the functional dependencies even if each relation satisfies the FDs imposed by its keys (this fact was originally pointed out in [13]). Second, an X-total projection of the representative instance cannot always be computed by joining several relations of the database and then projecting onto X. In part A of Example 4, the $ABCDF$-total projection of the representative instance is {11121}. However, no expression of the form $\pi_X(\bowtie_{j=1}^{m} R_{i_j})$ has as its value the relation {11121} over $ABCDF$. (Here, $R_{i_1}, \ldots, R_{i_m}$ are some of the relation schemes.) In the following sections we characterize the cases in which the problems indicated by Example 4 do not occur.

3.2 The Uniqueness Condition

Consider a tuple μ of $\cup_{j=1}^{n} \alpha_U(r_j)$ that has originated from the relation r_i. Tuple μ is nonnull in the columns of R_i and has distinct nulls in all the other columns. Therefore, a null value in column $A \in U - R_i$ of μ can be replaced during the chase process (either with another null or with a nonnull) only if $A \in R_i^+$ [8]. The problems illustrated in Example 4 are caused by the existence of two different ways that could potentially replace a specific null value. In order to avoid such troublesome situations, we require that for each relation scheme R_i, there is a unique way to derive R_i^+. We show that the representative instance of the database $r_1, \ldots, r_n$ satisfies $F = \cup_{i=1}^{n} F_i$ (for all possible databases $r_1, \ldots, r_n$) if and only if each relation scheme R_i^+ is uniquely derived. We define "uniqueness" after the following example.

Example 5. Consider the algorithm of Figure 1. This algorithm computes X^+ assuming that a cover of F is embodied in the relation schemes. Suppose the database scheme is $\langle ABC, \{A\}\rangle$, $\langle BD, \{B\}\rangle$, $\langle CD, \{C\}\rangle$. The closure of ABC can be computed (i.e., derived) in two different ways. Either we use BD and its key B to obtain $ABCD$ from ABC in line 3 of Figure 1, or we use CD and its key C to obtain $ABCD$ from ABC.

If the database scheme is $\langle ABC, \{A\}\rangle$, $\langle BD, \{B\}\rangle$, $\langle CE, \{C\}\rangle$, then there is only one way to compute $(ABC)^+$. BD is used to add D, and CE is used to add E.

A set Y is a *superkey* of R_j if Y contains X and A such that $X \in K_j$ (i.e., X is a key of R_j) and $A \in R_j - X$. We say that R_i satisfies the *uniqueness condition* if for all $j \neq i$, the closure $(R_i)^+_{F-F_j}$ does not contain any superkey of R_j. Note that the F_i's partition F, since there are no distinct relation schemes over the same set of attributes.

Example 6. For the first database scheme of Example 5, we have $F_1 = \{A \rightarrow BC\}$, $F_2 = \{B \rightarrow D\}$, and $F_3 = \{C \rightarrow D\}$. ABC does not satisfy the uniqueness condition, since $CD \subseteq (ABC)^+_{F-F_3}$ where C is a key of CD and D is another attribute of CD. In the second database scheme of Example 5, $F_1 = \{A \rightarrow BC\}$, $F_2 = \{B \rightarrow D\}$, and $F_3 = \{C \rightarrow E\}$. The relation scheme ABC satisfies the uniqueness condition. In proof, $(ABC)^+_{F-F_3} = ABCD$ and $ABCD$ does not contain any superkey of the relation scheme CE. Similarly, we have $(ABC)^+_{F-F_2} = ABCE$ and $ABCE$ does not contain any superkey of BD.

We now show that the uniqueness condition implies that R_i^+ is uniquely derived (in a sense to be defined soon).

PROPOSITION 1. *Suppose that R_i satisfies the uniqueness condition, and $R_j \subseteq R_i^+ (j \neq i)$. Let $X = (R_i)^+_{F-F_j} \cap R_j$. Then $X \in K_j$.*

PROOF. First, we observe the following fact.

Fact 1. During the computation of R_i^+ by the algorithm of Figure 1, we use the FDs of R_j only in the iteration that adds R_j to Y in line 3.

We now show that X must contain a key of R_j. Two cases are considered depending on whether R_j is added to Y (in line 3) during the computation of R_i^+.

Case 1. R_j is never added to Y in line 3. Thus, $R_i^+ = (R_i)^+_{F-F_j}$, and since $R_j \subseteq R_i^+$, a key of R_j is contained in X.

Case 2. R_j is added to Y in line 3. Let Y_0 be the value of Y at the beginning of the iteration that adds R_j to Y. By Fact 1, $Y_0 \subseteq (R_i)^+_{F-F_j}$, and Y_0 contains a key of R_j (since Y_0 is the value of Y when R_j is added to it in line 3). Therefore, X contains a key of R_j.

Since X contains a key of R_j, either $X \in K_j$ or X is a superkey of R_j. If X is a superkey, then the uniqueness condition is violated. Thus, $X \in K_j$. □

Let $X = (R_i)^+_{F-F_j} \cap R_j$, where $R_j \subseteq R_i^+$. By Proposition 1, $X \in K_j$. If $B \in X$, we say that R_j *uses* B in R_i^+ (sometimes we say that R_j *uses* X in R_i^+). If $B \in R_j - X$, we say that R_j *adds* B to R_i^+. We also say that R_i *adds* all its attributes to R_i^+ and *uses* none of them. The following propositions show that if R_i satisfies the uniqueness condition, then R_i^+ is uniquely derived in the sense that each $A \in R_i^+$ is added by a unique R_j.

PROPOSITION 2. *If R_i satisfies the uniqueness condition and $R_j \subseteq R_i^+$, then every $A \in R_j$ is either used by R_j in R_i^+ or added by R_j to R_i^+ (but not both).*

PROOF. Immediate from the definitions. □

PROPOSITION 3. *If R_i satisfies the uniqueness condition, then each $A \in R_i^+$ is added by a unique $R_j \subseteq R_i^+$.*

PROOF. Clearly, each $A \in R_i^+$ is added by at least one R_j. Suppose that for some $A \in R_i^+$ there are distinct R_q and R_p such that both R_q and R_p add A to R_i^+. Since R_q adds A, there is a key $X \in K_q$ such that $X \subseteq (R_i)^+_{F-F_q}$ and $A \in R_q - X$. Similarly, there is a key $Z \in K_p$ such that $Z \subseteq (R_i)^+_{F-F_p}$ add $A \in R_p - Z$. Consider a computation of R_i^+ by the algorithm of Figure 1.

Case 1. $q \neq i$ and R_q is never used in line 3 of the algorithm. Thus, $XA \subseteq R_q \subseteq (R_i)^+_{F-F_q}$ and so, R_i does not satisfy the uniqueness condition (a contradiction).

Case 2. $p \neq i$ and R_p is never used in line 3. This is similar to Case 1.

Case 3. R_q is used in line 3 after R_p (this includes the case where $p = i$, that is, R_p is not used at all in line 3). Let Y' be the value of Y just before R_q is used in line 3. Since $Y' \subseteq (R_i)^+_{F-F_q}$ and R_p has already been used (i.e., $R_p \subseteq Y'$), it follows that $R_p \subseteq (R_i)^+_{F-F_q}$. But $A \in R_p$ and so, $A \in (R_i)^+_{F-F_q}$. Since $X \subseteq (R_i)^+_{F-F_q}$, $X \in K_q$, and $A \in R_q - X$, the uniqueness condition is violated by R_i (a contradiction).

Case 4. R_p is used in line 3 after R_q. This is similar to Case 3.

Since at least one of q and p is different from i, no other case is possible. □

4. MAIN THEOREM

Let $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ be a database scheme, and suppose that the relations $r_1, \ldots, r_n$ of the database are not allowed to have any null values.[3] Consider the following two statements.

(I) For all databases $r_1, \ldots, r_n$ for $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ (i.e., each r_i satisfies F_i), the representative instance satisfies $F = \bigcup_{i=1}^n F_i$.

(II) For all i, the relation scheme R_i satisfies the uniqueness condition.

In this section we prove that I is true if and only if II is true. First, we prove two lemmas that are needed for the "if" part (which is more difficult to prove). For this we assume that each R_i satisfies the uniqueness condition, and we consider a database $r_1, \ldots, r_n$ for $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$. Let $r = \bigcup_{i=1}^n \alpha_U(r_i)$ and consider the computation of chase$_F(r)$. Suppose that μ is a tuple of r that has originated from r_i. We say that R_j *adds* A to μ if R_j adds A to R_i^+. Similarly, R_j *uses* A in μ if R_j uses A in R_i^+. By the uniqueness condition and Propositions 2 and 3, for each $A \in R_i^+$, there are unique R_j and key $X \in K_j$ such that Rj_j adds A to μ using X (clearly, $A \in R_j - X$).

LEMMA 1. *Suppose that all the R_i's satisfy the uniqueness condition. Let $R_p \subseteq R_i^+$, $Y \in K_p$, and $A \in R_p - Y$. If R_p either does not add A to R_i^+ or does not use Y in R_i^+, then there is $B \in Y$ such that R_p adds B to R_i^+.*

PROOF. Suppose that R_p does not add A to R_i^+. By Proposition 2, there is a key $V \in K_p$ used by R_p in R_i^+ such that $A \in V$. If $Y \subseteq V$, then $YA \subseteq V$ (since $A \in V$). Since $A \notin Y$, it follows that the key V properly contains the key Y of R_p. But this is impossible, and so $Y - V \neq \emptyset$ and every $B \in Y - V$ is added by R_p to R_i^+.

Now suppose that R_p does not use Y. If R_p does not use any key (i.e., $p = i$), then R_p adds every $B \in Y$. If R_p uses another key $V \in K_p$, then $Y \not\subseteq V$ (otherwise, one key properly contains another key). Therefore, there is $B \in Y - V$ that is added by R_p. □

We assume that chase$_F(r)$ is computed by repeatedly applying the FD rules for the embodied FDs. If an FD rule for $Y \rightarrow R_i - Y$ is applied to tuples μ_1 and μ_2, then we also say that μ_1 and μ_2 are equated *using* key Y of R_i or, simply, *using* R_i.

[3] In the Appendix, we prove the main result of this section also for databases with nulls.

LEMMA 2. *Suppose that all the R_i's satisfy the uniqueness condition. Let r' be an intermediate relation in the computation of $chase_F(r)$, and let $\mu_1, \mu_2 \in r'$.*

(*A*) *If $\mu_1(A) = \mu_2(A) = \delta_i$ for some column A, then*
  (A1) *the same R_j adds A to both μ_1 and μ_2, and the same key $X \in K_j$ is used by R_j in both μ_1 and μ_2, and*
  (A2) $\mu_1[R_j] = \mu_2[R_j]$.

(*B*) *If $\mu_1(A)$ is nonnull, then $\mu_1[R_j] \in r_j$, where R_j is the unique relation scheme that adds A to μ_1.*

PROOF. Induction on the number of applications of the FD rules that produce r' from r.

Basis. Zero applications. That is, $r' = r$, and so all null values in r' are distinct. Thus, part A is vacuously true. As for part B, let μ_1 be a tuple of r' that has originated from r_i. If μ_1(A) is nonnull, then $A \in R_i$ and R_i adds A to μ_1. But $\mu_1[R_i] \in r_i$, and so part B is true for r'.

Induction. Suppose that r'' is obtained from r by $n - 1 \geq 0$ applications, and r' is obtained from r'' by a single application. In particular, suppose that r' is obtained from r'' by equating tuples ν_1 and ν_2 using key Y of R_p (i.e., the FD rule for $Y \rightarrow R_p - Y$ is applied to ν_1 and ν_2 in r''). By the inductive hypothesis, the lemma is true for r'', and we have to show that it is true also for r'.

Part A. Let μ_1 and μ_2 be tuples of r' such that $\mu_1(A) = \mu_2(A) = \delta_i$ for some column A. If $\mu_1(A) = \mu_2(A)$ also in r'', then by the inductive hypothesis, conditions A1 and A2 of the lemma are satisfied in r'', and hence also in r' (if for some column B, we have $\mu_1(B) = \mu_2(B)$ in r'', then $\mu_1(B) = \mu_2(B)$ also in r'). Thus, we have to consider only μ_1, μ_2, and an A such that

(1) $\mu_1(A) = \nu_1(A)$ in r'',
(2) $\mu_2(A) = \nu_2(A)$ in r'', and
(3) $\nu_1(A) \neq \nu_2(A)$ in r'' and $A \in R_p - Y$.

(Otherwise, either $\mu_1(A) = \mu_2(A)$ in r'' or $\mu_1(A) \neq \mu_2(A)$ in r'.)

Case 1. R_p adds A using Y to both ν_1 and ν_2. Since $\nu_1(A) = \mu_1(A)$ in r'' (and $\mu_1(A)$ is null in r'', since it is null in r'), the inductive hypothesis implies that R_p adds A using Y also to μ_1, and

(i) $\mu_1[R_p] = \nu_1[R_p]$ in r''.

Similarly, R_p adds A using Y also to μ_2 and

(ii) $\mu_2[R_p] = \nu_2[R_p]$ in r''.

It remains to be shown that $\mu_1[R_p] = \mu_2[R_p]$ in r'. Suppose that $\nu_1(B)$ is nonnull in r'' for some $B \in R_p - Y$. Since R_p adds B to ν_1, part B of the inductive hypothesis implies that $\nu_1[R_p] \in r_p$ in r''. By (i), $\mu_1[R_p] \in r_p$ in r'', and so $\mu_1(A)$ is nonnull in r', contrary to assumption. Therefore, $\nu_1(B)$ is null in r'' for all $B \in R_p - Y$, and similarly for ν_2. Since ν_1 and ν_2 are equated using key Y of R_p, it follows that $\nu_1[R_p] = \nu_2[R_p]$ in r'. By (i) and (ii), $\mu_1[R_p] = \mu_2[R_p]$ in r'.

Case 2. R_p does not add A to ν_1. We show that that this case cannot actually occur. Let r_k be the relation from which ν_1 has originated. By [8, lemma 1.3], R_p

$\subseteq R_k^+$ (since ν_1 is equated with ν_2 using R_p). Lemma 1 implies that there is $B \in Y$ that is added by R_p to ν_1, because A is not added by R_p and $A \in R_p - Y$ (by (3)). Since ν_1 and ν_2 are equated using key Y of R_p, it follows that $\nu_1(B) = \nu_2(B)$ in r''. If $\nu_1(B)$ is null in r'', then by part A of the inductive hypothesis, $\nu_1(A) = \nu_2(A)$ in r'', contrary to condition (3). If $\nu_1(B)$ is nonnull in r'', then by part B of the inductive hypothesis, $\nu_1[R_p] \in r_p$ in r'', and, by condition (1), $\mu_1(A)$ is nonnull in r', contrary to assumption. Thus, $\nu_1(B)$ is neither null nor nonnull in r'', and, consequently, this case cannot occur.

Case 3. R_p does not add A to ν_2. This is similar to case 2.

Part B. We have to show that if $\mu_1(A)$ is nonnull in r', then $\mu_1[R_j] \in r_j$, where R_j adds A to μ_1. If $\mu_1(A)$ is also nonnull in r'', then by the inductive hypothesis, $\mu_1[R_j] \in r_j$ in r'', and so $\mu_1[R_j]$ is the same in r'' and r' (since all the columns of $\mu_1[R_j]$ in r'' are nonnull and cannot be changed). Thus, we have to consider only the following case:

(a) $\nu_1(A)$ is nonnull in r'',
(b) $\nu_2(A)$ is null in r'',
(c) $\mu_1(A) = \nu_2(A)$ in r'', and
(d) ν_1 and ν_2 are equated using key Y of R_p and $A \in R_p - Y$.

CLAIM 1. *$\nu_1[R_p] \in r_p$ in r'' (and, hence, also in r').*

PROOF

Case 1. R_p adds A to ν_1. Since $\nu_1(A)$ is nonnull in r'' (by (a)), the inductive hypothesis implies that $\nu_1[R_p] \in r_p$ in r''. Therefore, $\nu_1[R_p]$ is nonnull in r'', and hence, it is the same in r'' and r'.

Case 2. R_p does not add A to ν_1. By (d) and Lemma 1, there is a $B \in Y$ that is added by R_p to ν_1. Suppose that $\nu_1(B)$ is null. Since $\nu_1(B) = \nu_2(B)$ in r'' (by (d)), part A of the inductive hypothesis implies that $\nu_1(A) = \nu_2(A)$ in r''. This is contrary to (a) and (b), and therefore, $\nu_1(B)$ is nonnull in r''. By part B of the inductive hypothesis, $\nu_1[R_p] \in r_p$ in r''. □

CLAIM 2. *R_p uses Y in ν_2.*

PROOF. Suppose that R_p does not use Y in ν_2. By Lemma 1, there is a $B \in Y$ such that R_p adds B to ν_2. By (d), $\nu_2(B) = \nu_1(B)$ in r'', and by claim 1, $\nu_1[R_p] \in r_p$; hence, $\nu_2(B)$ is nonnull in r''. By part B of the inductive hypothesis, $\nu_2[R_p] \in r_p$ in r''. But this is impossible, since $\nu_2(A)$ is null in r''. □

By claim 2 and (d), R_p adds A to ν_2 using Y. Therefore, part A of the inductive hypothesis and (b) and (c) imply that R_p adds A to μ_1 using Y, and $\mu_1[R_p] = \nu_2[R_p]$ in r''. If $\nu_2(C)$ is nonnull in r'' for some $C \in R_p - Y$, then by claim 2 and part B of the inductive hypothesis, $\nu_2(A)$ is nonnull, contrary to (b). Hence, $\nu_2(C)$ is null in r'' for all $C \in R_p - Y$ and, by (d) and the FD rule for $Y \rightarrow R_p - Y$, we have $\nu_1[R_p] = \nu_2[R_p]$ in r'. Thus, $\mu_1[R_p] = \nu_2[R_p] = \nu_1[R_p] \in r_p$ in r'. □

COROLLARY 1. *Suppose that during the computation of chase$_F(r)$, the null in column A of μ_1 is replaced with a nonnull as a result of equating ν_1 and ν_2 using key Y of R_p. Then R_p adds A to μ_1 using Y.*

PROOF. Let ν_1, ν_2, and μ_1 be the same as in part B of the proof of Lemma 2. We have shown that R_p adds A to μ_1 using Y. □

THEOREM 1. *Let $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ be a database scheme, and suppose that the relations $r_1, \ldots, r_n$ of the database do not have any nulls. Then the following are equivalent:*

(I) For all databases $r_1, \ldots, r_n$ for $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$, the representative instance satisfies F.

(II) For all i, the relation scheme R_i satisfies the uniqueness condition.

PROOF. (II) implies (I). Suppose that some FD $X \rightarrow Y \in F_i$ is violated in the representative instance. That is, there are tuples μ_1 and μ_2 in the representative instance such that for some $A \in Y$

(1) $\mu_1[X] = \mu_2[X]$, and
(2) $\mu_1(A)$ and $\mu_2(A)$ are distinct nonnulls.

CLAIM 1. *$\mu_1[R_i] \in r_i$ and $\mu_2[R_i] \in r_i$.*

PROOF. We prove that $\mu_1[R_i] \in r_i$ (proving that $\mu_2[R_i] \in r_i$ is similar).

Case 1. R_i adds A to μ_1. By Lemma 2, part B, $\mu_1[R_i] \in r_i$.

Case 2. R_i does not add A to μ_1. By Lemma 1, there is $B \in X$ such that B is added by R_i to μ_1. Suppose that $\mu_1(B)$ is null. By Lemma 2, part A2, $\mu_1(A) = \mu_2(A)$, since $\mu_1(B) = \mu_2(B)$. This contradiction implies that $\mu_1(B)$ is nonnull and, by Lemma 2, $\mu_1[R_i] \in r_i$. □

By Claim 1 and condition (1) above, $\mu_1(A) = \mu_2(A)$, since r_i satisfies F_i. But this is contrary to condition (2), and so II implies I.

I implies II. Suppose that some R_i does not satisfy the uniqueness condition. That is, there is an $R_j (j \neq i)$, a key $X \in K_j$, and an attribute $A \in R_j - X$ such that $XA \subseteq (R_i)^+_{F-F_j}$. We have to show that there are relations $r_1, \ldots, r_n$ (without nulls) that satisfy $F_1, \ldots, F_n$, respectively, such that the representative instance does not satisfy F. We choose $r_1, \ldots, r_n$ as follows. For $k \neq j$, each r_k has exactly one tuple that maps all the attributes of R_k to 1. The relation r_j has exactly one tuple that maps A to 2 and each attribute in $R_j - A$ to 1. Let $r = \bigcup_{k=1}^{n} \alpha_U(r_k)$, and let μ_k be the tuple of r that has originated from r_k. We apply the chase process to r using only the FD rules for $F - F_j$ and tuples μ_k such that $k \neq j$. Let r' be the resulting relation. In r' we have $\mu_i(B) = 1$ for all $B \in (R_i)^+_{F-F_j}$. Therefore, tuples μ_i and μ_j of r' violate $X \rightarrow A$. Clearly, if we have a violation of $X \rightarrow A$ in r', then this violation exists also in the representative instance, and so the proof is complete. □

5. COMPUTING TOTAL PROJECTIONS OF THE REPRESENTATIVE INSTANCE

Suppose that $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ is a database scheme such that each R_i satisfies the uniqueness condition, and let X be a set of attributes. (We still assume that the relations of the database do not have null values.) In this section we show how to construct in polynomial time an expression E whose value is the X-total projection of the representative instance. The expression E is of the form

$\bigcup_j \pi_X(E_j)$, where each E_j is an extension join. Among all the expressions of this form whose value is the X-total projection of the representative instance, the expression E is minimal in both the number of extension joins and the total number of join operators.

Let $r = \bigcup_{k=1}^{n} \alpha_U(r_k)$, and $\mu \in \text{chase}_F(r)$ be a tuple that has originated from r_i. Consider the set

$$S = \{R_j \mid R_j \text{ adds } A \text{ to } \mu \text{ and } \mu(A) \text{ is nonnull}\}.$$

By Lemma 2, the tuple μ is nonnull exactly in the columns of $P = R_{j_1} \ldots R_{j_m}$, where $S = \{R_{j_1}, \ldots, R_{j_m}\}$. Let R_{j_p} and R_{j_q} be distinct members of S. By Corollary 1 and Lemma 2, the columns of μ for the attributes added by R_{j_p} become nonnull as a result of a single application of an FD rule for some FD of R_{j_p}, and similarly for R_{j_q}. Therefore, the columns of both R_{j_p} and R_{j_q} cannot become nonnull simultaneously (because R_{j_p} and R_{j_q} add distinct attributes). Consequently, let $R_{j_1}, R_{j_2}, \ldots, R_{j_m}$ be an ordering of the elements of S such that if $p < q$, then $\mu[R_{j_p}]$ becomes nonnull before $\mu[R_{j_q}]$. Note that $j_1 = i$.

LEMMA 3. *$\bowtie_{k=1}^{m} R_{j_k}$ is an extension join of R_i, and $\mu[P] \in \bowtie_{k=1}^{m} R_{j_k}$.*

PROOF. Consider μ just before $\mu[R_{j_p}]$ (for some $1 < p \leq m$) becomes nonnull. Let $\mu(B)$ be nonnull, and suppose that R_{j_q} adds B to R_i^+. By Lemma 2, $\mu[R_{j_q}]$ is nonnull, and, so, $q < p$. Therefore, the nonnull columns of μ are $R_j \cdots R_{j_{p-1}}$. By Corollary 1, $\mu[R_{j_p}]$ becomes nonnull as a result of applying some FD rule for an FD of R_{j_p}, and so $Y \subseteq R_{j_1} \ldots R_{j_{p-1}}$ for some key Y of R_{j_p}. Thus, $\bowtie_{k=1}^{m} R_{j_k}$ is an extension join (of R_i).

Now consider μ at the end of the chase process. By Lemma 2, $\mu[R_{j_k}] \in r_{j_k}$ for every $1 \leq k \leq m$, and μ is nonnull exactly in the columns of $P = R_{j_1} \cdots R_{j_m}$. Therefore, $\mu[P] \in \bowtie_{k=1}^{m} R_{j_k}$. □

Lemma 3 states that the nonnull portion of any tuple in the representative instance is in some extension join. Conversely, if $\bowtie_{j=1}^{k} R_{i_j}$ is any extension join and $\mu \in \bowtie_{j=1}^{k} R_{i_j}$, then there is a tuple ν in the representative instance such that $\nu[R_{i_1} \ldots R_{i_k}] = \mu$. Therefore, we have the following corollary.

COROLLARY 2. *If all the R_i's satisfy the uniqueness condition, then the X-total projection of the representative instance is given by the expression $\bigcup_j \pi_X(E_j)$, where each E_j is an extension join over a set of attributes containing X.*

In the sequel, an extension join over a set of attributes containing X is called an extension join *spanning* X. The expression E of Corollary 2 might have redundant subexpressions. We now show how to minimize it. Clearly, there is an extension join spanning X of R_{j_1} only if $X \subseteq R_{j_1}^+$. Furthermore, if E_1 and E_2 are two extension joins spanning X such that every operand of E_1 is also an operand of E_2, then $\pi_X(E_2) \subseteq \pi_X(E_1)$. Therefore, we need to consider only *minimal* extension joins spanning X of R_{j_1}, that is, extension joins $\bowtie_{k=1}^{m} R_{j_k}$ spanning X such that the following is true. There is no extension join spanning X of R_{j_1} whose set of operands is a proper subset of $\{R_{j_1}, \ldots, R_{j_m}\}$.

LEMMA 4. *If $X \subseteq R_i^+$, and $R_1, \ldots, R_n$ satisfy the uniqueness condition, then the minimal extension join spanning X of R_i is unique and can be found in linear time.*

```
    begin
(1) $S := \emptyset$;
(2) $Y := X$;
(3) while there is an $R_j$ such that $R_j \notin S$ and $R_j$ adds some $A \in Y$ to $R_i^+$ do
      begin
(4)     $S := S \cup \{R_j\}$;
(5)     $Y := YR_j$
      end
    end
```

Figure 2

PROOF. The algorithm of Figure 2 computes a set S of relation schemes that must be included in any extension join spanning X of R_i. The idea is that S must contain any R_j that adds an attribute $A \in X$ to R_i^+ and, recursively, if $R_k \in S$, then S includes also every R_j that adds an attribute of R_k. Now consider an ordering $R_{j_1}, \ldots, R_{j_p}$ of the elements of S such that if $k < m$, then R_{j_k} is used before R_{j_m} in line 3 of Figure 1 during the computation of R_i^+. (Note that $j_1 = i$.)

CLAIM 1. $\bowtie_{k=1}^{p} R_{j_k}$ *is an extension join* (*of* R_i).

PROOF. Let $V = R_{j_1} \ldots R_{j_m}$, and let Y be the key used by $R_{j_{m+1}}$ in R_i^+ ($1 \le m < p$). Consider an attribute $B \in Y$, and suppose that R_q adds B to R_i^+. R_q must have been added to S in line 4 of Figure 2, since $R_{j_{m+1}} \in S$. Further, R_q must be used before $R_{j_{m+1}}$ during the computation of R_i^+. Therefore, if $B \in Y$, then $B \in V$ and, so, $\bowtie_{k=1}^{p} R_{j_k}$ is an extension join. □

The algorithm of Figure 2 can be implemented in linear time using a data structure similar to that used in [5]. Similarly, the ordering of the elements of S can be determined in linear time.

We have shown that we need to consider only minimal extension joins spanning X of those R_j such that $X \subseteq R_j^+$. Now suppose that E_i and E_j are minimal extension joins spanning X of R_i and R_j, respectively. The following lemma implies that if R_i is an operand of E_j then $\pi_X(E_j) \subseteq \pi_X(E_i)$.

LEMMA 5. *If R_i is an operand of E_j, then every operand of E_i is also an operand of E_j.*

PROOF. Apply the algorithm of Figure 2 to X and R_i, and let $R_{i_1}, \ldots, R_{i_p}$ be the order in which the elements of S are added in line 4. We prove by induction on k that $XR_{i_1} \cdots R_{i_k} \subseteq V$ and R_{i_k} is an operand of E_j, where V is the set of attributes in E_j.

Basis. $k = 0$. Obvious.

Induction. Suppose that R_{i_k} is not an operand of E_j (i.e., $i_k \neq i$), and denote i_k by q. Since R_q is added after $R_{i_1}, \ldots, R_{i_{k-1}}$ in line 4, R_q adds some $B \in XR_{i_1} \ldots R_{i_{k-1}}$ to R_i^+ using a key Y, that is

$$Y \subseteq (R_i)^+_{F-F_q} \quad \text{and} \quad B \in R_q - Y. \tag{1}$$

By the inductive hypotheses, $B \in V$. Since E_j is an extension join of R_j that does not include R_q as an operand, it follows that

$$V \subseteq (R_j)^+_{F-F_q} \tag{2}$$

Figure 3

```
    begin
(1) $W := \emptyset$;
(2) Let $T = \{R_i \mid X \subseteq R_i^+\}$;
(3) for every $R_i \in T$ do
      begin
(4)     let $E_i$ be the minimal extension join spanning $X$ of $R_i$;
(5)     if there is no $R_j \in T(j \neq i)$ that is an operand of $E_i$ then
(6)       add $E_i$ to $W$
(7)     else remove $R_i$ from $T$
      end;
(8) let $W = \{E_1, \ldots, E_m\}$;
(9) return the expression $\cup_{j=1}^{m} \pi_X(E_j)$
    end
```

and so $B \in (R_j)^+_{F-F_q}$. In order to derive a contradiction, it remains to be shown that $Y \subseteq (R_j)^+_{F-F_q}$, since it implies that R_j violates the uniqueness condition. Since R_i is an operand of E_j, we have $R_i \subseteq V$, and it follows from eq. (2) that $R_i \subseteq (R_j)^+_{F-F_q}$. Therefore, we also have

$$(R_i)^+_{F-F_q} \subseteq (R_j)^+_{F-F_q} \tag{3}$$

and by eq. (1), $Y \subseteq (R_j)^+_{F-F_q}$. □

THEOREM 2. *Let $\langle R_1, K_1 \rangle, \ldots, \langle R_n, K_n \rangle$ be a database scheme such that each R_i satisfies the uniqueness condition, and suppose that the relation $r_1, \ldots, r_n$ of the database do not have any nulls. An expression whose value is the X-total projection of the representative instance can be found in $O(n^2)$ time, where n is the space needed to write down the database scheme.*

PROOF. The algorithm for constructing the expression is given in Figure 3. For a given value of T, we define $E(T) = \{E_k \mid E_k$ is the minimal extension join spanning X of R_k and $R_k \in T\}$. Let $F(T)$ be the union of all extension joins in $E(T)$ after projecting them onto X. Using Lemma 5, we can easily prove by induction that for all values of T obtained in line 3, $F(T) \equiv F(T_0)$, where T_0 is the initial value of T. By Corollary 2 and Lemma 4, the value of $F(T_0)$ is equal to the X-total projection of the representative instance. The expression returned by the algorithm of Figure 3 is $F(T_f)$, where T_f is the final value of T. Thus the algorithm returns an expression whose value is the X-total projection of the representative instance.

By Lemma 4 and [5], the algorithm of Figure 3 can be implemented to run in $O(n^2)$ time. □

An *SPJ expression* is any relational expression consisting only of the operators select, project, and join. A *union of SPJ expressions* is any relational expression of the form $\cup_{j=1} P_j$, where each P_j is an SPJ expression.

COROLLARY 3. *Among all unions of SPJ expressions whose value is the X-total projection of the representative instance, the expression returned by the algorithm of Figure 3 is minimal in both the number of union operators and the total number of join operators.*

PROOF. The expression returned in line 9 is nonredundant in the sense that if any subexpression of the form $\pi_X(E_j)$ is removed from the union, then we can find relations $r_1, \ldots, r_n$ for which the value of the resulting expression is not the X-total projection of the representative instance. In proof, consider the following $r_1, \ldots, r_n$. For each operand R_i of E_j, the relation r_i has exactly one tuple with 1 in every column; all the other relations are empty. Clearly, the value of E_j is a relation with exactly one tuple that has 1 in every column. All the other extension joins have at least one empty relation, and so their value is the empty relation. Thus, removing $\pi_X(E_j)$ changes the value of the expression from a relation with one tuple to the empty relation. By [23, theorem 5], if $\bigcup_{j=1}^{m} \pi_X(E_j)$ is nonredundant, then it is minimal in the number of union operators. The same theorem of [23] also implies that $\bigcup_{j=1}^{m} \pi_X(E_j)$ is minimal in the total number of join operators, since each $\pi_X(E_j)$ is a minimal SPJ expression (because each relation scheme has at most one occurrence in E_j, and the relation r_i for each R_i can be chosen independently of the other relations). □

Example 7. Suppose that the attributes are P(project), D(department), M(manager), L(location), and A(assistant), and the database scheme is $\langle LDP, \{LD\}\rangle$, $\langle DPM, \{DP\}\rangle$, $\langle LMA, \{LM\}\rangle$.

Intuitively, the database scheme describes an appliction in which each project belongs to several departments and is carried out in several locations, but a department can have only one project in each location. In each department participating in a project, there is a manager responsible for that project. Each manager has an assistant in each location. These relation schemes satisfy the uniqueness condition.

Suppose we want to compute the total projection of the representative instance onto LM. After line 2 of Figure 3 is executed, $T = \{LDP, LMA\}$. The minimal extension join spanning LM of LDP is $LDP \bowtie DPM$, and this extension join is added to W in line 6. The minimal extension join spanning LM of LMA is LMA, and it is also added to W in line 6. Thus, the expression for the LM total projection of the representative instance is $\pi_{LM}(LDP \bowtie DPM) \cup \pi_{LM}(LMA)$.

The result of the above expression is all tuples (l, m) such that either manager m has an assistant in location l or manager m manages some project in location l. Considering the fact that there might be partial information (e.g., a manager with a project in a location where he does not have an assistant, or a manager with an assistant in a location where he does not have a project), the correct answer is indeed given by the above expression.

6. CONCLUSIONS

We have proposed the representative instance as a measure of the data stored in the database, and we have characterized, in terms of the uniqueness condition, the database schemes for which the representative instance always satisfies the functional dependencies. The uniqueness condition can be viewed as an extension of Boyce-Codd normal form, since it removes interrelation anomalies in the sense that each relation can be updated independently of the contents of the other relations without violating global consistency. (It is an extension of Boyce-Codd normal form, since a relation scheme that satisfies the uniqueness condition is also in Boyce-Codd normal form.) This condition is much less restrictive than the one given in [7], and we believe that many practical applications satisfy it. We have also shown how to efficiently compute total projections of the representative instance if the uniqueness condition is satisfied.

In [22] we have dealt with database schemes that do not satisfy the uniqueness condition. We have shown that the representative instance can be guaranteed to satisfy the functional dependencies if the modified foreign-key constraint is imposed on the database. The modified foreign-key constraint is less restrictive and semantically more meaningful than imposing the existence of a universal instance. Under the modified foreign-key constraint, total projections of the representative instance can be computed by performing the union of several extension joins.

APPENDIX. DATABASES WITH NULLS

We now consider databases that may have (marked) nulls. Each null δ_k may appear in several relations, in each relation δ_k may appear in several tuples, and in each tuple δ_k may appear in several columns. When null values are allowed in the relations $r_1, \ldots, r_n$ of the database, we should compute $\text{chase}_{F_i}(r_i)$ (for all i) after every update in order to check that each r_i satisfies F_i. However, if δ_k appears in more than one relation and during the chase process it has to be replaced with another symbol s, then all occurrences of δ_k (in all the r_i's) must be replaced with s. Therefore, $\text{chase}_{F_1}(r_1), \ldots, \text{chase}_{F_n}(r_n)$ must be computed simultaneously in the following way. We start with $r_1, \ldots, r_n$, and as long as there is an FD rule for some F_i that can be applied to r_i, we apply that FD rule as follows. If the FD rule implies that δ_k should be replaced by the symbol s, then all occurrences of δ_k in all the r_i's are replaced with s. The chase process terminates when for all i, no FD rule for F_i can be applied to r_i. We still denote the relation r_i at the end of this process by $\text{chase}_{F_i}(r_i)$. Our definition for the satisfaction of functional dependencies still holds, that is, r_i satisfies F_i if $\text{chase}_{F_i}(r_i)$ satisfies F_i. Essentially, this definition means that the following two statements are equivalent.

(1) For all i, the relation r_i satisfies F_i.
(2) There is a way to replace in $r_1, \ldots, r_n$ all occurrences of each δ_k by some constant c and obtain relations $r'_1, \ldots, r'_n$ (without nulls) such that for all i, the relation r'_i satisfies F_i.

THEOREM 3. *Let $\langle R_1, K_1\rangle, \ldots, \langle R_n, K_n\rangle$ be a database scheme. Then the following are equivalent:*

(I) For all databases $r_1, \ldots, r_n$ for $\langle R_1, K_1\rangle, \ldots, \langle R_n, K_n\rangle$ (i.e., each r_i satisfies F_i), the representative instance satisfies F.
(II) For all i, the relation scheme R_i satisfies the uniqueness condition.

PROOF. The proof of I implies II is the same as that of Theorem 1. For the proof of II implies I, we observe the following. If we replace each r_i by $\text{chase}_{F_i}(r_i)$, and then replace all occurrences of each null value by a new distinct constant (that appears nowhere else) then, by Theorem 1, the new representative instance does not violate any FD. In the new representative instance we can replace back each new constant by its corresponding null value, and the result is the representative instance of the original database (which, of course, does not violate any FD). □

When null values are allowed in the relations of the database, it is not clear anymore that an X-total projection of the representative instance should be computed in response to a query. The nulls that are stored in the database by the user seem to convey information that might be of interest to the user (especially if the same null appears in many places). Therefore, we modify our definition of the X-total projection to include all tuples in the projection onto X consisting of either constants or nulls that appear in the relations of the database. With the new definition in mind, we can now use the results of Section 5 to compute an expression for the X-total projection of the representative instance.

REFERENCES

1. AHO, A.V., BEERI, C., AND ULLMAN, J.D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 297-314.
2. AHO, A.V., SAGIV, Y., AND ULLMAN, J. D. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 435-454.
3. AHO, A.V., SAGIV, Y., AND ULLMAN, J.D. Equivalences among relational expressions. *SIAM. J. Comput. 8*, 2 (May 1979), 218-246.
4. ARMSTRONG, W.W. Dependency structures of database relationships. In *Proc. IFIP 74*, North Holland, Amsterdam, 1974, pp. 580-583.
5. BERRI, C., AND BERNSTEIN, P.A. Computational problems related to the design of normal form relational schemas. *ACM Trans. Database Syst. 4*, 1 (March 1979), 30-59.
6. BERNSTEIN, P.A. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst. 1*, 4 (Dec. 1976), 277-298.
7. BERNSTEIN, P.A., AND GOODMAN, N. What does Boyce-Codd normal form do? In *Proc. Int. Conf. on Very Large Data Bases*, Montreal, Canada, 1980, pp. 245-259.
8. BISKUP, J., DAYAL, U., AND BERNSTEIN, P.A. Synthesizing independent database schemas. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Boston, 1979, 143-151.
9. CODD, E.F., A relational model for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377-387.
10. CODD, E.F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 397-434.
11. FAGIN, R., MENDELZON, A.O., AND ULLMAN, J.D. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst. 7*, 3 (Sept. 1982), 343-360.
12. HONEYMAN, P. Extension joins. In *Proc. Int. Conf. on Very Large Data Bases*, Montreal, Canada, 1980, pp. 239-244.
13. HONEYMAN, P. Testing satisfaction of functional dependencies. *J. ACM 29*, 3 (July 1982), 668-677.
14. HONEYMAN, P., LADNER, R.E., AND YANNAKAKAIS, M. Testing the universal instance assumption. *Inf. Proc. Lett. 10*, 1 (Feb. 1980), 14-19.
15. KORTH, H.F. A proposal for the SYSTEM/U query language. Unpublished memorandum, Stanford Univ., Stanford, Calif., 1980.
16. KORTH, H.F., AND ULLMAN, J.D. System /U: A database system based on the universal relation assumption. In *Proc. XP1 Conf.*, Stony Brook, N.Y., June 1980.
17. LIEN, Y.E. Multivalued dependencies with null values in relational data bases. In *Proc. 5th Int. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, 1979, pp. 61-66.
18. MAIER, D. Discarding the universal instance assumption: Preliminary results. In *Proc. XP1 Conf.*, Stony Brook, N.Y., June 1980.
19. MAIER, D., MENDELZON, A.O., AND SAGIV, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455-469.
20. MAIER, D., AND ULLMAN, J.D. Maximal objects and the semantics of universal relation data-

bases. Tech. Rep. 80-016, Dept. Computer Science, State Univ. New York at Stony Brook, Stony Brook, N.Y., Nov. 1980.
21. OSBORN, S.L. Towards a universal relation interface. In *Proc 5th Int. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, 1979, pp. 52-60.
22. SAGIV, Y. Can we use the universal instance assumption without using nulls? In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Ann Arbor, Mich., April 1981, pp. 108-120.
23. SAGIV, Y., AND YANNAKAKIS, M. Equivalences among relational expressions with the union and difference operator. *J. ACM 27*, 4 (Oct. 1980), 633-655.
24. SCIORE, E. The universal instance and database design. Tech. Rep. TR 271, Dept. Elec. Eng. and Comp. Sci., Princeton Univ., Princeton, N.J., June 1980.
25. VASSILIOU, Y. A formal treatment of imperfect information in database management. Tech. Rep. CSRG-123, Univ. Toronto, Nov. 1980.
26. VASSILIOU, Y. Functional dependencies and incomplete information. In *Proc. Int. Conf. on Very Large Data Bases*, Montreal, Canada, 1980, pp. 260-269.

Received August 1981; revised August 1982; accepted August 1982