Synthesizing Third Normal Form Relations from Functional Dependencies

PHILIP A. BERNSTEIN
University of Toronto

It has been proposed that the description of a relational database can be formulated as a set of functional relationships among database attributes. These functional relationships can then be used to synthesize algorithmically a relational schema. It is the purpose of this paper to present an effective procedure for performing such a synthesis. The schema that results from this procedure is proved to be in Codd's third normal form and to contain the fewest possible number of relations. Problems with earlier attempts to construct such a procedure are also discussed.

Key Words and Phrases: database schema, functional dependency, relational model, semantics of data, third normal form
CR Categories: 3.50, 4.33

1. INTRODUCTION

Research on the relational database model has shown that the functional relationship is an important concept when one is considering how to group attributes into relations [3, 8–12]. It has been proposed by some that the basic description of a database can be formulated purely as a set of such functional relationships from which the relational schema can be synthesized algorithmically [3, 12]. It is the purpose of this paper to develop a provably sound and effective procedure for synthesizing relations satisfying Codd's third normal form from a given set of functional relationships. Also, the schema synthesized by our procedure will be shown to contain a minimal number of relations.

This method assumes the existence of at most one functional relationship connecting any one set of attributes to another. This uniqueness assumption, which is required by all earlier methods as well, raises difficult semantic questions that will be discussed in detail.

The first three sections are an introduction to the problem of synthesizing relations from functional dependencies. Section 2 of the paper reviews the relational model, the concept of functional dependency, and Codd's normal forms. Two new concepts are introduced, "superkey" and "embodiment." Section 3 outlines the

This work was supported in part by the National Research Council of Canada.
Author's present address: Center for Research in Computing Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138.

general synthesis problem and presents a simple algorithm for synthesizing relations from functional dependencies. Section 4 introduces Armstrong's axiomatization of functional dependencies and comments on the uniqueness assumption and other semantic considerations of this theory. The main synthesis algorithm is described in Section 5. Section 6 examines Codd's third normal form property applied to relations that are synthesized by two versions of the synthesis algorithm. The section concludes with the presentation of a new algorithm for synthesizing provably third normal form relations. In section 7, the synthesized schema is shown to be minimal in size.

2. THE RELATIONAL MODEL

2.1 Relations

In Codd's relational database model, mathematical relations over a set of domains are used to describe connections among data items [7]. However, not all relations serve equally well in describing these connections [8]. To judge the efficacy of various classes of relations, we begin by reviewing the terminology associated with the relational model.

Conceptually, a *relation* is a table in which each column corresponds to a distinct *attribute* and each row to a distinct *entity* (or *tuple*). For each attribute there is a set of possible associated values, called the *domain* of that attribute. It is common for different attributes to share a single domain. For example, the attributes QUANTITY_IN_STOCK and SIZE_OF_CLASS both assume values from the domain called NONNEGATIVE INTEGERS.

An ⟨entity, attribute⟩ entry in a relation is a value associated with the entity chosen from the domain of the attribute. Formally, a relation is a (finite) subset of the Cartesian product of the domains associated with the relation's attributes.

The notation for describing the structure of a database relation includes a relation name (say **R**) and a set of attributes in **R** (say $\{A_1, A_2, \ldots, A_n\}$) and is written $\mathbf{R}(A_1, A_2, \ldots, A_n)$; e.g. see Figure (1a). The ordering of attributes is immaterial since attribute names are distinct within a relation. (This is one reason for distinguishing between attributes and domains.) Notationally we will use upper case letters near the beginning of the alphabet for simple (i.e. singleton) attributes and ones near the end of the alphabet for composite (i.e. groups of) attributes.

The set of entities that comprise a relation normally changes over time as entities are inserted, deleted, and modified. This is one important way that database relations differ from mathematical relations.

The word "relation" is often used in the literature to describe both the structure of the relation (e.g. $\mathbf{R}(A_1, \ldots, A_n)$), called its *intention*, which is static, and the

EMPLOYEE(EMP#, NAME, DEPT#)	EMP# → NAME
DEPARTMENT(DEPT#, MGR#)	EMP# → DEPT#
INVENTORY(STOCK#, DEPT#, QTY)	DEPT# → MGR#
	STOCK#, DEPT# → QTY
(a)	(b)

Fig. 1. Relations and functional dependencies: (a) an example of a relational schema (underlined attributes are keys); (b) functional dependencies for the schema of (a).

set of tuples in the relation, called its *extension. In what follows, the word "relation" will refer to an intention unless explicitly stated otherwise.* That is, we will usually be referring to the structure of a relation, rather than the set of tuples themselves.

2.2 Functional Dependencies

As we will see in later sections, it is important to consider functional relationships when choosing how to group attributes into relations. Functional relationships among database attributes are formalized in the concept of functional dependency.

Let A and B be attributes, let DOM(A) be the domain of A and DOM(B) be the domain of B, and let f be a time-varying function such that f: DOM(A) $\rightarrow$ DOM(B). f is not a function in the precise mathematical sense because we allow the extension of f to vary over time in the same sense that we allow extensions of database relations to change over time. That is, if f is thought of as a set of ordered pairs $\{(a, b) \mid a \in \text{DOM(A)}$ and $b \in \text{DOM(B)}\}$, then at every point in time for a given value of $a \in$ DOM(A) there will be at most one value of $b \in$ DOM(B). To distinguish f from a mathematical function, we call f a *functional dependency* (abbreviated FD). For notational convenience, we generally leave out the "DOM" and write f: A $\rightarrow$ B. If there is an FD f: A $\rightarrow$ B, then B is said to be *functionally dependent* on A.

The above definitions are generalized in the obvious way for functional dependencies over compound attributes. If $X = \{A_1, \ldots, A_n\}$ and $Y = \{B_1, \ldots, B_m\}$ are sets of attributes, then f: X $\rightarrow$ Y means f: DOM(A_1) $\times \cdots \times$ DOM(A_n) $\rightarrow$ DOM(B_1) $\times \cdots \times$ DOM(B_m). We will normally leave off the set notation in FDs and write f: $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$ simply as f: $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$. As an example, the functional dependencies for attributes in Figure 1(a) are given in Figure 1(b).

In this paper we will assume that for any two sets of attributes X and Y, there is at most one FD X $\rightarrow$ Y. Attributes may need to be renamed to guarantee this assumption. This restriction is an important one, and it will be discussed in detail in Section 4.2. We will also show later that nonfunctional relationships need not satisfy this uniqueness assumption.

Given this assumption, if f: A $\rightarrow$ B we will frequently write A $\rightarrow$ B as an abbreviation. The notation A $\nrightarrow$ B means that there is no FD A $\rightarrow$ B that is of interest (although at a given point in time in some relation, it may be true that no value of A has more than one corresponding value of B).

Let $\mathbf{R}(A_1, \ldots, A_n)$ be a relation and let X be a subset of $\{A_1, \ldots, A_n\}$. X is called a *key* of **R** if every attribute in $\{A_1, \ldots, A_n\}$ that is not in X is functionally dependent upon X and if no subset of X has this property. Clearly a relation can have many keys. A *superkey* of **R** is any set of attributes in **R** that contains a key of **R**. (Every key is also a superkey.) The concept of superkey is introduced mainly to simplify our proofs in later sections.

2.3 Operations on Relations

In his original description of the relational model, Codd introduced the *relational algebra* as a data manipulation language for the relational database model [7]. There are two basic relational algebraic operations that will be of some interest to us: projection and join.

The *projection* of the extension of a relation **R** on a subset of its attributes X is the set of tuples obtained by excising those attributes not in X. If two tuples are now indistinguishable because they only differed in the attributes that were eliminated, then they are "merged" into a single tuple. That is, the result of the projection must be a subset of the Cartesian product of the domains associated with the attributes of X.

The *join* operation is used to make a connection between attributes that appear in different relations. The only join operation we shall consider here is the *natural join* (i.e. equality join). The natural join of the extension of a relation **R**(A, B) with the extension of relation **S**(B, C) on domain B, denoted **R*S**, is defined to be $\{(a, b, c) \mid (a, b) \in \mathbf{R} \text{ and } (b, c) \in \mathbf{S}\}$. That is, it links together all values of A and C that are related to common B values.

2.4 Schemas

The purpose of any data model, relational or otherwise, is to allow the user of the model to describe and manipulate those relationsips among objects in the real world that he intends to store in the database. In the relational model, such a collection of relationships is represented in a relational schema. A *relational schema* consists of a set of database relations and for each relation the specification of one or more keys (e.g. see Figure 1(a)).

We will say that a functional dependency $\mathbf{X} \rightarrow$ A is embodied in a relation **R** if X is a key of **R** and A is any other attribute of **R**. The set of FDs embodied in a schema is the union of the FDs embodied in all of the relations of the schema.

Note that this formulation of schema is a modification of Codd's [8], where FDs are given as information *additional to* the relations and their keys. We have adopted this modified notion of schema for several reasons. First, all data definition languages that we know of only allow the specification of relations and keys. Second, our definition of schema eliminates the need to talk explicitly of FDs. The FDs exist implicitly by virtue of our definition of the keys and embodiment. Third, we will see that the third normal form organizes a relational schema so that every FD that is given as external information is either embodied in some relation or can be recovered from embodied FDs by the join and projection operations. Taking this notion of embodiment as a primitive concept simplifies the ensuing theory considerably.

2.5 Normalization

Codd observed that certain relations have structural properties that are undesirable for describing databases. This led him to define a series of three normal forms for relations.

First, relation-valued domains are excluded from relations. A relation is in *first normal form* (abbreviated 1NF) if each domain contains simple values. There are two main advantages to 1NF [7]. First, it allows the database to be viewed as a collection of tables—a very simple and understandable structure. Second, it permits the definition of a small class of primitive operators that are capable of manipulating relations to obtain all necessary logical connections among attributes.

The second and third normal forms are introduced to correct problems caused by certain functional dependencies. To examine these problems, consider the re-

lation **DEPT_INV** (STOCK#, DEPT#, QTY, MGR#) obtained by joining the **DEPARTMENT** and **INVENTORY** relations of Figure 1(a) on the attribute DEPT#. The insertion of the first inventory item for a particular DEPT# into the extension of **DEPT_INV** creates a new connection between that DEPT# and its MGR#. The deletion of the last inventory item for a particular DEPT# loses the connection between that DEPT# and its MGR#. These side effects, called *insertion-deletion anomalies*, only occur when the first or last tuple of a DEPT# is inserted or deleted. Also, the repetition of the connection between a DEPT# and its MGR# for each STOCK# in the DEPT# can lead to an inconsistent relation if arbitrary updates on individual tuples are permitted. These problems arise because MGR# is functionally dependent on only part of the key STOCK#, DEPT#. To eliminate these problems from **DEPT_INV**, **DEPT_INV** must be put into second normal form.

A partial dependency occurs when an attribute is functionally dependent upon a subset of a set of attributes. Let $f: A_1, \ldots, A_n \rightarrow B$ and $g: A_1, \ldots, A_m \rightarrow B$ be functional dependencies where $m < n$. The attributes $A_{m+1}, A_{m+2}, \ldots, A_n$ are extraneous in f since $A_1, \ldots, A_m$ are sufficient to functionally determine B. In this case, B is said to be *partially dependent* on $A_1, \ldots, A_n$. If for a given f there is no g with the above property, then B is *fully dependent* on $A_1, \ldots, A_n$. That is, there are no extraneous attributes in the domain of f.

If an attribute A_i appears in any key of **R** then it is said to be *prime* in **R**. Otherwise it is *nonprime* in **R**. A relation is in *second normal form* (abbreviated 2NF) if it is in 1NF and each of its nonprime attributes is fully dependent upon every key [8]. The relation **DEPT_INV** (STOCK#, DEPT#, QTY, MGR#) is not in 2NF because MGR# is a nonprime attribute and is partially dependent on the key STOCK#, DEPT#. The relations **DEPT** and **INVENTORY** in Figure 1(a) *are* in 2NF.

Consider now the relation **EMP_DEPT** (EMP#, NAME, DEPT#, MGR#) obtained by joining the **EMPLOYEE** and **DEPARTMENT** relations of Figure 1(a) on DEPT#. Although **EMP_DEPT** is in 2NF, it displays the same problems as **DEPT_INV**. Inserting or deleting the first EMP# in a particular DEPT# creates an anomaly, for a DEPT#-MGR# connection is created or destroyed in the process. The repetition of the DEPT#-MGR# connection for each EMP# in the DEPT# creates the same consistency problem as in **DEPT_INV**. In this case, the problems arise because MGR# is functionally dependent on the key EMP# via the attribute DEPT#. To eliminate the problems, the relations **EMP_DEPT** must be put into third normal form.

Let $\mathbf{R}(A_1, \ldots, A_n)$ be a relation. An attribute A_i is *transitively dependent* upon a set of attributes X if there exists a set of attributes $Y \subseteq \{A_1, \ldots, A_n\}$ such that $X \rightarrow Y$, $Y \nrightarrow X$, and $Y \rightarrow A_i$ with A_i not an element of X or Y.

A relation is in *third normal form* (abbreviated 3NF) if none of its nonprime attributes are transitively dependent upon any key [8]. A 3NF relation is also in 2NF, for if an attribute A_i is partially dependent on a key X, then A_i is transitively dependent on X since $X \rightarrow X'$, $X' \nrightarrow X$, and $X' \rightarrow A_i$ for some $X' \subset X$. The relation **EMP_DEPT** is not in 3NF because MGR# is nonprime and is transitively dependent upon the key EMP#. All of the relations in Figure 1(a) are in

3NF (and hence 2NF), given the FDs of Figure 1(b). Further examples of normal form relations and surrounding problems can be found in [7–9].

3. SYNTHESIZING A RELATIONAL SCHEMA

3.1 The Synthesis Problem and Nonfunctional Relationships

Codd showed that by applying simple decomposition steps to a 1NF relation in which the FDs were known, the relation could be split up into a set of relations in 3NF that embodies all of the FDs [8]. In [3] it was proposed that since the FDs completely determine whether or not a relation is in 3NF, one could choose the FDs as the basic concept and *build* 3NF relations from them. In advancing this proposal, an efficient algorithmic technique was presented to actually construct relations from FDs. In this paper we present an improved algorithm and then discuss properties of schemas synthesized by this algorithm.

The approach of building a relational schema from FDs rests entirely on the ability to represent *all* data relationships as FDs. Clearly though, not every logical connection in the world is functional. Nevertheless, we claim that all connections among attributes in a database description *can be represented* by FDs. As long as connections are functional there is of course no problem. Nonfunctional connections require special treatment.

A nonfunctional connection f among a group of attributes $A_1, A_2, \ldots, A_n$ will be represented as the following FD: $f: A_1, A_2, \ldots, A_n \rightarrow \Theta$. Θ is an attribute that is unique to f; it does not appear in any other FD. Each FD representing a nonfunctional relationship has its own private Θ attribute. The underlying domain for all of these Θ attributes is the set $\{0, 1\}$. For each element $(a_1, a_2, \ldots, a_n) \in \mathrm{DOM}(A_1) \times \mathrm{DOM}(A_2) \times \cdots \times \mathrm{DOM}(A_n)$, $f(a_1, a_2, \ldots, a_n) = 1$ if and only if $(a_1, a_2, \ldots, a_n)$ is related under f. Thus, the extension of f completely defines a nonfunctional relationship among $A_1, \ldots, A_n$. For example a nonfunctional relationship between a DRIVER and AUTOMOBILE, where each AUTOMOBILE can be driven by more than one DRIVER and each DRIVER can drive more than one AUTOMOBILE, is represented by the FD DRIVER,AUTOMOBILE $\rightarrow \Theta_1$.

Note that more than one nonfunctional relationship can exist among a set of attributes without violating the uniqueness assumption of FDs. For example we can have a second relationship between DRIVER and AUTOMOBILE that indicates ownership: DRIVER,AUTOMOBILE $\rightarrow \Theta_2$. By assigning a unique Θ to each nonfunctional relationship, the uniqueness assumption for FDs is retained.

This Θ notation allows us to represent all nonfunctional relationships as FDs. The synthesis algorithm will produce approximately one relation for each of these nonfunctional relationships. In Section 5 we shall show precisely how each of these "nonfunctional FDs" becomes embodied in the synthesized relational schema.

3.2 Formalizing the Synthesis Problem

Although the motivation for the synthesis problem is from database management, one can formalize the problem in purely symbolic terms as follows. We are given a set S of symbols (i.e. attributes) and a set F of mappings of sets of symbols into symbols (i.e. FDs). The problem is to find a collection $C = \{C_1, \ldots, C_m\}$ of

We are given the following set of FDs:

f_1: A → B f_4: B → D
f_2: A → C f_5: D → B
f_3: B → C f_6: ABE → F

We group the FDs according to common left-hand sides, obtaining four groups:

$g_1 = \{f_1, f_2\}$ $g_3 = \{f_5\}$
$g_2 = \{f_3, f_4\}$ $g_4 = \{f_6\}$

For each group we construct a relation consisting of all of the attributes in the group:

$\mathbf{R}_1$(A,B,C) $\mathbf{R}_3$(D,B)
$\mathbf{R}_2$(B,C,D) $\mathbf{R}_4$(A,E,B,F)

where the underscored attributes are keys.

Fig. 2. Deriving a schema from FDs

subsets of S (i.e. a collection of relations) and for each C_i a collection of subsets of C_i (i.e. a collection of keys for each relation) satisfying three properties: First, F is "embodied" in C (i.e. the relations embody the given FDs). Second, each C_i can have no transitive dependencies (i.e. it is in 3NF). Third, the cardinality of C is minimal.

This treatment of the problem is still somewhat fuzzy since we have not yet discussed the algebraic rules for composing FDs. To motivate the need for these rules, we present a simple synthesis algorithm. This algorithm ignores algebraic considerations and will be shown to be inadequate.

3.3 A Simple Synthesis Procedure

One (overly) simple way to obtain relations from a given set of FDs is to group together all attributes that are functionally dependent upon the same set of attributes. This suggests the following procedure. First, partition the given set of FDs into groups such that all of the FDs in each group have identical left sides. Then, for each group construct a relation consisting of all the attributes appearing in that group. The left side of the FD in each group is a key of the corresponding relation. For example see Figure 2.

Several undesirable properties of this method can be seen in the example. First, the synthesized relations are not in 3NF. For example, in relation $\mathbf{R}_1$ of Figure 2, C is transitively dependent on the key A. In $\mathbf{R}_4$, B is partially dependent on the key AE. The unnormalized relations are due to redundancies in the given set of FDs. We will see later that f_2 is redundant and that B is an extraneous attribute in f_6.

Second, the left sides of FDs are not necessarily keys of the relations, although they are always superkeys. In $\mathbf{R}_4$ ABE is a superkey, but not a key, since B is extraneous.

Third, this procedure synthesizes too many relations. Since f_4 and f_5 are inverses of each other, the relation $\mathbf{R}_3$ is extraneous. This results from a failure of the procedure when constructing $\mathbf{R}_2$ to recognize D as a second key by virtue of f_5, rather than to put f_5 into a separate relation.

To solve these problems, we must first formalize the concept of a redundant FD. We will then return to a presentation of a synthesis algorithm that overcomes the above difficulties.

4. THE ALGEBRA AND SEMANTICS OF FUNCTIONAL DEPENDENCIES

4.1 Armstrong's Axiomatization of Functional Dependencies

The complete axiomatization of FDs given by Armstrong [1] provides a theoretical background to the study of the algebra of FDs that is treated in later sections. Armstrong shows that if a given set of FDs exist in (the extension of) a relation, then any FDs that can be derived from the given set by using the axioms must also exist. Armstrong presents several equivalent axiomatizations of FDs. The one we will use is based on properties of FDs proved by Delobel and Casey [10]. They are

A1. (reflexivity) $X \rightarrow X$.

A2. (augmentation) if $X \rightarrow Z$ then $X + Y \rightarrow Z$.

A3. (pseudotransitivity) if $X \rightarrow Y$ and $Y + Z \rightarrow W$ then $X + Z \rightarrow W$.

where the symbol "+" means "set union" (of not necessarily disjoint sets).

If $\mathbf{R}(A, B)$ is a relation, then axiom A1 can be applied with $X = \{A, B\}$ to show that $A, B \rightarrow A, B$ or with $X = \{A\}$ to show that $A \rightarrow A$.

The meaning of A2 is simply that if $f: X \rightarrow Z$, then one can create another FD, g, where the domain of g includes X as well as some other extraneous attributes Y whose values have no effect on the value of Z selected by g. So, knowing that $A \rightarrow A$, we can obtain $A, B \rightarrow A$ (i.e. $X = \{A\}$, $Z = \{A\}$, and $Y = \{B\}$).

Axiom A3 is a substitution rule for composing FDs. Let $f: X \rightarrow Y$ and $g: Y + Z \rightarrow W$. The axiom claims that there is an $h: X + Z \rightarrow W$. To see where h comes from, consider the application of h to a given $x \in \mathrm{DOM}(X)$ and $z \in \mathrm{DOM}(Z)$ in two steps. First, f is applied to x, yielding a unique $y \in \mathrm{DOM}(Y)$. Second, g is applied to y and z, yielding a unique $w \in \mathrm{DOM}(W)$ and thereby completing the application of h. Symbolically we can say $h(x, z)$ is defined to be $g(f(x), z)$. Also, note that in the statement of axiom A3 if Z is the null set, then pseudotransitivity becomes simple transitivity.

Let G be a set of FDs. The *closure* of G, denoted G^+, is defined to be the smallest superset of G that is closed under A1, A2, and A3. For a given G, G^+ can be shown to be unique. By Armstrong's theory we know that if G is a given set of FDs for a relation $\mathbf{R}$, then each FD in G^+ also exists in $\mathbf{R}$.

An FD $g \in G$ is *redundant in* G if $G^+ = (G - \{g\})^+$. H is a *nonredundant covering* of a given set of FDs G if $G^+ = H^+$ and H contains no redundant FDs.

An important property of FDs that will be used later to prove a number of theorems is stated in Lemma 1. It is based on the concept of a "derivation," which we will informally consider to be a series of applications of Armstrong's axioms on a given set of FDs. A formal development appears in the Appendix.

LEMMA 1. *Let G be a set of FDs, and let $g: X \rightarrow Y$ be an FD in G. If $h: V \rightarrow W$ is in G^+ and g is used for some derivation of h from G, then $V \rightarrow X$ is in G^+.*

PROOF. We give here an intuitive argument using an informal notion of a derivation. A formal proof using the "derivation tree" model of derivations is given in the Appendix.

We introduce the notation $U \Rightarrow Z$ to mean that the FD $U \rightarrow Z$ can be derived by an application of one of Armstrong's axioms on a given set of FDs. The notation $U \Rightarrow_* Z$ means that $U \rightarrow Z$ is derivable by using several applications of the axioms. Now the lemma states that there is a derivation $V \Rightarrow_* W$ using g. That is,

there is a derivation $V \Rightarrow_* ZX \Rightarrow ZY \Rightarrow_* W$ for some (possibly empty) set of attributes Z (the step $ZX \Rightarrow ZY$ is the step that uses g). But $V \Rightarrow_* ZX$ implies $V \rightarrow ZX$, which implies $V \rightarrow X$, thereby proving the lemma. □

4.2 Semantics of Functional Dependencies

The treatment of FDs in this paper is a strictly syntactic one based on Armstrong's axioms. To use this approach, we must make the following assumption of uniqueness: For a given set of FDs G and an FD $X \rightarrow Y$, either $X \rightarrow Y$ is not in G^+ or there exists a unique FD $X \rightarrow Y$ in G^+. That is, if there are two FDs on the same set of attributes, then they are the same FD; if $f : X \rightarrow Y$ and $g : X \rightarrow Y$, then f is identical to g. Thus, the set of FDs that are accepted as input to the synthesis algorithm is assumed to satisfy not only Armstrong's axioms, but also the uniqueness assumption. (Both of these assumptions are also required for all previous syntactic approaches to 3NF (e.g. [10, 11, 12])). That this assumption is quite strong can be seen from several examples.

Let f_1: DEPT# → MGR# and f_2: MGR#,FLOOR → NUMBER_OF_EMPLOYEES. One interpretation of f_1 and f_2 is that f_1 determines the manager of each department and f_2 determines the number of employees working for a particular manager on a particular floor. By applying pseudotransitivity to f_1 and f_2 we obtain f_3: DEPT#,FLOOR → NUMBER_OF_EMPLOYEES, which determines the number of employees of the *manager* of a particular department on a particular floor. If a manager can manage more than one department, then f_3 is not the same as the syntactically identical FD g_1: DEPT#,FLOOR → NUMBER_OF_EMPLOYEES, which determines the number of employees of a particular *department* on a particular floor. To make g_1 distinct from f_3, one has to change an attribute name to make the FDs *syntactically* distinct. For example one could change f_2 and g_1 such that f_2: MGR#,FLOOR → NUMBER_OF_EMPLOYEES_OF_MANAGER and g_1: DEPT#,FLOOR → NUMBER_OF_EMPLOYEES_OF_DEPT. Now g_1 is distinct from the composition of f_1 and f_2.

As a second example, let f_4: EMP# → MGR# and f_5: MGR# → EMP#. It must be the case here that f_4 is the inverse of f_5. For if we compose f_4 and f_5, we obtain g_2: EMP# → EMP#. Since there is only one FD connecting EMP# to EMP# (by our assumption), and since by Armstrong's axioms the identity function must exist, then g_2 must be the identity map. This implies $f_4 = f_5^{-1}$. If we take the interpretation that f_4 maps an employee into his manager and f_5 maps a manager's MGR# into his corresponding EMP#, then of course $f_4 \neq f_5^{-1}$. So to take this interpretation, one must make f_4 and f_5 syntactically distinct (e.g. f_5: MGR# → EMP#_OF_MGR).

As a third example, let f_6: STOCK# → STORE# and f_7: STOCK#,STORE# → QTY. Since the composition of f_6 and f_7 is g_3: STOCK# → QTY, it must be (by our assumption) that the attribute STORE# in f_7 is not needed. But suppose f_6 maps a STOCK# into the STORE# of the store that is in charge of ordering that item and f_7 maps the STOCK# of an item and the STORE# of the store in which it is being sold into the quantity on hand. In this case g_3 does not imply that STORE# is extraneous in f_7. To prevent this syntactic inference from taking place, we must change an attribute name (e.g. f_6: STOCK# → ORDERING_STORE#).

In each of these examples, a syntactic inference was either erroneous or mislead-

ing. In each case, we solved the problem by renaming an attribute to distinguish it from another attribute. This renaming essentially moves some semantic knowledge that we have about an FD onto the syntactic level, where it can be used by the algebra of FDs.

Specifying a set of FDs that can lead to no invalid syntactic inferences is clearly a difficult problem. For no syntactic check based only on the algebra of FDs can determine whether a given set of FDs satisfies the uniqueness assumption. Yet, if we are to make use of a formal algebra of FDs, we must make the assumption that all syntactic inferences are valid. If we had an automated semantic analyzer that could judge the validity of each syntactic inference, then we could use it as a sieve to toss out invalid inferences. Unfortunately such a semantic analyzer is well beyond the state of the art. So we will add to our assumption of the validity of syntactic inferences the proviso that all syntactic inferences are (or at least can be) checked for semantic validity. If an inference is invalid, it can either result in renaming of some attributes or be simply rejected.

Third normal form is a strictly syntactic property that is governed by the algebra of FDs. In this paper we give a complete account of mapping from FDs into a 3NF schema, given that Armstrong's axioms and the uniqueness assumption are accepted. Given Armstrong's completeness proof, we believe these assumptions to be quite reasonable in modeling relational databases. We are not attacking the problem of how to judge the semantic validity of syntactic inferences. Semantic problems of this type are not well understood and seem to be more difficult than the syntactic problem of determining 3NF. Their solution remains a matter for further research.

5. A MORE SOPHISTICATED SYNTHESIS PROCEDURE

5.1 A Description of the Algorithm

The simple synthesis procedure of Section 3.3 led to problems because the rules for composing FDs were ignored. The main difficulty is that the redundant FDs that filter into the synthesized schema create extra attributes and contribute to unnormalized connections among attributes. By first taking a nonredundant covering of the given set of FDs, the normalization problems can be alleviated. In Figure 2 for example, f_2 is redundant and therefore will not appear in a nonredundant covering of the given FDs, and the 3NF violation of $\mathbf{R}_1$ is thereby avoided.

Finding a nonredundant covering is not sufficient to avoid problem FDs such as f_6 in Figure 2. This further problem can be eliminated by excising extraneous attributes from the left sides of FDs. An attribute X_i is *extraneous in an FD* $g \in G$, $g: X_1, \ldots, X_p \rightarrow Y$, if $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_p \rightarrow Y$ is in G^+. Eliminating extraneous attributes helps to avoid partial dependencies and superkeys that are not keys, as in $\mathbf{R}_4$ of Figure 2.

If two relations have keys that are functionally dependent upon each other (i.e. are equivalent), then the two relations can be merged together. This can be accomplished in the synthesis procedure by merging together two groups of FDs if their left sides are functionally equivalent. For example g_2 and g_3 in Figure 2 can be merged into a single group.

The following procedure includes the above improvements:

ALGORITHM 1

1. (Eliminate extraneous attributes.) Let F be the given set of FDs. Eliminate extraneous attributes from the left side of each FD in F, producing the set G. An attribute is extraneous if its elimination does not alter the closure of the set of FDs.
2. (Find covering.) Find a nonredundant covering H of G.
3. (Partition.) Partition H into groups such that all of the FDs in each group have identical left sides.
4. (Merge equivalent keys.) For each pair of groups, say H_1 and H_2, with left sides X and Y, respectively, merge H_1 and H_2 together if there is a bijection $X \leftrightarrow Y$ in H^+.
5. (Construct relations.) For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in in the group is a key of the relation. (Step 1 guarantees that no such set contains any extra attributes.) All keys found by this algorithm will be called *synthesized*. The set of constructed relations constitutes a schema for the given set of FDs.

In what follows, we shall refer to Algorithm 1 with step 4 excised as Algorithm 1(a).

A linear time algorithm for testing membership in the closure of a set of FDs is presented in [2]. Using this procedure, one can implement Algorithm 1 with a time bound of $O(L^2)$, where L is the length of the string encoding the given set of FDs.

5.2 Completeness of the Synthesized Schema

A schema $\mathcal{S}$ *completely characterizes* a set of FDs F if the closure of the FDs embodied in $\mathcal{S}$ equals F^+. To show that Algorithm 1 synthesizes a schema that is a complete characterization of the given FDs, consider a set of FDs F that is given as input to Algorithm 1. Let H be the set of FDs that result from eliminating extraneous attributes and redundant FDs. Clearly, H^+ still equals F^+. Let $\mathcal{S}$ be a schema synthesized from F. Since H is exactly the set of FDs embodied in $\mathcal{S}$ and $H^+ = F^+$, every FD in F can be derived from a subset of the FDs embodied in $\mathcal{S}$. Hence, $\mathcal{S}$ completely characterizes F.

We would like to be certain that the extension of every FD in the given set F can be retrieved from the extension of the synthesized schema $\mathcal{S}$ using relational algebra. We will argue that this follows from the fact that $\mathcal{S}$ completely characterizes F.

Consider some $f: X \rightarrow A \in F$. We begin by noting that extraneous attributes in X can be ignored. That is, if the extension of an FD $f': X' \rightarrow A$ where $X' \subseteq X$ can be retrieved from the extension of $\mathcal{S}$, then since f can be obtained from f' simply by augmentation, we can treat f' to be the same FD as f. Now, since $\mathcal{S}$ completely characterizes F, there is a derivation for f' based on the FDs H that are embodied in $\mathcal{S}$. In the Appendix, we show that if f' has no extraneous attributes, then it can be derived from H using only the pseudotransitivity axiom. Since an application of psuedotransitivity corresponds exactly to a join in relational algebra, the derivation for f' from H can be simulated by a sequence of joins on the extension of the relations of $\mathcal{S}$. In this way, the extension of every $f \in F$ can be retrieved from the extension of $\mathcal{S}$ using relational algebra. That is, our notion of "complete characterization" satisfies the intuition that all relationships specified in the given set of FDs are actually retrievable from the extension of the synthesized schema.

5.3 Nonfunctional Relationships

We introduced a special notation for representing nonfunctional relationships in our input FDs. We must now make sure that these FDs behave in the expected way.

If $X \rightarrow \Theta$ is in the set of FDs given to Algorithm 1, then either $X \rightarrow \Theta$ or $Y \rightarrow \Theta$, where $Y \leftrightarrow X$, appears in the schema synthesized by the algorithm. This is a consequence of the following lemma, which is proved in the Appendix.

LEMMA 2. *If $X \rightarrow \Theta$ is in a set of FDs G, then for any nonredundant covering H of G, either $X \rightarrow \Theta$ is in H or $Y \rightarrow \Theta$ is in H, where $X \rightarrow Y$ and $Y \rightarrow X$.* □

Thus the nonfunctional relationships appear in the schema in nearly the same form in which they are specified in the given set of FDs.

By the above lemma, the Θ attributes, which were invented to permit the representation of nonfunctional relationships, always appear in the synthesized schema. How are they interpreted? To see this, consider the following example. Suppose two nonfunctional relationships were specified in the given set of FDs, f_1: $AB \rightarrow \Theta_1$ and f_2: $AB \rightarrow \Theta_2$. (Note again that the uniqueness assumption of FDs does not force uniqueness of nonfunctional relationships between A and B.) Step 4 of Alg. 1 merges these two FDs into a single group, yielding a relation $\mathbf{R}(A, B, \Theta_1, \Theta_2)$. In order to distinguish whether a given pair of values for A and B satisfies f_1, f_2, or both f_1 and f_2, the Θ_1 and Θ_2 attributes must be retained. For example $(a, b, 0, 1) \in \mathbf{R}$ means a, b satisfies f_2 but not f_1. Note that if there is only one nonfunctional relationship among a set of attributes, then the Θ attribute can generally be dropped, since this problem of distinguishing among relationships disappears. For example, if only f_1 were present, then it is customary only to include (a,b) pairs that are related under f_1; a tuple $(a,b,0)$ would normally not be included in the extension. Therefore in this case, the Θ attribute would be dropped altogether.

6. THIRD NORMAL FORM SCHEMAS

6.1 Introduction

In this section we show under what conditions various synthesized relations are in 3NF. We begin by showing that Algorithm 1(a) (i.e. Algorithm 1 without step 4) always produces a 3NF schema. We then examine Algorithm 1. A property of derivations of nonprime attributes is introduced and shown to be a sufficient condition for Algorithm 1 to produce a schema in 3NF. Unfortunately there are cases of FDs that do not satisfy this property and therefore can lead to relations with transitive dependencies. One such example is presented and shown to be a counterexample to a theorem given by Delobel and Casey.

6.2 Algorithm 1(a) Schemas

To prove that every relation synthesized by Algorithm 1(a) is in 3NF, we show that a transitive dependency implies the existence of a redundant FD in the nonredundant covering. We will use Lemma 1 to show the existence of an FD that creates the contradiction. Lemma 1 will be used in this way in all succeeding 3NF proofs.

THEOREM 1. *Let* $\mathbf{R}(A_1, \ldots, A_n)$ *be a relation synthesized from the set of FDs* F *by using Algorithm* 1(*a*). *Then no nonprime attribute of* **R** *is transitively dependent upon any key of* **R**. *That is,* **R** *is in* 3*NF*.

PROOF. Suppose A_i is nonprime and is transitively dependent upon a key K of **R**. (K need not be synthesized.) That is, there is an $X \subseteq \{A_1, \ldots, A_n\}$ such that $K \rightarrow X$, $X \nrightarrow K$, and $X \rightarrow A_i$ are in F^+, and A_i is not in X.

We first observe that A_i is transitively dependent upon the synthesized key of **R**. Let Z be the key of **R** that appears on the left side of the FDs that were used in synthesizing **R**. $Z \rightarrow X$ is in F^+ since Z is a key of **R**. Furthermore $X \nrightarrow Z$, for if $X \rightarrow Z$, then $X \rightarrow Z$ and $Z \rightarrow K$ would imply $X \rightarrow K$, contradicting $X \nrightarrow K$ in the original transitive dependency. Hence $Z \rightarrow X$, $X \nrightarrow Z$, and $X \rightarrow A_i$ is also a transitive dependency.

Let H be the nonredundant covering of G computed in Algorithm 1(a). We shall now show that $Z \rightarrow A_i$, which appears in H, is redundant. To do this it is sufficient to show that $Z \rightarrow X$ and $X \rightarrow A_i$ can both be derived from $H - \{Z \rightarrow A_i\}$.

Since the only FDs used in synthesizing **R** are of the form $Z \rightarrow A_j$, it must be that $Z \rightarrow X_k$ is in H for all $X_k \in X$. Since A_i is not in X, $Z \rightarrow X_k$ is in $H - \{Z \rightarrow A_i\}$.

Suppose there is a derivation for $X \rightarrow A_i$ in H that uses $Z \rightarrow A_i$. Then by Lemma 1, we have $X \rightarrow Z$. But this violates $X \nrightarrow Z$ in the transitive dependency. So $X \rightarrow A_i$ must be derivable without the use of $Z \rightarrow A_i$.

Since $Z \rightarrow X$ and $X \rightarrow A_i$ can both be derived from $H - \{Z \rightarrow A_i\}$, it must be that $Z \rightarrow A_i$ is redundant in H, contradicting that H is nonredundant. But this in turn must mean that the transitive dependency did not exist. □

The above theorem was first presented by Wang and Wedekind [12]; however their proof was incorrect [4]. In their proof they only argued that the transitive dependency was derivable in H, not $H - \{Z \rightarrow A_i\}$. In terms of the above proof, they claimed that if $K \rightarrow X$ and $X \rightarrow A_i$ is a transitive dependency, then $K \rightarrow A_i$ is derivable by pseudotransitivity. This, they asserted, violates the fact that $K \rightarrow A_i$ is in a nonredundant covering. However, the latter is only true if one can show that both $K \rightarrow X$ and $X \rightarrow A_i$ are derivable from the closure without the use of $K \rightarrow A_i$. For example, $G = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$ is a set of FDs where $A \rightarrow B$ and $B \rightarrow C$ are in the closure but $A \rightarrow C$ is not redundant because $B \rightarrow C$ cannot be derived from G without $A \rightarrow C$. In any case, their theorem was correct as stated, and the above argument corrects their proof, using Lemma 1 and the important fact that $X \nrightarrow K$ in the transitive dependency.

It is interesting to note that they did not eliminate superkeys in their version of Algorithm 1(a). This was not an error, since they explicitly *assumed* that extraneous attributes did not exist on the left sides of FDs. However, one need not make this general assumption, since some extraneous attributes can be eliminated algorithmically. In fact, to be entirely consistent with the algebra of FDs, one *must* eliminate such extraneous attributes. Of course not all such extraneous attributes can be eliminated in this way; many semantic errors must remain the user's responsibility for reasons discussed in Section 4.2.

One might expect the proof of Theorem 1 to generalize to schemas synthesized by Algorithm 1. Unfortunately this is not the case. A schema that is not in 3NF can be synthesized by Algorithm 1, as shown in Figure 3(b). In the next section we

will add a further precondition that is sufficient to guarantee 3NF for schemas produced by Algorithm 1.

6.3 A Sufficient Condition for 3NF

To show that no nonprime attribute is transitively dependent upon any key of **R**, we will use the following property:

An attribute A is said to satisfy *property P* in relation **R** if the following proposition holds: Let H be the nonredundant covering produced from step 2 of Algorithm 1. If $K \rightarrow A$ is in H and $K \rightarrow A$ is used in synthesizing **R**, then for any prime attribute B of **R**, the FD $K \rightarrow B$ can be derived without the use of $K \rightarrow A$ (i.e. can be derived in $H - \{K \rightarrow A\}$).

Property P is strictly a syntactic property of derivations of FDs and to our knowledge has no semantic interpretation in terms of real world relationships. It is the weakest property we know of that is sufficient to guarantee that Algorithm 1 produces 3NF relations. The proof that property P is sufficient to guarantee 3NF follows the same lines as the proof of Theorem 1.

THEOREM 2. *Let* $\mathbf{R}(A_1, \ldots, A_n)$ *be one of the relations synthesized by using Algorithm* 1 *from a set of FDs F. If all nonprime attributes of* **R** *satisfy property P, then* **R** *is in 3NF.*

PROOF. Let A_i be a nonprime attribute of **R** that is transitively dependent upon some key Y of **R**. That is, there is a $Z \subset \{A_1, \ldots, A_n\}$ such that $Y \rightarrow Z$, $Z \nrightarrow Y$, and $Z \rightarrow A_i$ with A_i not in Z.

Let H be the nonredundant covering computed in Algorithm 1. Let K be a key such that $h: K \rightarrow A_i$ is in H. That is, h is an FD that brought A_i into **R** by Algorithm 1.

Since K is a key, $K \rightarrow Z$. Furthermore $Z \nrightarrow K$, for if $Z \rightarrow K$, then $Z \rightarrow K$ and $K \rightarrow Y$ implies $Z \rightarrow Y$, a contradiction. So we have a new transitive dependency: $K \rightarrow Z$, $Z \nrightarrow K$, and $Z \rightarrow A_i$. We want to show that $K \rightarrow Z$ and $Z \rightarrow A_i$ are in $(H - \{K \rightarrow A_i\})^+$ to establish the contradiction that H is redundant.

FDs	Schema synthesized by Algorithm 1
$f_1: X_1,X_2 \rightarrow A$	$\mathbf{R}_1(\underline{X_1,X_2},\underline{C},A)$ (from f_1 and f_2)
$f_2: C \rightarrow X_1,X_2$	
$f_3: A,X_1 \rightarrow B$	$\mathbf{R}_2(\underline{A,X_1},B)$
$f_4: B,X_2 \rightarrow C$	$\mathbf{R}_3(\underline{B,X_2},C)$

A does not satisfy property P, yet $\mathbf{R}_1$ is in 3NF.

(a)

FDs	Schema synthesized by Algorithm 1
$g_1: X_1,X_2 \rightarrow A,D$	$\mathbf{S}_1(\underline{X_1,X_2,C,D},A)$ (from g_1 and g_2)
$g_2: C,D \rightarrow X_1,X_2$	
$g_3: A,X_1 \rightarrow B$	$\mathbf{S}_2(\underline{A,X_1},B)$
$g_4: B,X_2 \rightarrow C$	$\mathbf{S}_3(\underline{B,X_2},C)$
$g_5: C \rightarrow A$	$\mathbf{S}_4(\underline{C},A)$

$\mathbf{S}_1$ is not in 2NF since A is partially dependent upon the key CD.

(b)

Fig. 3. An example of a violation of property P

Let $Z = \{B_1, \ldots, B_m\}$. We distinguish two cases. If B_j is prime, then property P guarantees that $K \rightarrow B_j$ is derivable without the use of $K \rightarrow A_i$. If B_j is not prime, then there is an FD $K' \rightarrow B_j$ that brought B_j into **R**. Since $K \rightarrow K'$ is derivable (by property P) from $H - \{K \rightarrow A_i\}$ and $K' \rightarrow B_j$ is in $H - \{K \rightarrow A_i\}$, we obtain that $K \rightarrow B_j$ is derivable without the use of $K \rightarrow A_i$. Hence $K \rightarrow Z$ is in $(H - \{K \rightarrow A_i\})^+$.

Now assume $Z \rightarrow A_i$ uses $K \rightarrow A_i$ in its derivation. Then by Lemma 1, $Z \rightarrow K$, contradicting the transitive dependency. Hence $Z \rightarrow A_i$ is in $(H - \{K \rightarrow A_i\})^+$.

The FDs $K \rightarrow Z$ and $Z \rightarrow A_i$ are in $(H - \{K \rightarrow A_i\})^+$, establishing that H is redundant, a contradiction. Hence the transitive dependency could not have existed. □

The need for property P arises from the merging of equivalent keys in step 4 of Algorithm 1. Suppose K_1 and K_2 are merged in step 4 because $K_1 \leftrightarrow K_2$, and suppose $K_1 \rightarrow Z$, $Z \nrightarrow K_1$, $Z \rightarrow A$ is a transitive dependency in the synthesized relation. This transitive dependency would not exist if K_1 and K_2 were the keys of two separate relations, as would be the case if Algorithm 1(a) were used. One reason why the transitive dependency can arise is that there is a $Z_i \in Z$, with $K_2 \rightarrow Z_i$ (and $K_1 \rightarrow A$) in the nonredundant covering, but $K_1 \rightarrow Z_i$ (and $K_2 \rightarrow A$) not in the covering. Thus a relation must contain both K_1 and K_2 to manifest the transitive dependency. The FD $K_1 \rightarrow Z$ in the transitive dependency is a composition of $K_1 \rightarrow K_2 \rightarrow Z$. If A does not have property P, then $K_1 \rightarrow A$ may be necessary to obtain $K_1 \rightarrow K_2$, in which case $K_1 \rightarrow A$ need not be redundant (as it would be in Algorithm 1(a)). However, if A does have property P, then $K_1 \rightarrow K_2$ does not need $K_1 \rightarrow A$, so $K_1 \rightarrow A$ *is* redundant, and we have the theorem.

Consider the set of FDs in Fig. 3(a) which produces the relation $\mathbf{R}_1(\underline{X_1, X_2}, \underline{C}, A)$ via Algorithm 1. (The reader can check that $X_1, X_2 \rightarrow C$ is in the closure of the given FDs.) The attribute A is nonprime in $\mathbf{R}_1$ and does not satisfy property P since the only way to derive $X_1, X_2 \rightarrow C$ is by using $X_1, X_2 \rightarrow A$. However, despite the violation of property P, relation $\mathbf{R}_1$ is in 3NF. Hence property P is not a necessary condition for 3NF.

Figure 3(b) presents an example of FDs that exhibit the same violation of property P as Figure 3(a) but induce a partial (and, hence, a transitive) dependency. In terms of the above discussion regarding transitive dependencies, we have $X_1, X_2 \rightarrow C$, $C \nrightarrow X_1, X_2$, and $C \rightarrow A$, but $X_1, X_2 \rightarrow A$ is not redundant since $X_1, X_2 \rightarrow C$ needs $X_1, X_2 \rightarrow A$ in its derivation.

Property P affects one other published procedure that synthesizes relations from FDs. Delobel and Casey [10] claim that their decomposition procedure, which is in some sense comparable to our Algorithm 1, produces 3NF relations. Their claim, however, is incorrect in that the example in Figure 3(b) falsifies their theorem [6].

6.4 Putting Relations into 3NF

A violation of property P may induce a 3NF violation. Once a particular violation of 3NF is found, in order to put the relation into 3NF the offending dependency must be removed. Conveniently enough, if a nonprime attribute is transitively dependent upon a key of a relation, then the attribute can simply be removed from the relation, and the resulting schema will still embody the same FDs.

THEOREM 3. *Let* $\mathbf{R}_k(A_1, \ldots, A_n)$ *be a relation in a schema* $\mathbb{S} = \{\mathbf{R}_1, \ldots, \mathbf{R}_m\}$ *that was synthesized by using Algorithm* 1. *Let* H *be the set of FDs embodied in* $\mathbb{S}$. *Let* A_i *be an attribute of* $\mathbf{R}_j$ *that appears in none of the* synthesized *keys of* $\mathbf{R}_j$, *and let* A_i *be transitively dependent upon a key of* $\mathbf{R}_k$. *Suppose* A_i *is removed from* $\mathbf{R}_k$, *resulting in a new relation* $\mathbf{R}_{k'}$ *and hence a new schema* $\mathbb{S}' = \{\mathbf{R}_1, \ldots, \mathbf{R}_{k'}, \ldots, \mathbf{R}_m\}$. *Then the closure of the set of FDs embodied in* $\mathbb{S}'$ *equals* H^+.

PROOF. Suppose A_i is removed from $\mathbf{R}_k$. Since A_i does not appear in any of the keys synthesized by Algorithm 1, its removal can only affect embodied FDs of the form $f: X \rightarrow A_i$, where X is a synthesized key of $\mathbf{R}_k$. Let H' be the set of FDs embodied in $\mathbb{S}'$. If we show that all such f are in $(H')^+$, then $H^+ = (H')^+$.

By the same argument used in the proofs of Theorems 1 and 2, if A_i is transitively dependent upon any key of $\mathbf{R}_k$, then it is transitively dependent upon all of them. For each f of the above form, X is a key. Therefore, for each such X, $X \rightarrow V$, $V \nrightarrow X$, and $V \rightarrow A_i$, A_i not in V, are a transitive dependency.

Since A_i is not in V, for each X the FD $X \rightarrow V$ is still embodied in $\mathbf{R}_k$ even after A_i is removed. Hence each $X \rightarrow V$ is in H^+.

To show $V \rightarrow A_i \in (H')^+$, we must show that $V \rightarrow A_i$ cannot use any of the FDs $X \rightarrow A_i$ in its derivation. But this follows directly since if $X \rightarrow A_i$ is used to derive $V \rightarrow A_i$, then by Lemma 1 $V \rightarrow X$, contradicting $V \nrightarrow X$ in one of the transitive dependencies.

Since $X \rightarrow V$ and $V \rightarrow A_i$ are in $(H')^+$, $f: X \rightarrow A_i$ is in $(H')^+$ for all such f. Hence $(H')^+ = H^+$. □

Theorem 3 provides us with a simple means of removing an unwanted transitive dependency: Namely, excise the offending attribute from the relation. The theorem guarantees that the resulting schema still embodies the given set of FDs.

That a transitive dependency can be removed so easily is rather surprising since the FDs that form the schema are nonredundant. It would seem that excising an attribute should result in the loss of an FD. However, in synthesizing the schema from a nonredundant covering of FDs, we have implicitly added new FDs to the covering in step 4 of Algorithm 1, and these FDs are explicitly embodied in the schema. If X and Y are the left sides of two distinct FDs in the nonredundant covering H with $X \rightarrow Y$ and $Y \rightarrow X$ in H^+, then X and Y are merged and put into a single relation. This adds $X \rightarrow Y$ and $Y \rightarrow X$ as two new FDs that are explicitly embodied in the schema, even though they may not have appeared in the covering. For example, in Figure 3(b) the FDs $X_1, X_2 \rightarrow C, D$ and $C, D \rightarrow X_1, X_2$ are explicitly embodied in $\mathbf{S}_1$, even though the former FD is not in the covering. It is the addition of the extra FDs that allows us to excise a transitive dependency without affecting the closure of the embodied FDs.

Looking at Theorem 3 in a different light, we can now modify Algorithm 1 to synthesize schemas that are guaranteed to be in 3NF. Let H be the nonredundant covering resulting from step 2 of Algorithm 1. Let J be the set of all FDs $X \rightarrow Y$ such that X and Y are equivalent keys discovered in step 4 of Algorithm 1. Let $h: Z \rightarrow A_i$, $h \in H$, be embodied in $\mathbf{R}_k$ in such a way that A_i appears in no synthesized key of $\mathbf{R}_k$ and A_i is transitively dependent upon a key of $\mathbf{R}_k$. Then Theorem 4 says that h is redundant; that is, $h \in (H + J - \{h\})^+$. So, if we eliminate every FD $h \in H$ whose right side is not in any synthesized key and for which

$h \in (H + J - \{h\})^+$, then we will have eliminated all transitive dependencies. If there were a nonprime A_i that was transitively dependent on a key of $\mathbf{R}_k$, then Theorem 4 would guarantee that our extra redundancy check would have eliminated it. This leads us to our main result, Algorithm 2, which synthesizes a provably 3NF schema.

ALGORITHM 2

1. (Eliminate extraneous attributes.) Let F be the given set of FDs. Eliminate extraneous attributes from the left side of each FD in F, producing the set G. An attribute is extraneous if its elimination does not alter the closure of the set of FDs.
2. (Find covering.) Find a nonredundant covering H of G.
3. (Partition.) Partition H into groups such that all of the FDs in each group have identical left sides.
4. (Merge equivalent keys.) Let $J = \varnothing$. For each pair of groups, say H_i and H_j, with left sides X and Y, respectively, merge H_1 and H_2 together if there is a bijection $X \leftrightarrow Y$ in H^+. For each such bijection, add $X \rightarrow Y$ and $Y \rightarrow X$ to J. For each $A \in Y$, if $X \rightarrow A$ is in H, then delete it from H. Do the same for each $Y \rightarrow B$ in H with $B \in X$.
5. (Eliminate transitive dependencies.) Find an $H' \subseteq H$ such that $(H' + J)^+ = (H + J)^+$ and no proper subset of H' has this property. Add each FD of J into its corresponding group of H'.
6. (Construct relations.) For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation. (Step 1 guarantees that no such set contains any extra attributes.) All keys found by this algorithm will be called *synthesized*. The set of constructed relations constitutes a schema for the given set of FDs.

Steps 1–4 are effectively implemented as in Algorithm 1. Step 5 can be effectively implemented by using the membership algorithm of [5]. Algorithm 2 can then be implemented in the same $O(L^2)$ time bound as Algorithm 1.

7. PROOF OF MINIMALITY

The purpose of this section is to examine the number of relations synthesized by Algorithm 2 (or 1) for a given set of FDs, compared with any other relational schema that embodies those FDs. We will show that all nonredundant coverings generate the same number of relations by showing that the number of equivalence classes of synthesized keys is the same across all nonredundant coverings of a given set of FDs. This will then imply that the schemas synthesized by Algorithm 2 are minimal in the number of relations synthesized.

LEMMA 3. *Let G_1 and G_2 be two nonredundant sets of FDs with $G_1^+ = G_2^+$. If $g: X \rightarrow A$ is in G_1, then there exists an $h: Y \rightarrow B$ with $h \in G_2$ and with $Y \rightarrow X$ and $X \rightarrow Y$ in G_1^+.*

PROOF. If $g \in G_1$ and $G_1^+ = G_2^+$, then $g \in G_2^+$. Hence there is a derivation for g in G_2. Each FD h used to derive g is in G_2, so each such h is in G_1^+.

If no h requires g in its derivation in G_1, then we can construct a derivation for g in G_1. The derivation is constructed by mimicking the derivation of g in G_2, replacing each h in this derivation with an h derived in G_1. Since this derivation does not use g, g must be redundant in G_1, a contradiction. So at least one h must require g in its derivation in G_1.

Suppose that $h: Y \rightarrow B$ requires $g: X \rightarrow A$ in its derivation in G_1. Then by Lemma

$$G = \{g_1: C \rightarrow D, \quad g_2: D \rightarrow C, \quad g_3: CE \rightarrow F\} \qquad H = \{h_1: C \rightarrow D, \quad h_2: D \rightarrow C, \quad h_3: DE \rightarrow F\}$$

G and H are nonredundant and $G^+ = H^+$. However, g_3 and h_3 generate different relations. This is an example of Lemma 3 where $X \neq Y$.

Fig. 4. Two equivalent coverings with different keys

1, $X \rightarrow Y$. Also, since h appeared in a derivation for g in G_2, by Lemma 1 $Y \rightarrow X$. Hence $h: Y \rightarrow B$ is in G_2 with $X \rightarrow Y$ and $Y \rightarrow X$, completing the proof. □

Lemma 2 cannot be strengthened so that $X = Y$. That is, one can have two nonredundant coverings with equivalent closures such that the two coverings have different left sides representing a key equivalence class. For example, in Figure 4 g_3 and h_3 have functionally equivalent left sides since $CE \leftrightarrow DE$, yet $CE \neq DE$.

Using Lemma 3 and recognizing that Algorithm 2 synthesizes a relation from each maximal group of FDs that have functionally equivalent left sides, we can now see that all nonredundant coverings of a given set of FDs produce the same number of relations by Algorithm 2.

THEOREM 4. *Let F be a set of FDs. Any two nonredundant coverings of F will produce the same number of relations via Algorithm 2.*

PROOF. Let G_1 and G_2 be two nonredundant coverings of F. By Lemma 3, if an FD $g: X \rightarrow A$ is in G_1, then there is an $h: Y \rightarrow B$ in G_2 with $X \rightarrow Y$ and $Y \rightarrow X$. Thus for any group of FDs in G_1 with functionally equivalent left sides, there must be exactly one such group in G_2, namely, the one that has the same functionally equivalent left sides. Since each such group generates one relation, G_1 and G_2 must produce the same number of relations. □

Theorem 4 states that all choices of nonredundant coverings are equally good in terms of number of relations synthesized. This is somewhat surprising in that it contradicts the intuition that a minimal-sized nonredundant covering would perhaps produce fewer relations than other larger nonredundant coverings.

The theorem also shows that on the logical level there is not very much choice as to how to pick relations that cover the given set of FDs. Some of the decomposition approaches (e.g. [10, 11, 12]) claim to allow the system to choose among a class of possible schemas, directing the choice by efficiency considerations. Since all coverings have the same set of equivalence classes of keys, the class of possible schemas is really quite small. Hence if one is guided on the logical level by normalization considerations rather than by efficiency considerations, one arrives at a set of nearly identical possible schemas.

From Theorem 4, we can see that the number of relations generated by Algorithm 2 is minimal among all those that embody the same given set of FDs. This gives a complete characterization of the optimal 3NF schemas discussed in [8].

COROLLARY 1. *Let $\mathbb{S}$ be a schema synthesized from a set of FDs F by using Algorithm 2. Let $\mathbb{S}'$ be any schema embodying a set of FDs G that covers F. Then* $|\mathbb{S}'| \geq |\mathbb{S}|$.

PROOF. Let $H \subset G$ be a nonredundant covering of F. Certainly H will generate, via Algorithm 2, no more relations than are in $\mathbb{S}'$. Furthermore, by Theorem 4, Algorithm 2 will generate the same number of relations from G as from F. Hence $|\mathbb{S}'| \geq |\mathbb{S}|$. □

8. CONCLUSION

The purpose of this paper was to develop an algorithm for synthesizing a 3NF schema from a given set of FDs and to examine some properties of such schemas. The main results were:

(1) Certain simple algorithms for synthesizing schemas either produce too many relations or violate 3NF.

(2) An algorithm that synthesizes *provably* 3NF schemas was presented. The essential aspect of this algorithm is that it eliminates as much redundancy as possible from the given set of FDs.

(3) All nonredundant coverings produce the same number of relations when this latter method is used. Hence synthesized schemas contain a minimal number of relations.

This is the first successful attempt, to our knowledge, to implement Codd's normalization procedure [8] both provably and effectively. (Errors in two earlier similar attempts were isolated.) Furthermore, by Corollary 1, the synthesized relations satisfy Codd's optimality criterion—no other schema covering the same FDs has fewer relations.

APPENDIX

Let X be a set of attributes, let G be a set of FDs over X, and let $g: D_1, \ldots, D_p \rightarrow E$ be an FD over X. If $g \in G^+$, then there is a sequence of applications of axioms A1, A2, and A3 on G that yields g. In this section we shall develop a graph model, called *derivation trees*, for such a sequence of applications of the axioms.

Derivations trees (abbreviated DTs) are formally defined as follows:

(1) If C is an attribute, then the labeled node C is a DT.

(2) If $\mathfrak{I}$ is a DT with C as a leaf node and $f: B_1, \ldots, B_m \rightarrow C$ is an FD, then the tree constructed from $\mathfrak{I}$ by adding each of $B_1, \ldots, B_m$ as children of the leaf node C is also a DT.

The derivation tree is therefore a simple model for the successive composition of FDs by pseudotransitivity (this is formalized below). A sample derivation tree construction is given in Figure 5.

Since a DT is characterized mainly by its leaf set, we will abbreviate the expression "a DT whose leaf set is contained in $\{X_1, \ldots, X_n\}$" by "an $\{X_1, \ldots, X_n\}$-DT."

The main property of DTs is described in Lemma 4.

LEMMA 4. *Let $\mathfrak{I}$ be a derivation tree constructed using a given set of FDs G. Let X be a nonempty subset of the nodes of $\mathfrak{I}$ and let Y be the set of all attributes that appear as leaves of $\mathfrak{I}$. Then $Y \rightarrow X$ is in G^+.*

PROOF. Consider first the case that X is simply the root node. This sublemma can be proved by induction on the number of FDs that are added to the DT (i.e. applications of (2) above). This follows directly since each such addition preserves the desired property that the root is functionally dependent upon the set of leaves by virtue of the pseudotransitivity rule.

Now suppose $X_i \in X$ is any internal node of $\mathfrak{I}$. Since X_i roots a Y-DT, by the

Given: G = {g1: AB->C; g2:C->D; g2:DE->F; f4:A->E}

Show: f: AB -> F ∈ G+

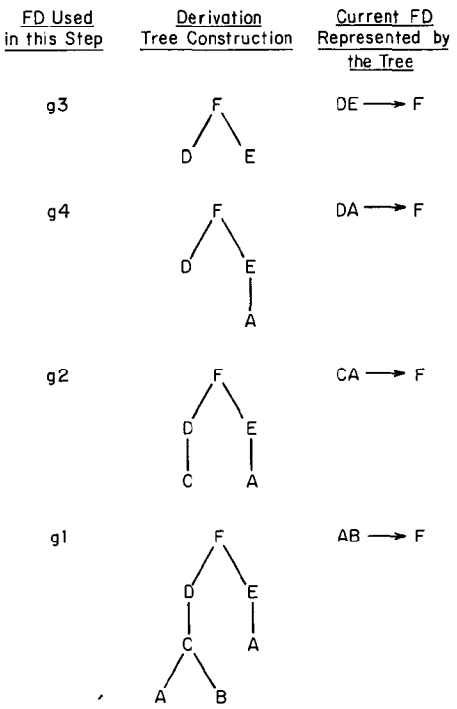

Fig. 5. A sample derivation

above sublemma we have $Y \rightarrow X_i$. By a property proved by Armstrong [1], if $Y \rightarrow X_i$ for all $X_i \in X$, then $Y \rightarrow X$, completing the proof. □

To make the DT model complete with respect to Armstrong's axioms we have to consider axioms A_1 (reflexivity) and A_2 (augmentation) as well. In a DT, reflexivity corresponds to taking a leaf node, making a copy of it, and connecting the copy as a child of the original leaf. Clearly this rule can add no new nodes to the leaf set of a DT, and hence it is basically a null operation. Except for FDs of the form $X \rightarrow X$, any FD that can be derived with reflexivity and pseudotransitivity can also be derived without reflexivity simply by eliminating all of the null replacements. The FD $X \rightarrow X$ is handled by part (1) of the DT definition.

Augmentation corresponds to the addition of extra leaf nodes connected to an internal node of the DT. All of the children of any node that was added by augmentation could have themselves been added by augmentation. Consider a DT in which augmentation was used to produce what is now a nonleaf node E of the tree. One can eliminate E from the tree by replacing it with all of its descendants that are leaves. Doing this to all internal nodes that were produced by augmentation yields a DT in which all applications of augmentation produce leaves. It follows that one application of augmentation at the very last step of a derivation is all that is needed to derive any derivable FD. (Notice that if the FD being represented

by the DT has no extraneous attributes, then no augmentation is required.) This leads us to the following theorem for the completeness of DTs.

THEOREM 5. *For a given FD $g: X \rightarrow C$ and a set of FDs G, $g \in G^+$ if and only if there is an X-DT, $\mathfrak{I}$, rooted at C.*

PROOF. $\mathfrak{I}$ represents an FD $X' \rightarrow C$ in G^+ where $X' \subseteq X$. Hence by Lemma 1 and augmentation, $g \in G^+$. To prove the converse, we know that if $g \in G^+$, then there is a sequence of (say) N applications of Armstrong's axioms yielding g from G. From the above discussion, we can assume there are no applications of reflexivity in the sequence, since they are all null steps, and that applications of augmentation are all postponed to the last application. Thus the first $N - 1$ applications are all pseudotransitivity and can be simulated by an X-DT rooted at C. □

Now, by using Theorem 5 and Lemma 4, Lemma 1 follows directly.

LEMMA 1. *Let G be a set of FDs, and let $g : X \rightarrow Y$ be an FD in G. If $h : V \rightarrow W$ is in G^+ and g is used for some derivation of h in G^+, then $V \rightarrow X$ is in G^+.*

PROOF. If $h \in G^+$, then there is a V-DT, $\mathfrak{I}$, using G. Since g is used in $\mathfrak{I}$, every attribute of X is a node of $\mathfrak{I}$. Hence we can apply Lemma 4 to obtain $V \rightarrow X$. □

The proof of Lemma 2 uses Lemma 1 and the fact that each Θ attribute appears in only one FD.

LEMMA 2. *If $X \rightarrow \Theta$ is in a set of FDs G, then for any nonredundant covering H of G, either $X \rightarrow \Theta$ is in H or $Y \rightarrow \Theta$ is in H, where $Y \rightarrow X$ and $X \rightarrow Y$.*

PROOF. If $X \rightarrow \Theta$ is in H, then we are done, so assume not. Since H covers G, there must be a derivation for $X \rightarrow \Theta$ in H. Let $Y \rightarrow \Theta$ be the root FD of a derivation tree for $X \rightarrow \Theta$ in H. By Lemma 1, $X \rightarrow Y$. To show $Y \rightarrow X$, we examine G^+. In G, $X \rightarrow \Theta$ is the only FD containing Θ. Thus any derivation for any FD in G^+ with Θ on the right side must use $X \rightarrow \Theta$ as the root FD. In particular $X \rightarrow \Theta$ is the root FD of any derivation for $Y \rightarrow \Theta$ in G^+. Hence by Lemma 1, $Y \rightarrow X$. □

ACKNOWLEDGMENTS

I am greatly indebted to C. Beeri for his help in rethinking and extending the results of [3] for this paper. In particular the definitions of schema and embodiment and step 1 of Algorithm 1 are due to him.

I would like to thank J.M. Smith for discovering an error in my original treatment of nonfunctional relationships and for pointing out many opaque sections of an earlier draft. I would also like to thank H.A. Schmid, K. Sevcik, J.R. Swenson, and D. Tsichritzis for many helpful discussions about approaches to and the significance of many of the results of this research. Finally I would like to thank the referees for numerous helpful suggestions in improving the comprehensibility of the final draft.

REFERENCES

1. ARMSTRONG, W.W. Dependency structures of data base relationships. Information Processing 74, North-Holland Pub. Co., Amsterdam, 1974, pp. 580–583.
2. BERNSTEIN, P.A. Normalization and functional dependencies in the relational data base model. Tech. Rep. CSRG-60, Ph.D. Diss., Comptr. Systems Res. Group, Dep. Comptr. Sci., U. of Toronto, Toronto, Canada, Nov. 1975.
3. BERNSTEIN, P.A., SWENSON, J.R., AND TSICHRITZIS, D.C. A unified approach to func-

tional dependencies and relations. Proc. ACM 1975 SIGMOD Conf., San Jose, Calif., pp. 237–245.
4. BERNSTEIN, P.A. A comment on "Segment synthesis in logical data base design." *IBM J. Res. Develop. 20*, 4 (July 1976), 412.
5. BERNSTEIN, P.A., AND BEERI, C. An algorithmic approach to normalization of relational data base schemas. Tech. Rep. CSRG-73, Comptr. Systems Res. Group, Dep. Comptr. Sci., U. of Toronto, Toronto, Canada, Sept. 1976.
6. CASEY, R.G., DELOBEL, C., AND BERNSTEIN, P.A. A correction to "Decomposition of a data base and the theory of Boolean switching functions." (to appear in *IBM J. Res. Develop.*).
7. CODD, E.F. A relational model of data for large shared data banks. *Comm. ACM 13*, 6 (June 1970), 377–387.
8. CODD, E.F. Further normalization of the data base relational model. In *Data Base Systems*, Courant Inst. Comptr. Sci. Symp. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, 1972, pp. 33–64.
9. DATE, C.J. *An Introduction to Database Systems*. Addison-Wesley, Reading, Mass., 1975.
10. DELOBEL, C., AND CASEY, R.G. Decomposition of a data base and the theory of Boolean switching functions. *IBM J. Res. Develop. 17*, 5 (Sept. 1972), 374–386.
11. RISSANEN, J., AND DELOBEL, C. Decomposition of files, a basis for data storage and retrieval. Res. Rep. RJ 1220, IBM Res. Lab., San Jose, Calif., May 1973.
12. WANG, C.P., AND WEDEKIND, H.H. Segment synthesis in logical data base design. *IBM J. Res. Develop. 19*, 1 (Jan. 1975), 71–77.

Received January 1976; revised June 1976