# FUNCTIONAL DEPENDENCIES AND INCOMPLETE INFORMATION

Yannis Vassiliou

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #16
GBA #81-08 (CR)

Published in VLDB '80 Conference Proceedings, Montreal, October 1980.

# Functional Dependencies and Incomplete Information

*Yannis Vassiliou* †

Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

**Abstract**

Functional dependencies play an important role in relational database design. They are defined in the context of a single relation which at all times must contain tuples with non-null entries. In this paper we examine an extension of the functional dependency interpretation to handle *null* values, that is, entries in tuples that represent incomplete information in a relational database. A complete axiomatization of inference rules for extended functional dependencies is also presented. Only after having such results is it possible to talk about decompositions and normalization theory in a context of incomplete information. Finally, we show that there are several practical advantages in using nulls and a weaker notion of constraint satisfiability.

## 1. Introduction

Work on relational database design started soon after the publication of the pilot papers on the relational model [Codd 70], [Codd 72]. Normalization, which is a relational schema design process, centers around the notion of data dependencies, a purely syntactic notion, that has been introduced to capture semantics in a relational database. These dependencies are used as guidelines for the design of a relational schema which is conceptually meaningful and is free of certain update anomalies [Date 77]. The theory of dependencies, in particular of the functional ones, has been studied in depth [Beeri et al 78].

Data dependencies are defined between attributes that belong to a single relation. As a starting point then, we need to assume that the portion of the real world that we wish to represent via the relational model can be modelled with one relation which contains all the attributes. This relation is called the *universal relation*. Sound and complete rules from which other dependencies are logically inferred may be considered only under this condition. Furthermore, at the instance level, any multi-relation database produced by a normalization process can be thought of as a collection of projections of a universal relation. The above is called the *Universal Relation Assumption* and is present, implicitly or explicitly, in any work on relational database design which uses the theory of dependencies. We note that the assumption has two aspects. First, we have the requirement of the existence of a universal relation which can model the real world. The second aspect is the requirement that any instance of a multi-relation database which also models the real world corresponds in a precise and restrictive way to an instance of the universal relation.

The validity of this fundamental assumption has been questioned both on practical and recently [Bernstein and Goodman 80] on theoretical grounds. The theoretical attacks concentrate on the second aspect of the assumption where it is shown that the benefits of normalized schemas (e.g. no update anomalies) are lost when the assumption is made. From a practical point, it is not realistic to assume that a universal relation instance will have all rows filled with values, either in its initial instance or after being recovered (via joins) from decompositions. It is therefore necessary to fill the "gaps" which are created in the universal relation instance with some special values, called *null* values. These values are forms of incomplete information and ways to treat them range from rudimentary ones (e.g. zeros, values like any other) to more sophisticated ones [Codd 79], [Lipski 79]. An overview of our work on the subject [Vassiliou 79] appears in section 2. It is noted that all the above reported research considers only the retrieval aspects of the problem. That is, how queries behave when *null* values are present in the database. The general approach is to extend existing functions and operations in order to have them defined on *nulls*. Often, the classical two-valued logic is abandoned for more appropriate n-valued logics and/or modal logics.

The semantic rules of a database (in our case the data dependencies) are defined in a context of no *nulls*. In order to allow for *nulls* we must carefully redefine dependencies (more precisely, their interpretations) together with their requirements of satisfiability and inference rules. The satisfiability requirements give us a pattern of allowable *nulls* in a (universal) relation.

In this paper, we extend the notion of a functional dependency (FD) interpetation to apply to *nulls*. We then present satisfiability requirements and give necessary and sufficient conditions under which these requirements are met. Finally, the properties of FDs are examined and inference rules, which are shown to be sound and complete are presented. We claim that only after having such results is it conceivable to safely talk about decompositions and normalization theory when *nulls* are allowed in relation instances.

The next section briefly overviews the framework of our work. Functional dependencies are discussed in section 3. In the fourth section, the interpretations of the dependencies are redefined to apply to *nulls*, and conditions are presented for the FDs to meet satisfiability requirements. Section 5 examines properties of FDs, specifically inference rules. In this section

† Author's current address: Graduate School of Business Administration, Computer Applications and Information Systems, New York University.

we depart from our system to work with an equivalent but well-axiomatized propositional logic system. The technique is the same as the one used in [Fagin 77]. Ways of efficiently testing for FD-satisfiability are presented in section 6. In addition, the notion of a "minimally incomplete" relation instance is introduced, and the rules for reaching such an instance are shown to constitute a finite Church-Rosser system. In our concluding remarks we discuss the importance of our results and outline our on-going research aimed at extending these results to include modification operations. The connection with the universal relation assumption is also discussed.

## 2. A Treatment of Null Values

A formal treatment for two of the manifestations of a null value was presented in [Vassiliou 79]. One can use null to represent a value which exists, but is presently unknown to the database. This type of null is called *missing*. Null may also be interpreted as a value which is inconsistent - it is known to the database as more than one value. We note that, even though the latter null may in general appear in a database, it has no place in a database where certain semantic rules are required to be valid. Therefore, we do not discuss the inconsistent null in the present context, concentrating on the missing null which we call simply *null*.

The inclusion of *null* in domains of database values presents complications when we consider the behavior of functions or operations on these domains. We use a uniform rule to extend all the functions which has a simple intuitive explanation. Since a *null* is an existing value, although unknown at present, it must represent one of the regular values in the domain. Any function, which is evaluated on the *null*, will take a particular value in its range only if for every non-null in the domain, the function evaluates to the same value. That is, for all values in the domain in turn, we substitute the *null* with one value and evaluate the function as usual. If all evaluations have the same result, it means that our incomplete knowledge is not essential for this function - it does not matter what the value represented as *null* is. Otherwise, we are not able to give a precise evaluation for the function.

The above intuitive idea is supported theoretically by Scott's mathematical theory of computation [Stoy 77]. It is shown in [Vassiliou 79] that the introduction of the *null* in a database domain makes the domain a *lattice* with an approximation ordering. *Null* carries less information than all other domain values. Thus, it approximates all the other values. The extensions of all functions defined on these domains must be continuous according to the theory. One of these continuous extensions, called the *least extension*, is the theoretical formulation of our intuitive rule. All values that the function takes, when evaluated on domain values, are collected and, according to the least extension rule, the best possible approximation to the correct value (least upper bound in the lattice of values) is returned.

As a simple illustration, consider queries (functions from relation tuples to truth-values) on the relation *R(***name**, *marital-status)*. Assume that the domain of *marital-status* has only two values "married" and "single", and that there is a tuple ("John", *null)* in an instance of $R$. If the query $Q$ is "Is John married?", we evaluate the query for each of the *marital-status* values and we return the least upper bound of the two answers. In this case: $Q$("John", *null*)=*lub*{*yes*,*no*}=*unknown*. On the other hand, for the query Q' "Is John either married or single?", according to our rule: Q'("John",*null*)=*lub*{*yes*,*yes*}=*yes*.

We note that the use of such an evaluation rule has an unacceptable complexity for practical considerations. In [Vassiliou 79], algorithms were presented for syntactic query transformations so that no substitutions are necessary for the evaluation.

## 3. Functional Dependencies and their Interpretations

Let $R$ be a relation scheme and $X,Y$ be sets of attributes in $R$ (not necessarily distinct). A functional dependency (FD) denoted by $f: X \rightarrow Y$, or simply $f$, is a statement about $R$. For example, consider the relation scheme in figure 1.1 and the statement: "Employees have only one salary and work in only one department." The expression of this semantic rule is the functional dependency: $E\# \rightarrow SL,D\#$. Accordingly, the interpretation of an FD is a *predicate* on instances of $R$ defined as†:

$$f(r) = \begin{cases} true & \text{if for every } t,t' \text{ in } r, \text{ either } t[X] \neq t'[X], \\ & \text{or, if } t[X]=t'[X], \text{ then } t[Y]=t'[Y] \\ false & \text{in any other case} \end{cases}$$

We say that $f$ *holds* (or *is satisfied*, or *is true*) in a relation instance $r$ if $f(r)$ is equal to *true*. Furthermore, $f$ is *valid* for $R$ if it holds in all its instances. It is trivial to verify that the functional dependencies $E\# \rightarrow SL,D\#$ and $D\# \rightarrow CT$ hold in the instance $r$ of figure 1.2.

For convenience, we now modify our notation to have $f$ defined as a function with two arguments, a tuple and a relation instance. Hence,

$$f(t,r) = \begin{cases} true & \text{if for every } t' \text{ in } r, \text{ either } t[X] \neq t'[X], \\ & \text{or, if } t[X]=t'[X], \text{ then } t[Y]=t'[Y] \\ false & \text{in any other case} \end{cases}$$

and say that $f$ *holds* in $r$ if for every $t$ in $r$ $f(t,r) = true$.

When a set of FDs hold in a relation instance $r$, there are usually some other dependencies that also hold in $r$. More formally, a functional dependency $f$ is *implied* by a set of FDs $F = \{f_1, f_2, \ldots, f_k\}$ if there is no counterexample relation $r'$, such that $F$ holds in $r'$ but $f$ does not hold in $r'$. A very important result which constitutes the basis for much research on FDs is Armstrong's inference rules which are sound and complete for functional dependencies.

We now make two observations commonly used for proofs about FDs (sometimes implicitly). Let $2T = \{ s \mid s \subseteq r, |s|=2 \}$ (i.e. $2T$ is the set of all the two-tuple subrelations of a relation instance $r$ ).

† The symbol $r$ denotes a relation instance or simply relation, $t$, $t'$ denote tuples in $r$, and $t[X]$ is the projection of $t$ on the values of the attributes in $X$.

Figure 1.1 An example relation scheme and functional dependencies.

$$R(E\#,\ SL,\ D\#,\ CT)$$

Attribute explanation

- $E\#$ - employee serial number
- $SL$ - salary
- $D\#$ - department number of the employee
- $CT$ - contract type.

Functional dependencies defined, $f_1 : E\# \rightarrow SL, D\#$, and $f_2 : D\# \rightarrow CT$

Figure 1.2 An instance of the relation scheme $R$.

| $r$ | | | |
|---|---|---|---|
| $E\#$ | $SL$ | $D\#$ | $CT$ |
| 1 | 10k | a | x |
| 2 | 30k | b | y |
| 3 | 20k | a | x |
| 4 | 15k | c | y |

Figure 1.3 An instance of $R$ with *nulls*.

| $r$ | | | |
|---|---|---|---|
| $E\#$ | $SL$ | $D\#$ | $CT$ |
| 1 | - | a | x |
| 2 | 30k | - | y |
| 3 | 20k | a | x |
| 4 | 15k | c | y |

Figure 2 Examples of FDs with *nulls*

Consider the relation scheme $R(A, B, C)$, the FD $f : AB \rightarrow C$, and the following four instances of $R$ ( a "–" means *null*).

$r_1$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | – |
| $a_2$ | $b_2$ | $c_2$ |
| $a_3$ | – | $c_3$ |

$r_2$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | – | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_3$ | $b_3$ | $c_3$ |

$r_3$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | – | $c_1$ |
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | – | – |

$r_4$

| $A$ | $B$ | $C$ |
|---|---|---|
| – | $b_1$ | $c_2$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_2$ | $b_1$ | $c_3$ |

We use Proposition 1 ($t_1$ represents the first tuple in each instance).

$f(t_1, r_1) = true$ because of [T2]

$f(t_1, r_2) = true$ because of [T3]

$f(t_1, r_3) = true$ because of [T3]

Assume that for the instance $r_4$ the domain of $A$ has only two values: $a_1$, $a_2$.

$f(t_1, r_4) = false$ because of [F2]

[1] The functional dependency $f$ holds in $r$ iff $f$ holds in every (two-tuple) relation in $2^r$.

[2] The functional dependency $f$ is implied by a set of functional dependencies $F$ iff $f$ is implied by $F$ in the world of two-tuple relations.

These observations allow us to consider only two-tuple relations in proofs about functional dependencies without loss of generality. We will see, however, that these observations are not always correct when *null* values are allowed.

## 4. Functional Dependencies in Relations with Null Values

From this point on, we assume that *nulls* are allowed in relation instances. An example of such an instance is figure 1.3. To extend the notion of a functional dependency (more precisely, its *interpretation* as a function) we use the *least extension rule*. Some definitions are needed first. Let $t$ be a tuple which may have null values. We define a *completion* of $t$ as a tuple $t'$ in which we substitute for all *null* values and $t'$ agrees with $t$ in all the values except where a value in $t$ is *null*. The set of all completions $AP$ of a tuple $t$ on a set of attributes $R$ is well-defined. †

$$AP(t,R) = \{t' \mid t' \text{ is a } completion \text{ of } t\}$$

Similarly, we define $AP(r,R)$, the set of all completions of $r$ projected on $R$.

$$AP(r,R) = \{r' \mid t' \in r' \text{ if } t' \text{ is a } completion \text{ of } t,\ t \in r\}$$

We are now ready to present the extension of the interpretation of a functional dependency $f : X \rightarrow Y$. Let $r$ be a relation and $t$ a tuple.

$$f^{\circ}(t,r) = \begin{cases} f(t,r) & \text{if all values in } t[XY],\ r[XY] \text{ are not } null \\ lub\{f(t',r') \mid t' \in AP(t,XY),\ r' \in AP(r,XY)\} & \text{otherwise} \end{cases}$$

We use $f^{\circ}$ to denote the extension of $f$, but from now on we will drop the "°" for simplicity. The above definition is refined on a case-by-case basis (considering the *null* as one of the $t[X]$ values or as one of the $t[Y]$ values) to establish necessary and sufficient conditions for an FD to take a particular truth value. Before we present these conditions formally, we give an informal explanation. Recall that the value of $f(t,r)$, with a *null* appearing in $t[XY]$, is *false (true)* only when it evaluates to *false (true)* for all substitutions of the *null* value. For the discussion below, the concept of an attribute domain and its size is important. Domains are finite and are assumed known. Further restrictions on domains will be presented and justified.

Assume first that the *null* appears in $t[Y]$ and that $Y$ has only one attribute. Trivially, $f(t,r) = true$ whenever $t[X]$ appears uniquely in $r$. When $t[X]$ is not unique in $r$, say $t[X] = t'[X]$ for some tuple $t'$ in $r$, we may not claim that $f(t,r) = true$. It is possible to substitute the *null* in $t[Y]$ so that $f(t,r)$ is *false* with the

† The name $AP$ is not arbitrary. It comes from the fact that the completions of $t$ are the non-null tuples that $t$ *approximates* in the lattice of tuples.

substituted value (e.g. any value that makes $t[Y] \neq t'[Y]$). However, we also may not claim that $f(t,r)=false$ since we can make it *true* if we substitute $t[Y]$ with $t'[Y]$. Hence, depending on how we substitute for the *null* the FD is either *true* or *false*. Since we do not know what the actual value of the *null* is we take the $lub\{true, false\}=unknown$ as the value of the FD on $t$ and $r$.

Assume now that the *null* appears in $t[X]$ ($X$ can consist of more than one attribute). The dependency will evaluate to *true* in two cases. First, there is no tuple in $r$ whose projection on $X$ is a completion of $t[X]$ (i.e. anytime we substitute for the *null* we end-up with a unique $t[X]$ among the tuples of $r$). In this case, $f(t,r)=true$ trivially. Second, for each tuple $t'$ in $r$ such that $t'[X]$ is a completion of $t[X]$, we also have $t'[Y]=t[Y]$. There is exactly one case where the value of the FD on $t$ and $r$ is *false* because of a *null* in $t[X]$. This case arises when we run out of domain values, while attempting substitutions of this *null*, while at the same time trying to keep the dependency *not false*, i.e. a substitution for which the dependency is *true*. For this to happen, it must be the case, first, that all completions of $t$ appear in $r$. (Otherwise, we may substitute for the *null* and create a completion that does not appear in $r$, thus ensuring that $f(t,r) \neq false$.) In addition, since for all such completions we must ensure that the dependency is *false*, it is required that the $t[Y]$ value is unique among all the $t'[Y]$ values where $t'$ is a completion of $t$ that appears in $r$. This is the only case where a *null* in a tuple $t$ makes the value of $f(t,r)$ identically equal to *false*. Formally:

**Proposition 1**

Let $R$ be a relation scheme, $X$, $Y \subseteq R$, such that $X \cap Y = \phi$ and $X \cup Y = R$, $f: X \rightarrow Y$ be a functional dependency in $R$, $r$ be an instance of $R$, and $t$ a tuple of $r$. Assume that $r-\{t\}$ has no nulls. Alternatively, consider all completions of $r-\{t\}$ iteratively.

$f(t,r) = true$ iff one of the following conditions holds

[T1] $t[XY]$ has no *nulls* and there exists no $t'$ in $r$ such that $t'[X]=t[X]$ and $t'[Y] \neq t[Y]$.

[T2] $t[Y]$ has a *null*, $t[X]$ has no *nulls* and there exists no $t'$ in $r$ such that $t[X]=t'[X]$.

[T3] $t[X]$ has a *null*, $t[Y]$ has no *nulls* and either no completion of $t[X]$ is in $r$, or if a completion of $t[X]$ is in $r$, say $t'[X]$, then $t[Y]=t'[Y]$.

$f(t,r) = false$ iff one of the following conditions holds

[F1] $t[XY]$ has no *nulls* and there exists a tuple $t'$ in $r$ such that $t[X]=t'[X]$ and $t[Y] \neq t'[Y]$

[F2] $t[X]$ has a *null*, $t[Y]$ has no *nulls* and both:

a.- all completions of $t[X]$ appear in $r$,

b.- $t[Y]$ is unique among all those completions.

$f(t,r) = unknown$ in all the other cases.

Examples of the above are given in figure 2. We say that a functional dependency $f$ (*strongly*) *holds* in an instance $r$ if $f(t,r)=true$ for every tuple $t$ in $r$. In addition, we say that a functional dependency $f$ *weakly holds* in an instance $r$ if $f(t,r) \neq false$ for every tuple $t$ in $r$. The second notion of satisfiability is justified intuitively since, in a framework of incomplete information, it is natural to weaken our expectations and allow for a margin of uncertainty in our semantic rules (as long as this does not lead to a certain denial - contradiction - of the constraint).

We now come to a separate issue which is the consideration of rules under which uncertain situations take specific interpretations when the requirement of satisfiability is enforced. That is, rules that guide us in substituting *nulls*. In this section we only informally discuss the rules. Their formalization and the examination of their properties is presented in section 6. A *null* may be substituted only if there is exactly one option making the dependency *true*. For instance, if the *null* appears among the $t[Y]$ values, $t[X]$ has no *nulls* and there is a tuple $t'$ in $r$ with $t[X]=t'[X]$, we may substitute the $t[Y]$ value with the $t'[Y]$ value. The justification of this substitution is two-fold. First, the resulting tuple has more information than the previous one. Second, and most important, this new information is not arbitrary - it is the only piece of information that makes the dependency *true*. The value which is substituted is the only value that a user can insert without the creation of an inconsistency. For substituting *nulls* in $t[X]$ the rule is more complicated and, unfortunately, is domain-dependent. One of the following two conditions must be met for such a substitution to take place.

(1) All completions of $t[X]$ appear in $r$, $t[Y]$ is not *null*, and there exists exactly one completion of $t[X]$ in $r$, say $t'[X]$, such that $t'[Y]=t[Y]$. The *null* in $t[X]$ may be substituted with the corresponding value in $t'[X]$.

(2) All completions of $t[X]$ appear in $r$ except one, $t[Y]$ is not *null*, and for all tuples $t'$ in $r$, such that $t'[X]$ is a completion of $t[X]$, $t'[Y]$ has no *nulls* and is distinct from $t[Y]$. The *null* in $t[X]$ may be substituted with the value of the domain of $X$ that does not appear in $r$.

Both conditions are not easy to test. In addition they seem unlikely to occur. For practical reasons, it may be better to leave the database incomplete, that is, prohibit substitutions of *nulls* in $t[X]$.

Before the conclusion of this section, we discuss the two observations we made in the previous section about two-tuple relations. The observations allow us to determine whether a dependency holds or is inferred by just looking at two-tuples. It can be trivially verified that both observations are valid when we consider the strong version of FD satisfiability. On the other hand, they are both false when the weak notion is considered. A counter-example of the first observation is given in the last instance $r_4$ which appears in figure 2. Note that any two-tuple combination in $r_4$, considered independently, makes the FD $f$ not *false*. But the dependency is *false* in the whole relation. To ensure that the observations are valid for both notions of satisfiability we require that tuples containing *nulls*, which make a dependency *false* for every possible substitution, do not appear in $r$. The test to find these tuples is very hard, being domain-dependent. On the other hand, we now argue that, in practice, it is unlikely that such tuples will appear in a database. For the simple case, where $X$ has only one attribute, this argument is intuitively justified. The "bad" case [F2] of proposition 1 requires all the domain values of the $X$ attribute to be in $r$ and any tuple which has a *null* for $X$ to disagree in the $Y$ values with all tuples in $r$. This amounts to the requirement that the number of actual determining objects is smaller than the number of determined objects. That is, a company gives more salaries than the number of employees it actually has! In a carefully designed database we would expect the domain of employee numbers to be sufficiently large - say, larger than the number of

the maximum number of tuples that may be inserted in the relation. Unfortunately, we can not apply our intuition as smoothly when $X$ has more attributes. After considering inference rules for FDs in the next section, we present in section 6, ways for testing efficiently weak and strong satisfiability for a set of FDs in a relation $r$.

## 5. Inference Rules for Functional Dependencies

One of the major applications of FDs is in the theory of normalization and schema design. In this section we will show how normalization theory and relational schema design can be applied in the presence of incomplete information. The examination of inference rules between FDs and in particular the establishment of sound and complete inference rules are of prime importance for this purpose. For the sake of simplicity in proofs (especially completeness proofs) we will obtain our results by reduction to a system which is equivalent to our system of extended FDs. We will first show an equivalence between our system and a well-axiomatized propositional logic system. The equivalence is between functional dependencies in our system and implication statements in the propositional logic. This equivalence will allow us to conclude that rules which are sound and complete for the implication statements have the same property for functional dependencies. Our approach is similar to that of [Fagin 77], but in a different environment.

System-C is a propositional logic system for unknown outcomes [Bertram 73]. It is a modal system which is not truth-functional. A unary operator, $\nabla$, which reads as "necessarily true" is added to the traditional operators of negation, disjunction, etc. C has been axiomatized. A detailed justification and explanation of the axioms is beyond the scope of this paper. We only note that some of the axioms comprise a set of axioms for classical two-valued logic, thus ensuring that everything provable in two-valued logic is also provable in C. The rest of the axioms give to C the modal interpretation and, in particular, the last axiom restricts C to a system of "logical necessity".

C has an unusual evaluation scheme † that uses the notion of two-valued tautologies. Let $P(p_1, p_2, \ldots, p_k)$ be a well-formed formula ($wff$) in C, expressed in terms of its atomic terms, and $\alpha = \{\alpha_i \mid i=1, 2, \ldots, n\}$ an assignment of truth values to $p_1, p_2, \ldots, p_n$. The evaluation of $P$ under $\alpha$, denoted by $V(P, \alpha)$ or simply $V(P)$ where $\alpha$ is understood, is defined by the following recursive rules:

1.- If $P$ is a tautology in the classical two-valued logic then $V(P) = true$

2.- if $P = p_i$ for some $i$, then $V(P) = \alpha_i$

3.- if $P = \neg Q$ then

$$V(P) = \begin{cases} true & \text{if } V(Q) = false \\ false & \text{if } V(Q) = true \\ unknown & \text{otherwise} \end{cases}$$

4.- if $P = Q \wedge S$ then

$$V(P) = \begin{cases} true & \text{if } V(Q) \text{ and } V(S) \text{ are } true \\ false & \text{if } V(Q) \text{ or } V(S) \text{ is } false \\ unknown & \text{otherwise} \end{cases}$$

5.- if $P = \nabla Q$, then

$$V(P) = \begin{cases} true & \text{if } V(Q) = true \\ false & \text{otherwise} \end{cases}$$

Rule.1 is always applied first and it is the reason why C is not truth-functional. The example here is $p \vee \neg p$. It is a two-valued tautology, thus having the value *true* in C. But if evaluated without rule.1 it has the value *unknown*. We now present a series of lemmas. The proofs are very similar to the one's in [Fagin 77] and they all appear in [Vassiliou 80].

**Lemma 1** [Vassiliou 80]

The function $V$ can be derived as the *least extension* of the evaluation function $V'$ in a classical two-valued logic system. ▪

Corollary

The system we are using and C are *equivalent* in that they have the same evaluation scheme. ▪

A C-tautology is a C wff which takes only the value *true* (under $V$). A C-theorem is a wff that is derived from the axioms of C. In [Bertram 73] it is proven that, given the particular evaluation scheme $V$, every C-tautology is a C-theorem and vice-versa (soundness and completeness). The reason for introducing C will now become apparent.

Implication is defined in the regular way: $P \Rightarrow Q := \neg P \vee Q$. We will consider a special type of implicational statement. Let $A$, $B$, $A_i$ denote propositional variables, $\overline{X}$, $\overline{Y}$, $\overline{Z}$ denote conjunctive terms of propositional variables, i.e. $\overline{X} = A \wedge B$ or simply $\overline{X} = AB$. The implicational statements of interest (denoted by $f$) have the form: $\overline{X} \Rightarrow \overline{Y}$. Notice the similarity with functional dependencies. From now on we will use the term "implicational statement" for any statement of this form. An implicational statement $f$ is *logically inferred* by a set of implicational statements $\overline{F} = \{f_1, f_2, \ldots, f_n\}$ if for every assignment of truth values $\alpha$ that gives to all $f_i$ in $\overline{F}$ the value *true* $\alpha(\overline{f})$ is also *true*. Similarly, we can define the notion of *weak logical inference* where we relax our requirements by having $\alpha(f) = \neg false$.

**Lemma 2** [Vassiliou 80] (Implicational Completeness)

The following inference rules are sound and complete for implicational statements in C.

[I1] if $\overline{Y} \subseteq \overline{X}$ then $\overline{X} \Rightarrow \overline{Y}$
[I2] if $\overline{X} \Rightarrow \overline{Y}$ and $\overline{Y} \Rightarrow \overline{Z}$ then $\overline{X} \Rightarrow \overline{Z}$
[I3] if $\overline{X} \Rightarrow \overline{Y}$ and $\overline{X} \Rightarrow \overline{Z}$ then $\overline{X} \Rightarrow \overline{YZ}$
[I4] if $\overline{X} \Rightarrow \overline{YZ}$ then $\overline{X} \Rightarrow \overline{Y}$ and $\overline{X} \Rightarrow \overline{Z}$

▪

As we noted before, implicational statements resemble, syntactically, functional dependencies. With the above lemma we establish a set of inference rules that produce all and only implicational statements. We now proceed to the major result, which is to show that the resemblance with functional dependencies is not coincidental. Rather, it is an equivalence.

† An evaluation scheme is a function $V$ from the set of propositional variables to the set of truth values $\{true, false, unknown\}$.

**Lemma 3** [Vassiliou 80]

Let $\alpha$ be an assignment of truth values, $s = \{t,t'\}$ a two-tuple relation, $X \rightarrow Y$ a functional dependency and $\overline{X} => \overline{Y}$ the corresponding implicational statement. For every $A$ in $X \cup Y$ suppose the following holds:

$$t[A] = t'[A] \quad iff \quad \alpha(\overline{A}) = true$$
$$t[A] \neq t'[A] \quad iff \quad \alpha(\overline{A}) = false$$
$$t[A] \text{ or } t'[A] = null \quad iff \quad \alpha(\overline{A}) = unknown$$

then: $X \rightarrow Y$ *strongly holds* in $s$ iff $\alpha(\overline{X} => \overline{Y}) = true$ ▪

**Lemma 4** [Vassiliou 80]

Consider the world of two-tuple relations. A functional dependency $X \rightarrow Y$ is inferred from a set of FDs $F$ iff $\overline{X} => \overline{Y}$ is a logical inference of $\overline{F}$. ▪

The above two lemmas show that there is an equivalence between functional dependencies (with null values) and strong satisfiability in two-tuple relations and implicational statements in C. As we have shown from the previous section the restriction "two-tuple relations" is only necessary for technical reasons in the case of strong satisfiability. Because of this equivalence and lemma 2, the following is a trivial consequence.

**Theorem 1**

Armstrong's inference rules are sound and complete for functional dependencies defined on relations with *nulls* and the requirement of strong satisfiability. ▪

With this result we may safely talk about decompositions and the theory of normalization applying even when *nulls* are allowed in relation instances. As was demonstrated in section 4, we cannot claim theoretically the same result in the case of weakly satisfiable dependencies (where we accept a dependency as long as it is guaranteed not to be *false*). On the other hand, if we impose the state and domain-dependent condition on allowable *nulls*, we show in the next section that the result holds for weak satisfiability in relation instances which we call "minimally incomplete".

## 6. Satisfiability for a Set of Functional Dependencies

In section 4 we discussed satisfiability for a single functional dependency. When we have a set of dependencies $F$, no dependency in $F$ can be tested for weak satisfiability *independently* from the others. Consequently, Armstrong's inference rules do not hold. The following example illustrates this fact. Consider the relation scheme $R(A, B, C)$, the FDs: $f_1 : A \rightarrow B$, $f_2 : B \rightarrow C$, and the instance $r$:

| $r$ | | |
|---|---|---|
| $A$ | $B$ | $C$ |
| $a_1$ | – | $c_1$ |
| $a_1$ | – | $c_2$ |

The functional dependencies $f_1$ and $f_2$ evaluated independently on $r$ take the value *unknown* (they are weakly satisfied). This is not the case when the dependencies are evaluated simultaneously. For $B \rightarrow C$ to hold in $r$, it must be that the two B-values in $r$ are distinct. Therefore, $A \rightarrow B$ is *false*. Informally, when an FD is satisfied in $r$, something more may be known about the possible values that the *nulls* in $r$ represent. Hence, the assumption that a *null* can be substituted with any domain value is not valid. This section deals with the above shortcomings.

A parenthesis concerning our notation. While $X$ may have several attributes, say $X=\{X_1,X_2, \cdots ,X_l\}$, we compare X-values directly with *null*. For a tuple $t$, $t[X]=null$ implies that one of the $X_i$ values is *null*. Similarly, $t[X] \neq null$ implies that no $X_i$ attribute value is *null*. We introduce a new type of constraint.

**Definition 1**

A *Null–Equality–Constraint*, (*NEC*), is a statement to the effect that two null values are equal (i.e. must take the same value in any substitution). For tuples $t_i$, $t_j$, and attributes $A$, $B$, a *NEC* is denoted as: $NEC: t_i[A]:=t_j[B]$. ▪

Null equality constraints introduce equivalence classes for null values. We now formalize the rules that allow for non-arbitrary increase of knowledge about null values in a relation where FDs are defined.

**Definition 2**

Given a relation $R$, a functional dependency $X \rightarrow Y$ embedded in $R$, and an instance $r$ of $R$, the *Null–Substitution–Rule* (NS-rule) corresponding to the FD $X \rightarrow Y$ is: If for two tuples $t_i$, $t_j$ in $r$ we have $t_i[X]=t_j[X] \neq null$ or the $NEC: t_i[X]:=t_j[X]$, then:

(a) if only one of $t_i[Y]$, $t_j[Y]$ is null, then this null is substituted with the non-null value of the other;

(b) if both $t_i[Y]$ and $t_j[Y]$ are null, then the null equality constraint $NEC: t_i[Y]:=t_j[Y]$ is introduced. ▪

**Proposition 2**

The NS-rules can only be applied a finite number of times on a relation instance $r$. To apply all rules takes time polynomial on the size of the relation instance.

Proof

(a) Finiteness

Initially, $r$ has a finite number of constant values and a finite number of null values with each null participating in a distinct equivalence class. The application of a rule *never* introduces a *new* constant value. Also, it may reduce but will never increase the number of equivalence classes (i.e. when a NEC is introduced). In the sequence of instances $r'$ produced after an NS-rule application, all elements are distinct. This suffices to show that the rules are a finite system.

(b) Complexity Analysis

The NS-rules are applied in several passes. In each pass, all NS-rules are applied for as many tuples as possible. In applying the NS-rule $X \rightarrow Y$ the instance $r$ may first be sorted in time $O(x \cdot n \cdot logn)$ where $x$ is the number of attributes in $X$ and $n$ is the number of tuples in $r$. When sorting, null values have the lowest precedence and are always distinct unless they belong to the same equivalence class. In this last case they appear together in the sorted relation. To apply $X \rightarrow Y$ in a pass requires the equation of Y-values in possibly more than one tuple (same equivalence class). Hence, a pass over the Y-values must be made for each change. This takes time $O(n^2)$. Since all rules are applied in each pass, the time required for a pass is $O(|F| \cdot (n^2 + x \cdot n \cdot logn))$ or, $O(|F| \cdot n^2)$ for $x$ substantially smaller than $n$. When we start in an instance with $p$ attributes we have at most $n \cdot p$ distinct symbols (constant values and nulls). Every pass reduces the number of distinct symbols, hence we have at most $n \cdot p$ passes. Therefore, no rule can be applied after $O(|F| \cdot n^3 \cdot p)$ time. † ▪

† According to a recent result by [Downey et al 80] the time complexity of the test is $O(|F| \cdot n \cdot \log(|F| \cdot n))$.

Given a relation $R$ and a set of FDs $F$ embedded in $R$, we say that an instance $r$ of $R$ is *minimally−incomplete* if no NS-rule can be applied on $r$. The intuitive meaning of a minimally incomplete relation state is that nothing more can be said about the nulls in this state. The importance of such states is demonstrated with the theorems that follow. The first two theorems present methods for testing FD-satisfiability. The test is done with an algorithm that works on relation instances where no nulls are present (Figure 3) and runs in time $O(|F|\cdot n\cdot logn)$. The third theorem makes the connection between "uniqueness" of a minimally incomplete state and satisfiability of FDs. This shows that the set of NS-rules is a finite Church-Rosser system.

**Theorem 2**

Let $R$ be a relation scheme, $F$ a set of FDs embedded in $R$, and $r$ an instance of $R$. Consider the application of the algorithm *TEST−FDs* with the following convention for null values:

*Convention:* Any equality comparison where a null is involved is positive. Also, any inequality comparison where a null is involved is positive, *unless* both values compared are null and they belong to the same equivalence class.

$F$ is *strongly−satisfied* in $r$ iff *TEST-FDs*$(r,F) = yes$

Proof.

The convention made implies that when testing an FD $X\rightarrow Y$ on two tuples $t_i$ and $t_j$ we have the following: if one of $t_i[X]$, $t_j[X]$ is null then the comparison $(t_i[X]=t_j[X])$ is *true*. If the previous comparison is *true* then so is the comparison $(t_i[Y]\neq t_j[Y])$, unless there exists a $NEC: t_i[Y]:=t_j[Y]$ (again we assume that at least one of $t_i[Y]$, $t_j[Y]$ is null). It may seem that with the above convention we have a problem in applying the algorithm *TEST−FDs*. † Two values that at some point were compared and found equal may at another point be found not equal. For $F$ not to be strongly satisfied, it suffices to find a completion of $r$ where the FDs are violated. In comparisons we consider *all* possible completions. Since Armstrong's rules are sound and complete we can test FDs for strong satisfiability independently. But notice that for the same dependency $X\rightarrow Y$ the same attribute values are never compared both for equality and for inequality when *TEST−FDs* is applied.

*if-part*

The assumption here is that *TEST−FDs*$(r,F) = yes$. We show that $F$ is strongly satisfied in $r$. Assume that there is an FD $X\rightarrow Y$ which is violated in $r$. That is, for two tuples $t_i$, $t_j$ in $r$, there exist completions $t'_i$, $t'_j$ such that $t'_i[X]=t'_j[X]$ and $t'_i[Y]\neq t'_j[Y]$. Trivially, when *TEST−FDs* is applied on $r$, the comparisons between the non-completed $X$ and $Y$ values of $t_i$ and $t_j$ are both positive, thus *TEST−FDs*$(r,F) = no$. A contradiction.

*only-if part*

Suppose that $F$ is strongly satisfied in $r$ and that *TEST−FDs* returns with a *no* when applied on $r$ with our convention for nulls. The contradiction is clear. Any time the equality comparison for the X-values of two tuples and the inequality comparison for the Y-values of the same tuples are found positive, a completion of $r$ is

† Another problem is sorting the null values under the above convention. Alternatively, another version of *TEST−FDs* may be used, where the relation is not sorted and each tuple is tested against every other tuple in the relation. The running time is now $O(|F|\cdot n^2)$.

Figure 3 Testing for FD-satisfiability

```
Test a set of FDs F in r for consistency.

TEST_FDs (parameters: r, F) returns(boolean)

begin
  comment: comparison is based on
           lexicographical order
  comment: read_next_tuple(r), reads the next
    tuple in order from r and returns EOF
    if no more tuples.

  for each X→Y in F do
    begin
      sort relation r on X
      t_first ← read_next_tuple(r)
      while t_first ≠ EOF do
        begin
          t_next ← read_next_tuple(r)
          while t_next[X] ≠ EOF and t_next[X] = t_first[X] do
            begin
              if t_next[Y] ≠ t_first[Y]
                then return(no)
                else t_next ← read_next_tuple(r)
              end
          t_first ← t_next
          end
      end
  return(yes)
  end
```

**Complexity Analysis**

The algorithm runs in $O(|F|\cdot n\cdot logn)$ time, where $n$ is the size (number of tuples) of $r$ and $|F|$ is the number of dependencies. Each FD is tested in time *nlogn*, the time to sort the relation.

*Additional Assumptions.* If bucket sort is used, sorting takes time $O(n\ p)$ where $p$ is the number of attributes in $X$ for a dependency $X\rightarrow Y$. Furthermore, if there is only one dependency (e.g. BCNF with one key), and the relation is already sorted, the test requires time linear on the relation size.

illustrated where the FD $X \rightarrow Y$ is violated. ▪

**Theorem 3**

Let $R$ be a relation scheme, $F$ a set of FDs embedded in $R$, and $r$ a *minimally−incomplete* instance of $R$. Consider the application of the algorithm *TEST−FDs* with the following convention for null values:

*Convention:* Any inequality comparison where a null is involved is negative. Also, any equality comparison where a null is involved is negative, *unless* both values compared are null and they belong to the same equivalence class.

$F$ is *weakly−satisfied* in $r$ iff *TEST-FDs*$(r,F)$ = *yes*

Proof.

As in the previous theorem, we consider the problem that may be introduced with our convention for null values. It may be that the same two values are compared and found equal at some point and at another point not equal. We show that when *TEST−FDs* is applied in a minimally incomplete state of $r$ this never happens. Consider two tuples $t_i$ and $t_j$ in $r$. Suppose that $t_i[X]$ is null and $t_j[X]$ is not. If $X$ appears only on the left of dependencies, we have no problem (*TEST−FDs* makes only equality comparisons on $X$ values). Similarly for $X$ appearing only on the right of dependencies. Consider the case where $X$ appears on the left of a dependency $X \rightarrow Y$ and on the right of a dependency $Z \rightarrow X$. In testing $X \rightarrow Y$ we assume that $t_i[X] \neq t_j[X]$. In testing $Z \rightarrow X$, we will only consider X-values when $t_i[Z] = t_j[Z]$. If the state is minimally incomplete, we wouldn't have $t_i[X]$ null (by application of the NS-rule). The case of both $t_i[X]$ and $t_j[X]$ being null is treated with similar arguments. In this case the outcome of the evaluation depends on whether or not the nulls belong to the same equivalence class. We also note that the convention allows for sorting. Null values are considered distinct and their order is not important. (They are never equated unless they are in the same equivalence class in which case they appear together.)

*if-part*

The substitution of nulls with different values from the one's appearing in $r$ illustrates a completion of $r$ where all FDs are satisfied.

*only-if-part*

We show that if there is a completion of $r$ where the FDs are satisfied, then *TEST−FDs*$(r,F)$ = *yes*. Suppose $r'$ is such a completion. If *TEST−FDs* has a *no* answer there must exist two tuples $t_i$ and $t_j$ in $r$ such that for a functional dependency $X \rightarrow Y$ the comparisons $(t_i[X] = t_j[X])$ and $(t_i[Y] \neq t_j[Y])$ are both positive. The first comparison is positive under our convention when both $t_i[X]$, $t_j[X]$ are equal constants, or, both are nulls in the same equivalence class. In this last case, they both have the same completions in $r'$ (as in any other completion of $r$). Similarly, for the second comparison to be positive it must be that both $t_i[Y]$ and $t_j[Y]$ are distinct constant values. It follows immediately from the above arguments that the FD $X \rightarrow Y$ is violated for the two tuple completions in $r'$. A contradiction. ▪

Note that the test for strong satisfiability is less expensive than the one for weak satisfiability since it does not require a minimally incomplete instance. This comes as no surprise - very few relation instances are strongly-consistent.

The NS-rules applied in a different order may result in different minimally incomplete states. This is illustrated with an example. Consider a relation $R$ with three attributes, the dependencies $A \rightarrow B$, $C \rightarrow B$, and the instance $r$ (figure 5). Applying the rule $A \rightarrow B$ first we get a minimally incomplete state $r'$. On the other hand if we first apply $C \rightarrow B$ we get a different minimally incomplete state $r''$.

| $r$ | | |
|---|---|---|
| $A$ | $B$ | $C$ |
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_1$ | − | $c_2$ |

| $r'$ | | |
|---|---|---|
| $A$ | $B$ | $C$ |
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_1$ | $b_1$ | $c_2$ |

| $r''$ | | |
|---|---|---|
| $A$ | $B$ | $C$ |
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_1$ | $b_2$ | $c_2$ |

Figure 5

From definition 2, an NS-rule for an FD $X \rightarrow Y$ is applied if there exist tuples $t$, $u$ such that $t[X] = u[X]$ and one or both of $t[Y]$, $u[Y]$ is null. We now extend the notion of an NS-rule application and we assume that a rule may be applied even if none of $t[Y]$, $u[Y]$ is null, but as constants they are distinct. In this case they are both replaced by the *inconsistent* element (the *nothing* data value). This triggers the replacement with *nothing* of all constants that are equal to them. In our example, if $A \rightarrow B$ is applied first producing $r'$, then $C \rightarrow B$ can be applied on $r'$ resulting in an instance with all values in the $B$ column equal to *nothing*. It is easily observed that the application of the rules in reverse order will produce the same instance. The theorem below is proven in [Graham 80]

**Theorem** 4 [Graham 80]

Given a relation scheme $R$, a set of FDs $F$, and an instance $r$ of $R$. Then,

(a) The application of the NS-rules will produce a *unique* minimally incomplete instance (the NS-rules constitute a Church-Rosser system).

(b) $F$ is weakly-satisfied in $r$ iff there is no *nothing* value in the resulting minimally incomplete instance. ▪

For the proof of the theorem the notion of congruence closure is used [Downey et al 80]. The construction of the graph from the instance $r$ for the purpose of applying congruence closure is reversed. The result is a relation instance (possibly with *nothing* values). This instance is unique and is exactly the minimally incomplete instance produced from the NS-rule applications. The theorem above verifies that in any minimally incomplete instance, produced from NS-rules application, the test for satisfiability will determine correctly whether the FDs are satisfied.

## 7. Concluding Remarks

Functional dependencies were examined in the light of incomplete information in a database. Our results are both encouraging and, in some respect, discouraging concerning the possibility of allowing *nulls* with no restrictions in relations. Two notions of FD satisfiability were introduced. The first is the regular one which requires that an FD takes the truth-value *true* when it is interpreted as a predicate on relation instances. In addition, a weak notion of satisfiability was defined which allows for uncertainty about the validity of an FD as long as this uncertainty does not introduce

contradictions. It was shown that a *null* value does not have an impact on the validity of an FD if it appears in special places. This is because there exists a substitution of this null (possibly all substitutions) which results in having the dependency satisfied. Furthermore, to find these cases of satisfiability is not computationally hard.

On the other hand, there are some extreme cases where all substitutions of the *null* result in inconsistent states. This occurs with the weak notion of satisfiability and domain-size restrictions. The test to find such cases is domain and state-dependent, thus having an unacceptable complexity for practical considerations. It was argued that in practice the above extreme cases are unlikely to appear, provided that the dependencies are carefully defined (e.g. on attributes with large domains).

Weak satisfiability seems to be the more important and interesting notion from a practical point. Databases are usually "overconstrained". That is, there is a large number of semantic constraints (rules) which would make sense for a database. However, database systems do not usually have the ability to maintain all these constraints. The test of constraint validity in a database instance, apart from being prohibitively expensive, results mainly in verifying that most of the data is "dirty". On the other hand, null values and weak satisfiability allow constraints to be valid in more instances.

A basic result of this paper is the verification for extended FDs of the soundness and completeness of the same inference rules that were sound and complete for FDs with no *nulls*. This allows us to conclude that all work on normalization, decomposition, etc. where FDs are involved can be applied directly in our framework of incomplete information.

We now discuss the importance of our results taken together with on-going research. As was mentioned in the introduction, the universal relation assumption is questioned both on practical and theoretical grounds. With our work we have provided a partial reply to the practical attacks on the possibility of a universal relation instance. More realistic instances may now be perceived; the ones where *nulls* are allowed. In [Bernstein and Goodman 80] it is shown that the requirement of having the universal relation assumption defeats the purpose of normalization, which is to avoid update anomalies. [Bernstein and Goodman 80] attempted the use of *nulls* to overcome the anomalies, but decided that for each semantic they tried, certain bizarre behavior was forced. It is our conjecture that a more careful approach to *null* interpretation and treatment, like the one we presented here, contributes to the attempts to bridge the gap between the static (normalization) and the dynamic (modification) properties of the database [Graham and Vassiliou 80]. A "weaker" version of the universal relation assumption is conceivable that allows for universal instances (with nulls) where the dependencies are only weakly-satisfied. In addition to the work reported here, more research is needed on the semantics of the ways a database acquires information. This acquisition may be internal (non-ambiguous substitution of *nulls*), or external (modification operations by the users).

**References**


[Beeri et al 78]

Beeri C., Bernstein P.A., and N.Goodman, **A Sophisticate's Introduction to Data Base Normalization Theory,** *Proceedings, 4th International Conference on Very Large Data Bases, West Berlin, Germany, September 1978, pp. 113-124.*

[Bernstein and Goodman 80]

Bernstein P.A., Goodman N., **What Does Boyce-Codd Normal Form Do?,** *Proceedings, 6th International Conference on Very Large Data Bases, Montreal, Canada, October 1980.*

[Bertram 73]

Bertram B., **A Logic for Unknown Outcomes,** *Dept. of Computer Science, Rutgers, The State University of N.J., CBM-TM-35, December 1973.*

[Codd 70]

Codd E.F., **A Relational Model of Data for Large Shared Data Banks,** *Comm. ACM 13, 6, July 1970, pp.377-3.*

[Codd 72]

Codd E.F., **Further Normalization of the Data Base Relational Model,** *in Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 33-64.*

[Codd 75]

Codd E.F., **Understanding Relations,** *(installment no.7), ACM SIGMOD FDT Bulletin 7, 3-4, 1975, pp.23-28.*

[Date 77]

Date C.J., **An Introduction to Data Base Systems,** *2nd ed., Addison-Wesley, Reading, MA, 1977.*

[Downey et al 80]

Downey P., Sethi R., Tarjan P.E., **Variations on the Common Subexpression Problem,** *J. ACM (to appear), 1980.*

[Fagin 77]

Fagin R., **Functional Dependencies in a Relational database and Propositional Logic,** *IBM J. of Res. and Dev., Vol. 21, No. 6, Nov. 1977, pp. 534-544.*

[Graham 80]

Graham M., **On the Church-Rosser Property of the Tableau Chase** *unpublished manuscript, 1980.*

[Graham and Vassiliou 80]

Graham M., Vassiliou Y., **Weakening the Universal Relation Assumption,** *(working paper), 1980.*

[Lipski 79]

Lipski W. Jr., **On Semantic Issues Connected with Incomplete Information Databases**, *ACM Transactions On Database Systems, Vol. 4, No. 3, September 1979, pp. 262-296.*

[Stoy 77]

Stoy J.E., **Denotational Semantics, the Scott-Straychey Approach in Programming Language Theory**, *MIT press, 1977.*

[Vassiliou 79]

Vassiliou Y., **Null Values in Database Management - A Denotational Semantics Approach**, *ACM/SIGMOD International Symposium on Management of Data, May-June 1977, pp. 162-169.*

[Vassiliou 80]

Vassiliou Y., **A Formal Treatment of Imperfection in Database Management**, *Ph.D. thesis, University of Toronto, 1980.*