A SOPHISTICATE'S INTRODUCTION TO DATABASE NORMALIZATION THEORY[†]

Catriel Beeri
Computer Science Department
The Hebrew University
Jerusalem, ISRAEL

Philip A. Bernstein
Aiken Computation Lab.
Harvard University
Cambridge, MA 02138

Nathan Goodman
Computer Corp. of America
575 Technology Square
Cambridge, MA 02139

Abstract

Formal database semantics has concentrated on dependency constraints, such as functional and multivalued dependencies, and on normal forms for relations. Unfortunately, much of this work has been inaccessible to researchers outside this field, due to the unfamiliar formalism in which the work is couched. In addition, the lack of a single set of definitions has confused the relationships among certain results. This paper is intended to serve the two-fold purpose of introducing the main issues and theorems of formal database semantics to the uninitiated, and to clarify the terminology of the field.

1. INTRODUCTION

1.1 Database Semantics

A database is a collection of information about some enterprise in the world. The role of *database semantics* is to ensure that stored information accurately represents the enterprise. Database semantics studies the creation, maintenance, and interpretation of databases as models of external activities. A wide variety of database semantic tools exist, ranging from *data type* constraints, to *integrity constraints*, to semantic modelling structures used in Artificial Intelligence [26,36,39].

This paper is concerned with a specific type of database semantic tool, namely *data dependencies*-- both functional and multi-valued dependencies. This paper surveys the major results in this area. Our aim is to provide a unified framework for understanding these results.

1.2 Database Models

Most work on data dependencies uses the *relational* data model, with which we assume reader familiarity at the level of [16]. Briefly, a relational database consists of a set of *relations* defined on certain *attributes*. R(X) is our notation for a relation named R defined on a set of attributes X.[1] The relation R(X) is a set of m-*tuples*, where $m=|X|$. A relation can be visualized as a *table* whose columns are labelled with attributes and whose rows depict tuples. Fig. 1 illustrates a relation in this way. The data manipulation operators used in this paper are *projection* and *natural join*. The projection of relation R(X) on attributes T is denoted R[T]. If V=X-T, $R[T] = \{\langle t\rangle | \langle t,V\rangle \in R(X)\}$, and is defined iff $T \subseteq X$. (If we visualize R as a table, R[T] is those columns of R labelled with elements of T.) The natural join of relations R and S is denoted R*S. Given R(X,Y) and S(Y,Z), where X,Y,Z are disjoint sets, $R*S = \{\langle x,y,z\rangle | \langle x,y\rangle \in R \text{ and } \langle y,z\rangle \in S\}$.

Functional and multivalued dependencies are predicates on relations. Intuitively, a *functional dependency* (abbr. FD) $f:X \to Y$ holds in R(X,Y,Z) iff each value of X in R is associated with exactly one value of Y (see Fig. 1). The truth-value of f can of course vary over time, since the contents of R can vary over time. A multivalued dependency (abbr. MVD) $g:X \to\to Y$ holds in R iff each X-value in R is associated with a set of Y-values in a way that does not depend on Z-values (see Fig. 1). FDs and MVDs are defined formally in the next section.

FIGURE 1. A relation with functional and multi-valued dependencies

Relation: RENTAL-UNITS
Attributes: LANDLORD,ADDRESS,APT#,RENT,OCCUPANT,PETS
Functional dependencies:
ADDRESS,APT# → RENT--Each unit has one rental
OCCUPANT → ADDRESS,APT#--Every occupant lives in one unit
Multivalued dependencies:
LANDLORD →→ ADDRESS--Each landlord can own many buildings
OCCUPANT →→ PETS --Each occupant may have several pets
Tuples:

LANDLORD,	ADDRESS,	APT#,	RENT,	OCCUPANT,	PETS
Wizard,	Oz,	#3,	$ 50,	Tinman,	Oilcan
Wizard,	Oz,	#1,	$ 50,	Witch,	Bat
Wizard,	Oz,	#1,	$ 50,	Witch,	Snake
Wizard,	Oz,	#2,	$ 75,	Lion,	Mouse
Codd,	3 NF St,	#1,	$500,	Beeri,	Fish
Codd,	3 NF St,	#1,	$500,	Bernstein,	Dog
Codd,	3 NF St,	#1,	$500,	Bernstein,	Rhino
Codd,	3 NF St,	#2,	$600,	Goodman,	Cat

[†]This work was supported in part by the National Science Foundation under Grant MCS-77-05314.

[1]More generally, the notation R(X,Y,Z,...) denotes relation R defined on $X \cup Y \cup Z \cup \ldots$.

1.3 Description vs. Content

The interplay between database *description* and database *content* is a major theme in database semantics. A database description is called a *schema*, and contains descriptions for each relation in the database. The description of a single relation is called a *relation scheme* and consists of the relation name, its attributes and a set of data dependencies. $\underline{R}=\langle T,\Gamma\rangle$ denotes a relation scheme $\underline{R}$ with attributes T and dependencies Γ (see Fig. 2). We sometimes use the notation $\underline{R}(T)$ when Γ is either unknown or irrelevant.

FIGURE 2. Formal notation for a relation scheme based on Figure 1.

RENTAL-UNITS =
  <{LANDLORD,ADDRESS,APT#,RENT,OCCUPANT,PETS},
   {ADDRESS,APT# → RENT;
    OCCUPANT → ADDRESS,APT#;
    LANDLORD →→ ADDRESS;
    OCCUPANT →→ PETS}>

The contents of a relation is called the *state* or *extension* of the corresponding scheme, and is a set of tuples as stated above. $R(T)$ denotes an extension of $\underline{R}=\langle T,\Gamma\rangle$. If $R(T)$ satisfies all dependencies in Γ, it is called an *instance* of $\underline{R}$ (notationally, R denotes an instance of $\underline{R}$). A *relational database* for a schema is a collection of instances, one for each relation scheme in the schema.

In summary, *schema* and *scheme* are syntactic objects; *database* and *relation* refer to database content. The distinction between schema-related and content-related concepts is often subtle yet important, and we keep it sharp in this paper.

1.4 The Universal Relation Assumption

Most work on data dependencies assumes that all relations in a database are projections of a single relation. Formally, suppose $R_1(T_1)$, $R_2(T_2),\ldots,R_n(T_n)$ is a database of interest, and let $T = \cup_{1\le i\le n} T_i$. It is assumed that a *universal relation* $U(T)$ exists, such that $R_i = U[T_i]$ for $1\le i\le n$.

This "universal relation assumption" is a controversial issue in the field. On the one hand, it has formal advantages: it permits us to specify relations solely in terms of their attributes; also it supports the FD and MVD *uniqueness* rule which states that syntactically identical dependencies are semantically equivalent. On the other hand, many practical applications do not naturally conform to the assumption; to force these applications into the universal relation mold places an added burden on the database administrator, and can obscure desired relationships in the database. The reader should note that all results in this paper make the universal relation assumption, and in some cases they do not extend to alternative frameworks.

1.5 Topics

Formal work in database semantics falls roughly into the areas of *schema design* and *data manipulation*. We limit our attention to the first area, though some of the work we cover has application in the second area also. The problem of schema design is: Given an initial schema, find an *equivalent* one that is *better* in some respect. As we will see, different definitions of "equivalent" and "better" lead to startlingly different results.

The paper is organized as follows. Section 2 formally defines data dependencies and reviews their basic properties. Section 3 states the schema design problem more precisely. Then Sections 4, 5 and 6 examine several definitions of schema "equivalence" and several criteria for one schema to be "better" than another. Section 7 ties these ideas together by looking at specific schema design methods. We conclude with an historical look at our field and predications for its future.

2. DATA DEPENDENCIES

2.1 Definition and Basic Properties

An FD is a statement of the form $F{:}X\to Y$, where X and Y are sets of attributes. f is *defined* for a relation $R(T)$ or a relation scheme $\underline{R}(T)$ if X and Y are subsets of T. If f is defined for R, then f is a predicate on R's state; f is *valid* in R iff every two tuples of R that have the same X-value also have the same Y-value. From the definition we see that f's validity depends only on the values assigned to X and Y. We say that FDs enjoy the *projectivity* and *inverse projectivity* properties: For sets $X,Y \subseteq T' \subseteq T$, $X\to Y$ is valid in $R(T)$ iff it is valid in $R[T']$.

An MVD is a statement of the form $g{:}X\to\to Y$. g is *defined* for $R(T)$ or $\underline{R}(T)$ if X and Y are subsets of T. Let $Z=T-(X\cup Y)$. For a Z-value, z, we define $Y_{xz}=\{y\mid\langle x,y,z\rangle\in R\}$. g is *valid* in R iff $Y_{xz}=Y_{xz'}$ for each x,z,z' such that Y_{xz} and $Y_{xz'}$ are nonempty. This definition implies that g's validity depends on values assigned to Z, not just X and Y. If g is valid in $R(T)$, then it is valid in all projections of $R(T)$; the converse, however, does not hold. MVDs thus enjoy the *projectivity* property but *not inverse projectivity*.

The FD $X\to Y$ states that a unique Y-value is associated with each X-value; the MVD $X\to\to Y$ states that a unique *set* of Y-values is associated with each X-value. So essentially, an FD is just an MVD plus a functionality condition.

2.2 Inference Rules for Dependencies

Given a set of dependencies in a relation, it is often possible to deduce other dependencies that also hold in that relation. Consider once again the relation in Fig. 1. By examining its contents we see that OCCUPANT → RENT and LANDLORD → ADDRESS hold, although neither is expressly stated. This is not coincidental; these two FDs are logical consequences of the given set of FDs and MVDs.

Given a schema $\underline{R}=\langle T,\Gamma\rangle$, and a dependency g, Γ *implies* g *in* $\underline{R}$ if g holds in every instance of $\underline{R}$. Note that a dependency g' may hold in *some* instances of $\underline{R}$ without being implied by Γ. For example, ADDRESS → LANDLORD holds in Fig. 1 although it is *not* implied by the given dependencies.

It is possible to tell whether g is implied by Γ using systems of *inference rules* [3,6]. Inference rules permit us to derive new dependencies implied by a given set. A system of inference rules is *complete* if (a) every g derivable from Γ is in fact implied by Γ, and (b) every g implied by Γ is derivable using the rules. Fig. 3 shows three complete systems of inference rules for FDs and MVDs. The FD-rules are complete when FDs only are considered. The MVD-rules are complete for MVDs. When FDs and MVDs are considered, all three systems are needed for completeness. Fig. 3 also presents other rules that are useful, though not needed.

FIGURE 3. Inference rules for FDs and MVDs [3,6]

<u>FD-rules</u>:

FD_1 (reflexivity): If $Y \subseteq X$ then $X \to Y$.

FD_2 (augmentation): If $Z \subseteq W$ and $X \to Y$ then $XW \to YZ$.

FD_3 (transitivity): If $X \to Y$ and $Y \to Z$ then $X \to Z$.

Other useful rules:

FD_4 (pseudo-transitivity): If $X \to Y$ and $YW \to Z$ then $XW \to Z$.

FD_5 (union): If $X \to Y$ and $X \to Z$ then $X \to YZ$.

FD_6 (decomposition): If $X \to YZ$ then $X \to Y$ and $X \to Z$.

<u>MVD-rules</u>:

MVD_0 (complementation): Let $X+Y+Z=U$ and $Y \cap Z \subseteq X$; then $X \to\to Y$ iff $X \to\to Z$.

MVD_1 (reflexivity): If $Y \subseteq X$ then $X \to\to Y$.

MVD_2 (augmentation): If $Z \subseteq W$ and $X \to\to Y$ then $XW \to\to YZ$.

MVD_3 (transitivity): If $X \to\to Y$ and $Y \to\to Z$ then $X \to\to Z-Y$.

Other useful rules:

MVD_4 (pseudo-transitivity): If $X \to\to Y$ and $YW \to\to Z$ then $XW \to\to Z-YW$.

MVD_5 (union): If $X \to\to Y$ and $X \to\to Z$ then $X \to\to YZ$.

MVD_6 (decomposition): If $X \to\to Y$ and $X \to\to Z$ then $X \to\to Y \cap Z$, $X \to\to Y-Z$, $X \to\to Z-Y$.

<u>FD-MVD rules</u>:

$FD\text{-}MVD_1$: If $X \to Y$ then $X \to\to Y$.

$FD\text{-}MVD_2$: If $X \to\to Z$ and $Y \to Z'$, $Z' \subseteq Z$, and if Y and Z are disjoint, then $X \to Z'$.

Another useful rule:

$FD\text{-}MVD_3$: If $X \to\to Y$ and $XY \to Z$ then $X \to Z-Y$.

The set of all dependencies derivable from Γ using a complete system of rules is called the *closure of* Γ, denoted Γ^+. From the foregoing it should be clear that Γ^+ is the exact set of dependencies implied by Γ.

2.3 <u>The Membership Problem</u>

Given a set of dependencies Γ and a dependency g, the *membership problem* is to tell whether $g \in \Gamma^+$. For $g: X \to Y$ and Γ containing just FDs this problem is solved by determining the *maximum* set Z such that $X \to Z$ is in Γ^+. Then (by rules FD_5 and FD_6, Fig.3), $g \in \Gamma^+$ iff $Y \subseteq Z$. Z can be computed as follows:

1. Initialize $Z:=X$. (Since $X \to X$ by FD_1.)
2. If $U \to V$ is in Γ and $U \subseteq Z$, then set $Z:=Z+V$.
3. Repeat step 2 until more attributes can be added to Z.

A straightforward implementation yields an $O(n^2)$ time algorithm [7]; linear time implementation of this algorithm is described in [5].

For MVDs the best known membership algorithm requires $O(n^4)$ time [4].

2.4 <u>Coverings</u>

A *covering* of Γ is any set $\hat{\Gamma}$ such that $\hat{\Gamma}^+ = \Gamma^+$. $\hat{\Gamma}$ is *nonredundant* if no proper subset of it is a covering. One can obtain a nonredundant covering of Γ as follows. A dependency $g \in \Gamma$ is redundant iff $g \in (\Gamma-\{g\})^+$. For each $g \in \Gamma$ the above test is performed using the membership algorithm, and g is removed from Γ if it is found to be redundant.

2.5 <u>Inherently Difficult Dependency Problems</u>

We list here two inherently difficult dependency problems. Other such problems are presented in [5,28].

<u>Key Finding</u>: Given a set of FDs F over attributes U, a relation scheme $\underline{R}(X)$ where $X \subseteq U$, and a subset of R's keys, determine whether $\underline{R}$ has any other keys. This problem is NP-complete [5] (i.e., probably requires exponential time [2]).

<u>Key Listing</u>: Given F and $\underline{R}$ as above, list all keys of $\underline{R}$. This problem has exponential worst-case time since there are relation schemes with an exponential number of keys [40].

3. THE SCHEMA DESIGN PROBLEM

We now return to the problem of schema design. Our treatment considers one particular schema design scenario. We assume that a schema S_ϕ containing a single relation scheme is given. The problem is to design a schema S_D that is *equivalent* to S_ϕ, but is *better* in some specified way. Let $S_\phi = \{\underline{U} = \langle T,\Gamma\rangle\}$ and $S_D = \{\underline{R_i} = \langle T_i,\Gamma_i\rangle \mid i=1,\dots,n\}$. In our scenario S_D contains "projections" of $\underline{U}$; i.e., each $T_i \subseteq T$ and Γ_i is "inherited" from Γ. For FDs, "inheritance" means Γ_i is a covering of the FDs in Γ^+ that are defined for $\underline{R_i}$. For MVDs, the situation is more complicated and will not be elaborated here. An instance U of $\underline{U}$ is represented in S_D's database by $\{U[T_i] \mid i=1,\dots,n\}$.

Our study of schema design can now be considered to be a study of the mapping between S_ϕ and S_D and between the set of instances of S_ϕ and the sets of instances of S_D.

4. THE PRINCIPLE OF REPRESENTATION

A clear requirement for schema S_D to replace S_ϕ is that S_D and S_ϕ be equivalent; that is, S_D must *represent the same information* as S_ϕ. Different researchers formulate this concept in different ways--ways that lead to startingly different conclusions. In the following, let $S_\phi = \{\underline{U} = \langle T,\Gamma\rangle\}$ and $S_D = \{\underline{R_i} = \langle T_i,\Gamma_i\rangle \mid i=1,\dots,n\}$.

Definition Rep1. S_D represents the same information as S_ϕ if they contain the same attributes; that is, if $\cup_{i=1}^{n} T_i = T$.

This definition is inadequate because it ignores relationships among attributes. By this definition, the schemas in Figs. 4 and 5 are equivalent to the one in Fig. 2, even though they contain no data dependencies.

FIGURE 4. A schema equivalent to one in Fig. 2 under Def. Rep1.

$$S_D = \{\underline{R}_1, \underline{R}_2, R_3\}$$

$\underline{R}_1$ = <{LANDLORD,RENT,PETS},{}>

$\underline{R}_2$ = <{ADDRESS,APT#},{}>

$\underline{R}_3$ = <{OCCUPANT},{}>

FIGURE 5. Another schema equivalent to one in Fig. 2 under Rep1.

$$S_D = \{\underline{R}_1, \underline{R}_2, \underline{R}_3, \underline{R}_4, \underline{R}_5, \underline{R}_6\}$$

$\underline{R}_1$ = <LANDLORD,{}>

$\underline{R}_2$ = <RENT,{}>

$\underline{R}_3$ = <ADDRESS,{}>

$\underline{R}_4$ = <APT#,{}>

$\underline{R}_5$ = <OCCUPANT,{}>

$\underline{R}_6$ = <PETS,{}>

Definition Rep2. S_D represents the same information as S_ϕ if they have the same attributes and the same data dependencies.

When only FDs are involved, this definition can be made precise. The FDs of S_ϕ are Γ^+. The FDs of S_D are $(\cup_{i=1}^{n} \Gamma_i)^+$. S_D represents S_ϕ if $\Gamma^+ = (\cup_{i=1}^{n} \Gamma_i)^+$, i.e., if $(\cup_{i=1}^{n} \Gamma_i)$ is a covering of Γ.

However, there is a problem with the definition as stated. The inference rules in Sec. 2 are only defined with respect to dependencies in a *single* relation. Since S_D involves *multiple* relations, it is not obvious that those inference rules can validly be applied to it. Suppose, $S_\phi = \{\underline{U} = \langle T = \{X,Y,Z\}, \Gamma = \{X \to Y; Y \to Z\}\rangle$, and $S_D = \{\underline{R}_1 = \langle T_1 = \{X,Y\}, \Gamma_1 = \{X \to Y\}\rangle; \underline{R}_2 = \langle T_2 = \{Y,Z\}, \Gamma_2 = \{Y \to Z\}\rangle\}$. Notice that $X \to Z$ is in Γ^+ and that S_D represents S_ϕ by the above definition. Yet S_D does not even contain a relation scheme in which $X \to Z$ is defined!

This problem is rectified by the "universal relation assumption" (Sec. 1) and the "inverse projectivity property of FDs" (Sec. 2). Let R_1 and R_2 be instances of $\underline{R}_1$ and $\underline{R}_2$; define $\underline{R}_{12} = \langle T_{12} = T_1 \cup T_2, \Gamma_{12} = \Gamma_1 \cup \Gamma_2\rangle$; and define $R_{12} = R_1 * R_2$. From the inverse projectivity property it can be shown that R_{12} is an instance of $\underline{R}_{12}$ if R_1 and R_2 are instances of $\underline{R}_1$ and $\underline{R}_2$. Thus the FD $X \to Z$ (which is in Γ_{12}^+) is valid in R_{12}. Moreover, by the universal relation assumption R_1 and R_2 are projections of U, as is $R_{12}[XZ]$. Consequently the user can obtain the "extension" of $X \to Z$ from S_D, even though $X \to Z$ is not explicitly represented. In fact, all FD inference rules can be "simulated" by relational operators applied to relations containing the FDs. It follows that all FDs in Γ^+ can be retrieved from S_D if S_D's schemes contain a covering of Γ.

MVDs, on the other hand, do not possess the inverse projectivity property, and definition Rep2 is not easily generalized to them. More research is needed to formulate a suitable generalization of Rep2 for MVDs.

Definition Rep3. S_D represents the same information as S_ϕ if they have the same attributes and the databases of S_D contain the same data as the databases of S_ϕ.

In contrast to Rep2, this definition stresses the data component of equivalence. Two schemas are equivalent under Rep3 if at all times their databases contain the same information, albeit in different formats.

The definition is formalized by the concept of *lossless join* [1]. Suppose U(T) is an instance of $\underline{U}$ and the corresponding set of instances of S_D is $\{R_i(T_i) = U[T_i] \mid i=1,\ldots,n\}$. To answer a query involving, say, all attributes of T, we must reconstruct U from $\{R_i\}$ via the join operator. If $U = R_1 * \ldots * R_n$, then U can be precisely reconstructed from its projections. If, however, $U \subset R, \ldots * R_n$, the join contains tuples that are not in U, and $\{R_i\}$ is not a faithful representation. This phenomenon is called a *lossy join* and is illustrated in Fig. 6.

FIGURE 6. An Example of a lossy join.

Let S_ϕ = {RENTAL-UNITS} defined in Fig. 2, with instance of Fig. 1.

Let S_D = {LAND-APT#,APT#-RENT,PERSON-PETS}

LAND-APT# = <{LANDLORD,APT#},{}>
APT#-RENT = <{APT#,RENT},{}>
PERSON-PETS = <{OCCUPANT,PETS,ADDRESS},
{OCCUPANT→ADDRESS;OCCUPANT→→PETS}>

Instances corresponding to Fig. 1 are

LAND-APT#(LANDLORD,APT#)

Wizard,	#1
Wizard,	#1
Wizard,	#2
Codd,	#1
Codd,	#2

APT#-RENT(APT#,RENT)

#3,	\$ 50
#1,	\$ 50
#2,	\$ 75
#1,	\$500
#2,	\$600

LAND-APT#*APT#-RENT =

Attributes: LANDLORD, APT#, RENT

Tuples:

Wizard,	#3,	\$ 50
Wizard,	#1,	\$ 50
Wizard,	#1,	\$500
Wizard,	#2,	\$ 75
Wizard,	#2,	\$600

FIGURE 6 continued

Attributes:	LANDLORD,	APT#,	RENT
Tuples:	Codd,	#1,	\$ 50
	Codd,	#1,	\$500
	Codd,	#2,	\$ 75
	Codd,	#2,	\$600

Note that each LANDLORD is associated with RENTs charged by the other.

Formally we say that S_D has the *lossless join property* if for each instance U of $\underline{U}$,

$$U = \mathop{*}_{i=1}^{n} U[T_i].$$

When only pairs of relations are considered, we have the following results.

FACT 1: If $\underline{U} = \langle T,F\rangle$ (that is, only FDs are given) then for sets T_1, T_2 such that $T_1 \cup T_2 = T$, $\{\underline{R}_1(T_1), \underline{R}_2(T_2)\}$ has the lossless join property iff either $T_1 \cap T_2 \to T_1$ or $T_1 \cap T_2 \to T_2$ is in Γ^+[33].

FACT 2: For $\underline{U} = \langle T,\Gamma\rangle$ and for T_1, T_2 as above, $\{\underline{R}_1(T_1), \underline{R}_2(T_2)\}$ has the lossless join property iff $T_1 \cap T_2 \twoheadrightarrow T_1$ (and, by rule MVD_0, $T_1 \cap T_2 \twoheadrightarrow T_2$) is in Γ^+ [23].

These facts are stated as properties of universally quantified sets of instances; i.e., the conditions of Facts 1 & 2 hold iff *all* instances of the given schemas have lossless joins. It is possible, though, for the conditions not to hold, yet for *specific* instances to have lossless joins, nonetheless. Facts 1 & 2 can be adapted for specific instances as follows.

FACT 1': Given $\underline{U} = \langle T,\Gamma\rangle$, and T_1, T_2 as above. An instance $U = U[T_1]*U[T_2]$ if (but *not* only if) $T_1 \cap T_2 \to T_1$ or $T_1 \cap T_2 \to T_2$ holds in U.

FACT 2': Given $\underline{U} = \langle T,\Gamma\rangle$, and T_1, T_2 as above. An instance $U = U[T_1]*U[T_2]$ iff $T_1 \cap T_2 \twoheadrightarrow T_1$ (and by MVDs $T_1 \cap T_2 \twoheadrightarrow T_2$) holds in U.

When more than pairs of relations are considered, the situation is more complex. An algorithm for deciding the lossless join property in general is presented in [1]. The algorithm requires polynomial time for FDs but may require exponential time for MVDs. Another interesting result is that for all $n > 2$ there are sets of n relation schemes that have the lossless join property, for which no proper subset has this property.

Definition Rep4. S_D represents the same information as S_ϕ iff there exists a one-to-one mapping between the databases of S_ϕ and databases of S_D.

Rep4 combines definitions Rep2 and Rep3. Rep3 says that every database of S_ϕ is represented by a unique database of S_D. Rep2 says that every database of S_D satisfies the same dependencies as S_ϕ, and hence represents a legal database of S_ϕ. Together, Rep2 and Rep3 imply Rep4.

If only FDs are given, Rep4 is identical to the notion of *independent components* [31], and the following is proved:

Let $\underline{U} = \langle T,F\rangle$, $\underline{R}_1 = \langle T_1,F_1\rangle$ and $\underline{R}_2 = \langle T_2,F_2\rangle$. $\{R_1,R_2\}$ are independent components of $\underline{U}$ iff (a) $(F_1 \cup F_2)^+ = F^+$, and (b) F^+ contains $T_1 \cap T_2 \to T_1$ or $T_1 \cap T_2 \to T_2$ [31].

Comparison of Definitions. Fig. 5 illustrated a schema equivalent to the schema of Fig. 2 by Rep1 but not by Rep2, Rep3, or Rep4. Fig. 7 differentiates between Rep2 and Rep3. Fig. 7(a) is similar to an example in [16, p. 165]. S_D is equivalent to S_ϕ under Rep3 but not Rep2; it would be considered a good design by [16,23], but not by [7]. In Fig. 7(b), S_D is equivalent to S_ϕ by Rep2 but not by Rep3; it would be approved by [7], but not [23,37]. These differences of opinion are examined further in later sections.

FIGURE 7. Situations where Rep2 differs from Rep3.

S_ϕ = {DWELLER = <{ADDRESS,APT#,OCCUPANT},
{ADDRESS,APT# → OCCUPANT;
OCCUPANT → ADDRESS;OCCUPANT → APT#}>}

S_D = {ADD-OCC = <{ADDRESS,OCCUPANT},
{OCCUPANT → ADDRESS}>;
APT-OCC <{APT#,OCCUPANT},{OCCUPANT → APT#}>}

S_D does not Rep2-represent S_ϕ. S_D Rep3-represents S_ϕ.

S_ϕ = {RENTAL = <{LANDLORD,ADDRESS,APT#,OCCUPANT},
{LANDLORD → ADDRESS;
ADDRESS,APT# → OCCUPANT;
OCCUPANT → ADDRESS;OCCUPANT → APT#}>}

S_D = {OWNER = <{LANDLORD,ADDRESS},
{LANDLORD → ADDRESS}>;
DWELLER = <{ADDRESS,APT#,OCCUPANT},
{ADDRESS,APT# → OCCUPANT;
OCCUPANT → ADDRESS;OCCUPANT → APT#}>}

S_D Rep2-represents S_ϕ. S_D does not Rep3-represents S_ϕ.

5. THE PRINCIPLE OF SEPARATION

The next question is to understand how S_D can be "better than" S_ϕ. One way is for "independent relationships" to be represented by S_D in independent relation schemes. To illustrate this point, let S_ϕ be the RENTAL-UNITS scheme of Figs. 1 & 2, and suppose we want to add a new LANDLORD to the database. This can only be done if values for other attributes are given, too. The new LANDLORD must be associated with an ADDRESS; the ADDRESS must be associated with an APT#; the ADDRESS,APT# pair requires a RENT and an OCCUPANT; and the OCCUPANT needs PETS. So to add a new LANDLORD, S_ϕ forces us to add information that is at most distantly related to him. By the same token, when the last PET of the last OCCUPANT of the last APT# of a given ADDRESS runs away, the association between LANDLORD and ADDRESS is also destroyed.

Another problem with S_ϕ is data redundancy. Each LANDLORD is represented in four tuples although each only owns one building. To change the building owned by Codd, say, requires that all four tuples with ADDRESS = "3 NF St" be updated. If some of these tuples were forgotten, the database would be *inconsistent*, meaning that some dependencies would no longer hold. In this case, LANDLORD $\twoheadrightarrow$ ADDRESS would no longer hold.

These difficulties are caused by a lack of separation in S_ϕ. To overcome these difficulties, a series of *database normal forms* have been proposed, four of which are of interest. Before defining them, we present several preliminary concepts.

Let $S = \{\underline{R}_i = \langle T_i, \Gamma_i \rangle | i = 1, \ldots, n\}$ be a schema and let $\Gamma = (\bigcup_{i=1}^{n} \Gamma_i)$. (1) *Superkey*--Let $X \subseteq T_i$; X is a superkey of $\underline{R}_i$ if $X \to T_i$ is in Γ^+. (2) *Key*--Let $X \subseteq T_i$; X is a key of $\underline{R}_i$ if X is a superkey and no $X' \subset X$ is. (3) *Prime attribute*--Let $A \in T_i$; A is prime in $\underline{R}_i$ if A is in any key of $\underline{R}_i$. (4) *Transitive dependence*--Let $A \in T_i$ and $X \subseteq T_i$; A is transitively dependent on X in $\underline{R}_i$ if there exists $Y \subseteq T_i$ such that $X \to Y \in \Gamma^+$, $Y \to A \in \Gamma^+$, $Y \to X \notin \Gamma^+$, and $A \notin Y$. (5) *Trivial FD*--$X \to Y$ is trivial, meaning it holds in *all* relations, if $Y \subseteq X$. (6) *Trivial MVD*--$X \twoheadrightarrow \phi$ and $X \twoheadrightarrow T_i - X$ are trivial in $\underline{R}_i(T_i)$.

We now define four normal forms of interest.

1. Third Normal Form (abbr. 3NF)*: [14] $\underline{R}_i \in S$ is in 3NF if none of its nonprime attributes is transitively dependent on any of its keys
2. Boyce-Codd Normal Form (BCNF): [15] Let $f: X \to Y$ be any nontrivial FD in Γ^+, defined on $\underline{R}_i \in S$. $\underline{R}_i$ is in BCNF if for all such f, X is a superkey of $\underline{R}_i$.
3. Weak Fourth Normal Form (W4NF): Let $g: X \twoheadrightarrow Y \in \Gamma^+$ be any nontrivial MVD in $\underline{R}_i \in S$. $\underline{R}_i$ is in W4NF if it is in 3NF and all such g are FDs.
4. Fourth Normal Form (4NF): [23] Let $g: X \twoheadrightarrow Y \in \Gamma^+$ be any nontrivial MVD in $\underline{R}_i \in S$. $\underline{R}_i$ is in 4NF if for all such g, X is a superkey of $\underline{R}_i$.

Notice that 3NF is a weak version of BCNF, and W4NF is a weak version of 4NF. Also W4NF implies 3NF and 4NF implies BCNF. BCNF and 4NF always succeed in separating independent relationships into separate schemes. This is illustrated in Fig. 8. Notice that S_D in Fig. 8 Rep2-represents RENTAL-UNITS. There are cases, though, where the stronger normal forms cannot be achieved and we must settle for the weaker forms. Fig. 9 shows an example of this sort. The following formalize this observation. Let $S_\phi = \{\underline{U} = \langle T, F \rangle\}$.

FACT 3: There always exists a 3NF schema that Rep2-represents S_ϕ [7].

FACT 4: There need not exist a BCNF schema that Rep2-represents S_ϕ. Moreover the question, "Is schema S in BCNF?" is NP-hard [5].

FACT 5: There need not exist a 4NF schema that Rep2-represents S_ϕ. (Follows from Fact 4 when $G = \phi$.) It is not known whether a W4NF scheme Rep2-representing S_ϕ need always exist.

FACT 6: There always exists a 4NF schema that Rep3-represents S [23]. It follows that a BCNF schema Rep3-representing S_ϕ is always achievable, too.

*1NF simply requires that relations be "flat", non-hierarchical. 2NF is a weak form of 3NF and is subsumed by it [14,16].

FIGURE 8. 4NF schema and instance corresponding to RENTAL-UNITS (Figs. 1 & 2)

S_D = {OWNS, CHARGES, LIVES, LOVES}

OWNS = <{LANDLORD, ADDRESS}, {LANDLORD $\twoheadrightarrow$ ADDRESS}>
CHARGES = <{ADDRESS, APT#, RENT}, {ADDRESS, APT# $\to$ RENT}>
LIVES = <{OCCUPANT, ADDRESS, APT#}, {OCCUPANT $\to$ ADDRESS, APT#}>
LOVES = <{OCCUPANT, PETS}, {OCCUPANT $\twoheadrightarrow$ PETS{>

OWNS (LANDLORD, ADDRESS)

Wizard	Oz
Codd	3 NF St

CHARGES (ADDRESS, APT#, RENT)

Oz,	#3	$ 50
Oz,	#1	$ 50
Oz,	#2	$ 75
3 NF St,	#1	$500
3 NF St,	#2	$600

LIVES (OCCUPANT, ADDRESS, APT#)

Tinman,	Oz,	#3
Witch,	Oz,	#1
Lion,	Oz,	#2
Beeri,	3 NF St,	#1
Bernstein,	3 NF St,	#1
Goodman,	3 NF St,	#2

LOVES (OCCUPANTS, PETS)

Tinman,	Oilcan
Witch,	Bat
Witch,	Snake
Lion,	Mouse
Beeri,	Fish
Bernstein,	Dog
Bernstein,	Rhino
Goodman,	Cat

FIGURE 9. W4NF schema and instance.

S_ϕ = {RENTAL-UNITS' =
<{LANDLORD, ADDRESS, APT#, RENT, OCCUPANT, PETS},
{ADDRESS, APT# $\to$ RENT; OCCUPANT $\to$ ADDRESS, APT#,
LANDLORD $\twoheadrightarrow$ ADDRESS; OCCUPANT $\twoheadrightarrow$ PETS;
ADDRESS, APT# $\to$ LANDLORD}>

S_D = {OWNS', CHARGES, LIVES, LOVES}, CHARGES, LIVES, LOVES same as in Fig. 8.

OWNS' = <{LANDLORD, ADDRESS, APT#}, {LANDLORD $\twoheadrightarrow$ ADDRESS; ADDRESS, APT# $\to$ LANDLORD}>

CHARGES, LIVES, LOVES in 4NF (from Fig. 8)

OWNS' in W4NF since LANDLORD $\to$ ADDRESS implied by S_ϕ's dependencies:

(1) OCCUPANT $\to$ ADDRESS, APT# $\Rightarrow$ OCCUPANT $\to$ ADDRESS (FD_6)
(2) LANDLORD $\twoheadrightarrow$ ADDRESS and OCCUPANT $\to$ ADDRESS $\Rightarrow$ LANDLORD $\to$ ADDRESS ($FD\text{-}MVD_2$)

Extension of OWNS', given data in Figs. 1 and 8.

OWNS' (LANDLORD, ADDRESS, APT#)

Wizard,	Oz,	#3
Wizard,	Oz,	#1
Wizard,	Oz,	#2
Codd,	3 NF St,	#1
Codd,	3 NF St,	#2

Another observation to make from Fig. 9 is that W4NF doesn't achieve total separation in the way 4NF does. OWNS' has redundant information and suffers the same kind of update anomalies as RENTAL-UNITS does. The same is true of 3NF vs. BCNF. And since 4NF and BCNF cannot always be achieved under Rep2 (Facts 4 & 5), we must conclude that Rep2 and total separation are incompatible concepts. This result is both surprising and fundamental; it holds for non-computerized databases as well as computerized ones, and has applicability in all data models.

This result has been interpreted differently by some workers [16,24] who argue that BCNF and 4NF schemas should be obtained even if Rep2 is not achieved. We saw such a case in Fig. 7(a), which we replicate in Fig. 10. In that example, S_D violates Rep2 because it does not include ADDRESS,APT# → OCCUPANT. Without this FD legal instances of S_D can correspond to illegal instances of S_ϕ, and may represent illegal conditions in the real world (see Fig. 10(b)). It is suggested in [16] that these illegal instances be prevented by adding ADDRESS,APT# → OCCUPANT to S_D as an "interrelational constraint." However, because Rep2 is incompatible with BCNF in this case, this suggestion is futile. If we add the suggested interrelational constraint, the two *relations* can no longer be updated independently, which simply defeats the original goal of separation.

In other words, while total separation is a goal of schema design, there simply are cases where it cannot be achieved.

FIGURE 10. An instance of S_D that is not an instance of S_ϕ.

S_ϕ = {DWELLER = <{ADDRESS,APT#,OCCUPANT} {ADDRESS,APT# → OCCUPANT; OCCUPANT→ADDRESS; OCCUPANT→APT#}>}

S_D = {<u>ADD-OCC</u> = <{ADDRESS,OCCUPANT}; {OCCUPANT → ADDRESS}>;

<u>APT-OCC</u> = <{APT#,OCCUPANT}; {OCCUPANT → APT#}>}.

S_D does not Rep2-represent S_ϕ. S_D Rep3-represents S_ϕ.

(a)

Relation: ADD-OCC

Attributes: <u>ADDRESS,OCCUPANT</u>

Tuples:

Oz,	Tinman
Oz,	Witch
Oz,	Lion
Oz,	Scarecrow

Relation: APT-OCC

Attributes: <u>APT#, OCCUPANT</u>

Tuples:

#3,	Tinman
#1,	Witch
#2,	Lion
#2,	Scarecrow

Each relation has legal contents--all dependencies hold. But, ADDRESS,APT# → OCCUPANT does not hold in ADD-OCC*APT-OCC.

ADD-OCC*APT-OCC =

Attributes: <u>ADDRESS, APT#, OCCUPANT</u>

Tuples:

Oz,	#3,	Tinman
Oz,	#1,	Witch
Oz,	#2,	Lion
Oz,	#2,	Scrarecrow

(b)

6. THE PRINCIPLE OF MINIMAL REDUNDANCY

Another goal in designing S_D is *minimal redundancy*; S_D must contain the information needed to represent S_ϕ but it should not contain the information redundantly. The meaning of minimal redundancy depends on the definition of representation. Only by knowing what it means to represent information can we judge whether a certain representation is redundant.

Virtually all work on schema design adopts some notion of minimal redundacy, although often this point is addressed intuitively. Consequently our treatment of redundancy must be sketchier than the previous sections. We present here different definitions of redundancy analogous to the definitions of representation in Sec. 4. In the following, let $S_D = \{\underline{R_i} = \langle T_i, \Gamma_i\rangle \mid i = 1,\ldots,n\}$.

<u>Definition Red1</u>. $\underline{R_i} \in S_D$ is redundant if $T_i \subseteq \cup_{j=1, j\neq i}^{n} T_j$. This approach, like Definition Rep1, is unsatisfactory since it does not account for relationships among attributes. Also, minimal redundancy under Red1 is always attained in S_ϕ since each attribute appears only once.

<u>Definition Red2</u>. $\underline{R_i} \in S_D$ is redundant if $\underline{R_i}$'s data dependencies are represented by the other schemes. For the case of FDs, the definition can be made formal. $\underline{R_i} \in S_D$ is redundant if $(\cup_{j=1}^{n} \Gamma_j)^+ = (\cup_{j=1, j\neq i}^{n} \Gamma_j)^+$. Note that the FDs of Γ_i need not be explicitly represented. Rather, they need only be derivable from the FDs in the other schemes. As for Rep2, this definition does not easily generalize to MVDs since rules for manipulating MVDs in different relations are not known.

<u>Definition Red3</u>. $\underline{R_i} \in S_D$ is redundant if for each database of S_D, the data in R_i is contained in $\{R_j \mid j = 1,\ldots,n,\ j \neq i\}$. For this definition to be meaningful, a database of S_D must be viewed as a set of related relations, since if relations can assume independent values, no relation scheme is ever redundant. The universal relation assumption (Sec.1) provides the necessary connection and leads to the following. $\underline{R_i}$ is redundant in S_D if $\mathop{*}_{j=1}^{n} R_j = \mathop{*}_{j=1, j\neq i}^{n} R_j$, for all databases of S_D.

<u>Definition Red4</u>. $\underline{R_i} \in S_D$ is redundant if there is a one-to-one correspondence between the set of instances of S_D and the set of instances of $S_D - \{\underline{R_i}\}$. This definition, like Rep1, combines data and dependency aspects of schema design.

We note, in conclusion, that other approaches to redundancy are possible, e.g., using as a measure the number of data items in relations, etc.

7. SCHEMA DESIGN METHODS

Traditionally, schema design has been called "database normalization" in the literature in this area and two approaches are prominent: *synthesis* [5,7], and *decomposition* [14,20,21,25,41]. This section describes both approaches, explaining how they interpret and achieve the schema design principles discussed earlier.

The key difference between synthesis and decomposition lies in the definition of *representation* that each adopts. In synthesis S_D *Rep2-represents* input S_ϕ, whereas with decomposition S_D *Rep3-represents* S_ϕ.* This difference leads to a series of other discrepancies between the methods: (1) Since Rep2 is not compatible with total separation (Sec.5), synthesis can only achieve 3NF and not higher normal forms; decomposition, on the other hand, is not limited in this way. (2) Rep2 leads to the Red2 definition of redundancy, while Rep3 leads to Red3; therefore synthesis strives for minimality of dependencies while decomposition strives for minimality of data content. (3) Because definitions Rep2 and Red2 do not easily extend to MVDs, it is not known how (or if) synthesis can handle MVDs; decomposition, on the other hand, is straightforwardly extendable to MVDs. (4) Finally, as explained in Sec. 5, Rep3 does not guarantee that all instances of S_D correspond to legal instances of S_ϕ; thus schemas produced by decomposition admit instances that would not be permitted by synthesized schemas. These differences are summarized in Figure 11.

FIGURE 11. Differences between principal Normalization Methods

Method	Synthesis	Decomposition	Decomposition
Described by	Bernstein [7]	Fagin [25]	Rissanen [31]
Definition of Representation	Rep2, "S_D has same dependencies as S_ϕ"	Rep3, "S_D has same data content as S_ϕ"	Rep4, "S_D and S_ϕ databases are 1-to-1"
Dependencies	FDs	FDs + MVDs	FDs
Normal Form	3NF	4NF, BCNF	3NF
Definition of Redundancy	Red2, "redundancy of dependencies" (attained)	Red3, "redundancy of data content" (not attained by current algorithms)	Red4, "both Red2 + Red3" (not attained by current algorithms)
Instances admitted by S_D	Same as S_ϕ	More than S_ϕ	Same as S_ϕ

7.1 The Synthesis Approach

We discuss the synthesis approach in terms of the specific method of [7]. A central concept of this method is *embodied* FDs, which are FDs implied by keys. Formally, given $\underline{R_i} = \langle T_i, \Gamma_i \rangle$, $X \rightarrow A$ is embodied in $\underline{R_i}$ if $X \rightarrow A \in \Gamma_i$, and X is a superkey of R_i. Fig. 12 presents a simplified synthesis algorithm (called SYN1) that uses embodied FDs to construct an S_D Rep2-representing S_ϕ. SYN1 is a first step towards a correct synthesis algorithm. SYN1 is not yet correct because S_D is not necessarily in 3NF, so transitive dependencies can be exhibited within *individual* FDs, due to *extraneous attributes* in their left hand sides. An attribute is extraneous in an FD if it could be eliminated from the FD without affecting the closure (F^+).

FIGURE 12. Simplified Synthesis Algorithm

Algorithm SYN1

Input: $S_\phi = \{\underline{U} = \langle T, F \rangle\}$

Output: $S_D = \{\underline{R_i} = \langle T_i, F_i \rangle\}_{i=1,\dots,n}$

1. (Find Covering). Find a nonredundant covering $\hat{F}$ of F.
2. (Partition). Partition $\hat{F}$ into "groups", F_i, $i = 1,\dots,n$, such that all FDs in each F_i have the same left hand side, and no two groups have the same left hand side.
3. (Construct Relations). For each F_i construct a relation scheme $\underline{R_i} = \langle T_i, F_i \rangle$ where T_i = all attributes appearing in F_i.

Important Fact: The left hand side of every FD in F_i is a *superkey* of $\underline{R_i}$; each FD in F_i is *embodied* in $\underline{R_i}$.

Let us precede SYN1 with a step that eliminates extraneous attributes from the left sides of FDs in F, and call the resulting algorithm SYN1'. SYN1' produces schemas that Rep2-represent the input and are guaranteed to be in 3NF. Algorithm SYN1' thus meets the representation and separation goals of schema synthesis.

The next step is to achieve minimality. Let $\hat{F}$ be the nonredundant covering of F obtained by SYN1' after excising extraneous attributes, and suppose $\hat{F}$ includes $V \rightarrow W$ and $X \rightarrow Y$, $X \neq V$. Clearly these FDs will be embodied in different relation schemes. But suppose V and X are *equivalent*; i.e., $V \rightarrow X$ and $X \rightarrow V$ are in F^+. Then $V \rightarrow X$ and $X \rightarrow V$ can be embedded in one relation scheme with *both* V and X as keys. Doing so reduces the number of synthesized schemes and makes explicit the equivalence of X and V.

So, we add a stage to SYN1' to find and merge all relation schemes with equivalent keys. Unfortunately, this modification takes a step backward: it no longer produces 3NF schemes! When we merge relation schemes we also add FDs to $\hat{F}$ from

*A decomposition approach is suggested by [Rissanen,77] in which S_D Rep4-represents S_ϕ. This approach is algorithmically similar to the other decomposition approaches so will not be discussed separately.

$(F^+-\hat{F})$ which may thereby cause F to become redundant. A final stage is needed to eliminate this redundancy. This modification brings us to algorithm SYN2 (Fig. 13), which is our final schema synthesis algorithm. The following facts, are proved in [7], establishing that SYN2 achieves the three schema design principles. Given $S_\phi = \{\underline{U} = <T,F>\}$ and S_D = the result of applying SYN2 to S_ϕ.

FACT 7: S_D Rep2-represents S_ϕ.

FACT 8: S_D is in 3NF.

FACT 9: S_D is minimally redundant under definition Red2. In fact S_D is minimal in an even stronger sense. Let $S_D = \{S_D' | S_D'$ Rep2-represents S_ϕ and all FDs in S_D' are *embodied* FDs$\}$. S_D contains no more relation schemes than any other scheme in S_D. In other words, S_D is the smallest schema that can Rep2-represent S_ϕ using just keys.

FIGURE 13. A Correct Synthesis Algorithm [5].

Algorithm SYN2

Input: $S_\phi = \{\underline{U} = <T,F>\}$

Output: $S_D = \{\underline{R}_i = <T_i,F_i> | i=1,\ldots,n\}$

1. (Eliminate Extraneous Attributes) Eliminates extraneous attributes from the left side of each FD in F, producing the set F'.
2. (Find Covering) Find a nonredundant covering $\hat{F}$ of F'.
3. (Partition) Partition $\hat{F}$ into groups F_i, $i=1,\ldots,n$, as in step 2 of SYN1.
4. (Merge Equivalent Keys) Set $J:=\emptyset$. For each pair of groups F_i, F_j with left hand sides X_i, X_j do the following: If $X_i \rightarrow X_j \in F^+$ and $X_j \rightarrow X_i \in F^+$, merge F_i and F_j, add $X_i \rightarrow X_j$ and $X_j \rightarrow X_i$ to J, and remove them from $\hat{F}$.
5. (Eliminate Transitive Dependencies) Find a minimal $\hat{F}' \subseteq \hat{F}$ such that $(\hat{F}+J)^+ = (\hat{F}'+J)$. Delete each element of $\hat{F}-\hat{F}'$ from the group in which it appears. For each $X_i \rightarrow X_j$ in J, add it to the corresponding group.
6. (Construct Relations) For each F_i construct a relation scheme $\underline{R}_i = <T_i,F_i>$ where T_i = all attributes appearing in F_i.

7.2 The Decomposition Approach

Fig. 14 shows a typical decomposition algorithm (which we call algorithm DEC) adapted from [25]. DEC achieves the representation and separation goals of decomposition but does not achieve minimal redundancy. These conclusions are stated formally as follows. Given $S_\phi = \{\underline{U} = <T,\Gamma>\}$ and S_D = the result of applying DEC to S_ϕ.

FACT 10: S_D Rep3-represents S_ϕ.

Reason: Whenever a scheme $\underline{R}$ is decomposed into $\underline{R}_1$ and $\underline{R}_2$ (in step 3 of DEC) $\underline{R}_1,\underline{R}_2$ has the lossless join property (by Fact 2).

FIGURE 14. Basic Decomposition Algorithm

Algorithm DEC

Input: $S_\phi = \{\underline{U} = <T,\Gamma>\}$

Output: $S_D = \{\underline{R}_i = <T_i,\Gamma_i> | i=1,\ldots,n\}$

1. (Initialize) Set $k:=\emptyset$.
2. (Test for Separation) If all schemes in S_k are in 4NF, then output $S_D:=S_k$.
3. (Decompose) Set $S_{k+1}:=\emptyset$. Let $\underline{R}_i = <T_i,\Gamma_i>$ be any non-4NF scheme in S_k, and set $S_k:=S_k-\underline{R}_i$. Decompose $\underline{R}_i$ into $\underline{R}_{i,1}$ and $\underline{R}_{i,2}$ as follows: (1) Let $X \twoheadrightarrow Y$ be any non-trivial MVD in Γ defined on $\underline{R}_1$. (2) Let $\underline{R}_{i,1}:=<X \cup Y, \{g \in \Gamma_i | g$ is defined on $\underline{R}_{i,1}\}$. (3) Define $\underline{R}_{i,2} = <T_i - Y$ $\{g \in \Gamma_i | g$ is defined on $\underline{R}_{i,2}\}$. (4) Set $S_{k+1}:=S_{k+1} \cup \{\underline{R}_{i,1},\underline{R}_{i,2}\}$. (5) Repeat for all $\underline{R}_i \in S_k$.
4. (Eliminate Some Redundancy) For each $\underline{R}_i,\underline{R}_j \in S_{k+1}$. If $T_i \subseteq T_j$ set $S_{k+1}:=S_{k+1} - \{\underline{R}_i\}$.
5. (Iterate) Set $k:=k+1$. Go to step 2.

FACT 11: S_D is in 4NF.

Reason: DEC will not stop until S_D is in 4NF.

FACT 12: S_D is *not* necessarily minimally redundant under Rep3 (or most other reasonable definitions).

Reason: Fig. 15 shows two ways DEC could decompose the same schema, one of which is minimal and one of which is not. Few minimality facts have been established regarding decomposition, and it is not even known whether minimal schemas can be produced by *non-deterministic* decomposition. Also, it is not known whether decomposition can consider *coverings* of dependencies rather than entire *closures*; in the specific case of Fig. 15(a) minimality would be guaranteed if S_ϕ were decomposed using a nonredundant, nonextraneous covering of Γ.

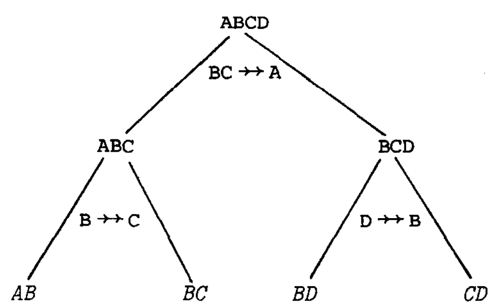
Algorithmic aspects of decomposition have not been considered fully either, and current algorithms have high computational complexity. For example, DEC is probably very slow, because the question "Is schema S in 4NF?" is NP-hard and DEC asks this question repeatedly. A related problem is caused by using closures rather than coverings. Closures can be exponentially large and their use can lead to exponential worst case running time.

Another problem is that decompositions are not unique. At each stage the algorithm may have several decomposition choices with different choices leading to very different outputs (e.g., Fig. 15). Some choices produce "natural looking" schemas while other choices may lead to bizarre results (see Fig. 16). Also, the dependencies in the output schema can depend idiosyncratically on the input and the algorithm (see Fig. 17).

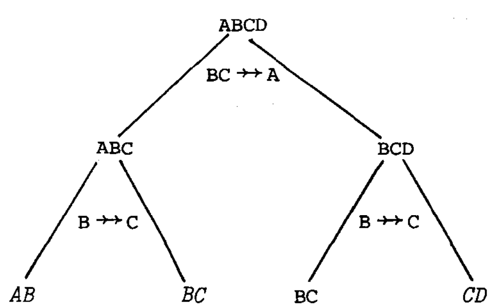
Notice in Fig. 17 that S_D^1 Rep3-represents S_ϕ^1, S_D^2 Rep3-represents S_ϕ^2, and S_ϕ^1 and S_ϕ^2 both Rep3-represent a third schema $S_\phi = \{\underline{U} = <T, (F^1+F^2)^+$. Nonetheless, S_D^1 and S_D^2 have substantially different sets of legal instances. Heuristics for choosing "good" decompositions are suggested in [41] but no rules are known to work in all cases.

FIGURE 15. Algorithm D does not achieve minimal redundancy (italicized attributes become relations schemes).

$S_\phi = \{\underline{U} = <T = \{A,B,C,D\},\ \Gamma = \{B \twoheadrightarrow C; D \twoheadrightarrow B; BC \twoheadrightarrow A\}>\}$



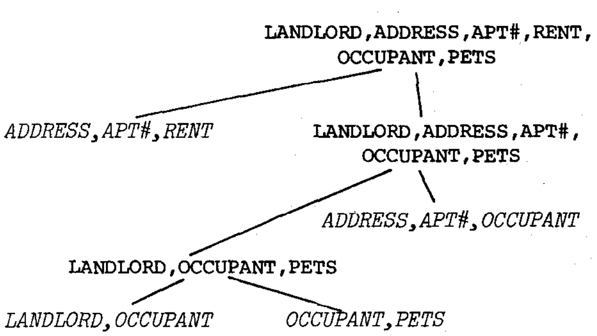
(a)



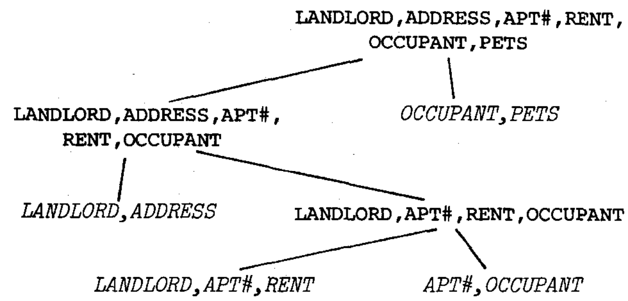
(b)

FIGURE 16. Natural and unnatural 4NF schemas produced by decomposition.

$S = <T = \{LANDLORD, ADDRESS, APT\#, RENT, OCCUPANT, PETS\}$
$= \{LANDLORD \twoheadrightarrow ADDRESS; ADDRESS, APT\# \rightarrow RENT;$
$OCCUPANT \rightarrow ADDRESS, APT\#; OCCUPANT \twoheadrightarrow PETS\}>$



(a) S_D^1



(b) S_D^2

FIGURE 17. Idiosyncratic behavior of decomposition

$S_\phi^1 = \{\underline{U}^1 = <T = \{ABCD\}, F^1 = \{A \rightarrow B; A \rightarrow C; A \rightarrow D, B \rightarrow C, B \rightarrow D, D \rightarrow C\},$

$S_\phi^2 = \{\underline{U}^2 = <T = \{ABCD\}, F^2 = \{A \rightarrow BCD; B \rightarrow CD; C \rightarrow D\}.$

(Note: $(F^1)^+ = (F^2)^+$).

$$S_D^1 = \{\underline{R}_1^1 = <T_1 = \{AB\},\ F_1^1 = \{A \rightarrow B\}>$$
$$\underline{R}_2^1 = <T_2 = \{BC\},\ F_2^1 = \{B \rightarrow C\}>$$
$$\underline{R}_3^1 = <T_3 = \{CD\},\ F_3^1 = \{C \rightarrow D\}>\}$$

$$S_D^2 = \{\underline{R}_1^1 = <T_1 = \{AB\},\ F_1^2 = \{\}>$$
$$\underline{R}_2^2 = <T_2 = \{BC\},\ F_2^2 = \emptyset>\{\}>$$
$$\underline{R}_3^2 = <T_3 = \{CD\},\ F_3^2 = \{C \rightarrow D\}>\}\ .$$

8. HISTORY, CONCLUSIONS, AND FUTURE WORK

The history of database normalization theory begins with Codd's early work [14]. Codd introduced the notion of FD, but did not formalize it. The first mathematizations of FDs were by Delobel [17], Rissanen and Delobel [33], and Delobel and Casey [20]; these authors concentrated on formal properties of dependencies and their relationship to the decomposition approach. They were followed by Armstrong [3] who introduced the notion of completeness of inference rules and proved the completeness of a set of rules for FDs. This work laid the groundwork for the formal theory that has developed since. The earliest synthesis algorithm was an informal one described by Wang and Wedekind [38]. Bernstein [17] followed with a synthesis algorithm that used Armstrong's theory to prove properties of synthesized schemas. Bernstein's algorithm was the first to use a formal definition of representation. This algorithm was subsequently enhanced by Bernstein and Beeri [5,8] who improved its running time.

The first generalization of FDs was the concept of first order hierarchical decomposition by Delobel [18] and Delobel and Leonard [21]. The related concept of MVD was introduced by Fagin [23] and Zaniolo [41], and 4NF was introduced by Fagin

[23]. Completeness of inference rules for MVDs is treated by Beeri, Fagin and Howard [6] and Mendelzohn [29], and algorithmic questions about MVDs by Beeri [4]. Recently, attention has been directed to the representation principle by the work of Aho, Beeri, and Ullman [1] and Rissanen [31].

These references are a mere sketch of the history of normalization theory; a more complete bibliography follows.

A variety of important results appear in these papers, but the lack of uniform definitions has obscured the relationships among many works. We hope the paper will clear up some of the confusion by comparing the major definitions and outlining a general framework in which all can be embedded.

Our main theme is that schema design is directed by the three principles of representation, separation, and minimal redundancy. A goal of research in schema design is to develop a design methodology that satisfies these three principles. Specific formulations of the principles depend upon the types of constraints involved, so a thorough understanding of the formal properties of FDs and MVDs is a prerequisite for achieving this goal.

Many questions still remain unanswered. We list four important areas where more work is needed:

1. *Other dependency structures*--An MVD can hold in a projection of a relation, although it does not hold in the entire relation [19,25]. These *embedded* MVDs (abbr. EMVD) may appear when decomposing a relation scheme into smaller schemes. While some inference rules for EMVDs have appeared, a complete set is not currently known [19].

MVDs characterize lossless joins between two relations. Dependency structures that characterize lossless joins among N relations have recently been suggested, and should be integrated into the theory [30,32]. In addition, the concept of representation (particularly Rep2, Rep4) has only been developed for FDs. Representation questions about MVDs and other dependency structures are open.

2. *Semantic operations on dependencies*--Dependency structures can be used to guide correct retrievals given only minimal logical access path information [11,34]. However, the influence of dependency structures on data operations and the constraints that hold in a relation constructed by operations are only known for special cases.

3. *Universal relation assumption*--This assumption simplifies many theoretical problems but apparently does not hold in practice. It should either be abandoned or adapted for practical situations in some way.

4. *Design tools*--Mechanical procedures must be developed to assist the database designer. A schema synthesis algorithm that takes FDs *and* MVDs as input could be one such design aid. Mechanical mappings from high level data descriptions (e.g., [36]) into dependency structures are also needed. The true test of the theory is demonstrating its effectiveness in solving day to day database design problems. On this metric the theory will live or die.

REFERENCES

1. A.V. Aho, C. Beeri, and J.D. Ullmann, "The Theory of Joins in Relational Databases," Proc. 18th IEEE Symp. on Foundations of Computer Science, Oct. 1977.

2. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

3. W.W. Armstrong, "Dependency Structures of Database Relationships," Proc. IFIP 74, North Holland, 1974, pp. 580-583.

4. C. Beeri, "On the Membership Problem for Multivalued Dependencies in Relational Databases," TR-229, Dept. of Elec. Eng. and Comp. Science, Princeton Univ., Princeton, N.J., Sept. 1977.

5. Beeri, C. and P.A. Bernstein, "Computational Problems Regarding the Design of Normal Form Relational Schemas," *ACM Trans. on Database Sys.*, to appear.

6. C. Beeri, R. Fagin, and J.H. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies," Proc. ACM-SIGMOD Conf., Toronto, Aug. 1977, pp. 47-61.

7. P.A. Bernstein, "Synthesizing Third Normal Form Relations from Functional Dependencies," *ACM Trans. on Database Sys.*, Vol. 1, No. 4 (Dec. 1976), pp. 277-298.

8. P.A. Bernstein and C. Beeri, "An Algorithmic Approach to Normalization of Relational Database Schemas," TR CSRG-73, Computer Systems Research Group, Univ. of Toronto, Sept. 1976.

9. P.A. Bernstein, J.R. Swenson, and D.C. Tsichritzis, "A Unified Approach to Functional Dependencies and Relations," Proc. ACM-SIGMOD Conf., San Jose, Cal., 1975, pp. 237-245.

10. J-M. Cadiou, "On Semantic Issues in the Relational Model of Data," Proc. Intern. Symp. on Math. Foundations of Comp. Science, Gdansk, Poland, Sept. 1975, Springer-Verlag Lecture Notes in Computer Science.

11. Carlson, C.R. and R.S. Kaplan, "A Generalized Access Path Model and its Application to a Relational Database Systems," Proc. 1976 ACM-SIGMOD Conf., ACM, N.Y., pp. 143-156.

12. P.P-S. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Trans. on Database Sys.*, Vol. 1, No. 1 (Sept. 1976), pp. 9-36.

13. E.F. Codd, "A Relational Model for Large Shared Data Bases," *CACM*, Vol. 13, No. 6 (June, 1970), pp. 377-387.

14. E.F. Codd, "Further Normalization of the Data Base Relational Model," in *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 33-64.

15. E.F. Codd, "Recent Investigations in Relational Data Base Systems," Proc. IFIP 74, North-Holland, 1974, pp. 1017-1021.

16. C.J. Date, *An Introduction to Database Systems* (2nd ed.), Addison-Wesley, Reading, MA, 1977.

17. C. Delobel, "A Theory About Data in an Information Systems," IBM Res. Rep. RJ964, Jan. 1972.

18. C. Delobel, "Contributions Théoretiques à la Conception d'un Système d'Informations," Ph.D. Thesis, Univ. of Grenoble, Oct. 1973.

19. C. Delobel, "Semantics of Relations and Decomposition Process in the Relational Data Model," Computer Laboratory, Univ. of Grenoble, 1977.

20. C. Delobel and R.C. Casey, "Decomposition of a Data Base and the Theory of Boolean Switching Functions," *IBM J. of Res. and Dev.*, 17:5 (Sept. 1972), pp. 370-386.

21. C. Delobel and M. Leonard, "The Decomposition Process in a Relational Model," Int. Workshop on Data Structures, IRIA, Namur (Belgium), May 1974.

22. R. Fadous and J. Forsythe, "Finding Candidate Keys for Relational Data Bases," Proc. 1st ACM-SIGMOD Conf., pp. 203-210, San Jose, Cal., 1975.

23. R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM Trans. on Database Sys.*, Vol. 2, No. 3 (Sept. 1977), pp. 262-278.

24. R. Fagin, "The Decomposition Versus the Synthetic Approach to Relational Database Design," Proc. 3rd VLDB Conf., Tokyo, Oct. 1977, pp. 441-446.

25. R. Fagin, "Functional Dependencies in a Relational Database and Propositonal Logic," *IBM J. of Res. and Dev.*, Vol. 21, No. 6, (Nov. 1977), pp. 534-544.

26. M.M. Hammer and D.J. McLeod, "Semantic Integrity in a Relational Database System," Int. Conf. on Very Large Data Bases, ACM, N.Y., pp. 25-47, 1975.

27. W. Kent, "A Primer of Normal Forms," IBM Sys. Dev. Div., TR02.600, San Jose, Cal., 1973.

28. C.L. Lucchesi and S.L. Osborn, "Candidate Keys for Relations," Tech. Rep. Univ. of Waterloo, Waterloo, Ontario, Canada, 1976.

29. A. O. Mendelzon, "On Axiomatizing Multivalued Dependencies in Relational Databases," Dept. of Electr. Eng. and Computer Science, Princeton Univ., Princeton, N.J., July 1977.

30. J.M. Nicolas, "Mutual Dependencies and Some Results on Undecomposable Relations," ONERA-CERT, Toulouse, France, Feb. 1978.

31. J. Rissanen, "Independent Components of Relations," *ACM Trans. on Database Sys.*, Vol. 2, No. 4 (Dec. 1977), pp. 317-325.

32. J. Rissanen, "Theory of Relations for Databases--A Tutorial Survey," IBM Research Lab., San Jose, Cal., Apr. 1978.

33. J. Rissanen and C. Delobel, "Decomposition of Files--A Basis for Data Storage and Retrieval," IBM Res. Rep. RJ1220, San Jose, Cal., May 1973.

34. K.L. Schenk and J.R. Pinkert, "An Algorithm for Servicing Multi-Relational Queries," Proc. 1977 ACM-SIGMOD Conf., ACM, N.Y., pp. 10-19.

35. H.A. Schmid and J.R. Swenson, "On the Semantics of the Relational Data Model," Proc. 1975 ACM-SIGMOD Conf., San Jose, Cal., pp. 211-223.

36. J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. on Database Sys.*, Vol. 2, No. 2 (June 1977), pp. 105-133.

37. Y. Tanaka and T. Tsuda, "Decomposition and Composition of a Relational Database," Proc. 3rd VLDB Conf., Tokyo, Oct. 1977, pp. 454-461.

38. C.P. Wang and H.H. Wedekind, "Segment Synthesis in Logical Data Base Design," *IBM J. of Res. and Dev.*, Vol. 19, No. 1 (Jan. 1975), pp. 71-77.

39. H.K.T. Wong and J. Mylopoulos, "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Data Management," Tech. Rep., Computer Science Dept., Univ. of Toronto, 1977.

40. C.T. Yu and D.T. Johnson, "On the Complexity of Finding the Set of Candidate Keys for a Given Set of Functional Dependencies," *Information Processing Letters*, Vol. 5, No. 4 (oct. 1976), pp. 100-101.

41. C. Zaniolo, "Analysis and Design of Relational Scemata for Database Systems," Tech. Rep. UCLA-ENG-7769, Dept. of Computer Science, UCLA, July 1976.

Acknowledgments

We gratefully acknowledge the assistance of Renate D'Arcangelo for her expert preparation of this manuscript.