CHAPTER 5

FUNDAMENTALS OF DEPENDENCY THEORY

Moshe Y. Vardi†
IBM Almaden Research Center

5.1 Introduction

A *database management system* (DBMS) is essentially a computer system containing a large permanent store of data, with special routines and procedures for handling that data. As such systems evolved over the years, certain ideas emerged as the key features of DBMSs: *integration*, *views*, and *data independence*.

Integration means that a DBMS enables many applications to share data in common, thus sharing costs and eliminating redundancy. Integration does not mean that all applications have to adopt the same *view* of the data. Rather, every application can have its own view, thus "seeing" only the data that is relevant to it. Finally, *data independence* means that the application programs are shielded from the details of the physical organization of the data, and they need not be affected by a physical reorganization of the data.

The main problem with DBMSs that were developed in the 1960s was that they did not quite achieve data independence. While application programs did not have to deal with details of physical storage, they were still dependent on the ordering in which data was stored, on the way in which it was indexed, and on the access paths that were *a priori* chosen by the database designer.

The *relational model* was introduced by Codd [Co70] with the goal of achieving a greater degree of data independence. It is based on two fundamental ideas:

(1) All information in a relational database is represented as data values in relations (tables).

(2) No information is represented by the ordering among columns or tuples (rows) in the relations.

---

† Address: IBM Almaden Research Center K53-802, 650 Harry Rd., San Jose, CA 95120-6099, USA.

Thus, one can say that the relational model of data is almost devoid of semantics. A tuple in a relation represents a relationship between certain values, but it does not represent any information about the nature of this relationship, such as whether it is a one-to-one or a one-to-many relationship.

One approach to remedy this deficiency is to devise means to specify the missing semantics. These semantics specifications are called *semantic* or *integrity constraints*, since they specify which databases are meaningful for the application and which are meaningless. Of particular interest are the constraints called *data dependencies*, or dependencies for short.

Dependency theory, the theory of data dependencies in the relational model, has been an active area of research in the last decade. This article is meant to serve as an introduction to this area. Rather than describe the historical development of the theory, we shall describe it with the wisdom of hindsight. More comprehensive surveys can be found in the books [Ma83,Ul82] and in [FV86].

5.2 Basic Definitions

5.2.1 The Relational Model

We now describe the mathematical model that aims at capturing our intuition about tables where both columns and tuples are essentially unordered. The first thing that we model is the fact that columns in a table have names; these are the column headings. In dependency theory they are called *attributes*. We assume that attributes are symbols taken from a given finite set $U=\{A_1, \ldots, A_n\}$. Following customary notation in dependency theory, we use the letters $A,B,C,\ldots,H$ to denote attributes, and we use $R,S,\ldots,X,Y,Z$ to denote attribute sets. We usually do not distinguish between the attribute A and the attribute set $\{A\}$. The union of X and Y is denoted by XY. Thus, ABD denotes the set $\{A,B,D\}$. The complement of X in U is denoted by $\bar{X}$.

Underlying each column in a table there is a domain of values, out of which the entries in that column are taken. Thus, with each attribute A we associate a set $DOM(A)$, called the *domain* of A. For simplicity we assume that $DOM(A)$ is an infinite set. (This assumption is not always realistic. Consider, for example, the attribute *SEX*.) We denote by *Dom* the set $\bigcup_{A\in U} DOM(A)$.

A tuple spanning a set of columns is essentially an assignment of an entry to each column, where the entries are taken from the corresponding domains. Thus, a *tuple* on an attribute set X is a mapping $u:X\rightarrow Dom$, such that $u(A)\in DOM(A)$ for all $A\in X$. A *relation* on X is a set of tuples on X. (We do not yet require that this set be finite, even though this seems to be a very reasonable requirement, but we allow it to be either finite or infinite. We shall come back to this point later.) If I is a relation on X, then X is

called the *scheme* of I.

Here are some more conventions. We denote tuples by the letters $u,v,\cdots$, and we denote relations by the letters $I,J,\cdots$. Unless explicitly stated otherwise, a tuple is a tuple on U and a relation is a relation on U.

Example 5.1
Consider the following table:

EMP	DEPT	MGR
Hilbert	Math	Gauss
Pythagoras	Math	Gauss
Turing	Computer Science	von Neumann

This is a relation on the attribute set {EMP,DEPT,MGR}. The domain of EMP is the set of employees, the domain of DEPT is the set of departments, and the domain of MGR is the set of managers. The relation consists of three tuples, each one being a mapping from the set of attributes to their associated domains. For example, the first tuple is a mapping u such that u(EMP)=Hilbert, u(DEPT)=Math, and u(MGR)=Gauss. ■

5.2.2 Operations on Relations

We now describe two operations of Codd's *relational algebra* [Co72b]. The *projection* operation is a unary operations on relations whereby a subset of the columns is selected. That is, if I is a relation on X and Y is a subset of X, then the projection of I on Y, denoted $\pi_Y(I)$, is obtained by deleting the columns corresponding to attributes in $Y-X$ and eliminating duplicate tuples in the result.

We now give a formal definition. For a tuple w on X and $Y\subseteq X$, we denote by $w[Y]$ the restriction of w to Y. Thus, $w[Y]$ is a tuple on Y. (Note that for tuple w on X and an attribute $A\in X$, $w[A]$ is a tuple on A, while $w(A)$ is an element in $DOM(A)$. We will not distinguish, however, between the two.) Let I be a relation on X and $Y\subseteq X$. Then $\pi_Y(I)=\{w[Y] : w\in I\}$. Thus $\pi_Y(I)$ is a relation on Y.

Example 5.2 Let I be the relation in Example 5.1. $\pi_{\{DEPT,MGR\}}(I)$ is the relation:

DEPT	MGR
Math	Gauss
Computer Science	von Neumann

$\pi_{\{EMP,DEPT\}}(I)$ is the relation:

EMP	DEPT
Hilbert	Math
Pythagoras	Math
Turing	Computer Science

■

The *join* operation can be viewed as the dual of the projection operation. It combines together tuples from several relations when these tuple agree on the columns that they have in common. Formally, if $I_1, \ldots, I_k$ are relations on $X_1, \ldots, X_k$, respectively, then their join, denoted $I_1 * \cdots * I_k$ (or $\underset{k}{*} I_j$), is a relation on $X = \bigcup_{j=1}^{k} X_j$ defined by

$$\underset{j=1}{\overset{k}{*}} I_j = \{w \text{ is a tuple on } X : w[X_j] \in I_j, \text{ for } 1 \leq j \leq k\}.$$

Note that the join is commutative and associative.

Example 5.3 Consider the two relations of Example 5.2. Their join is the relation of Example 5.1. ■

The reader may wonder whether the projection and the join are inverse to each other. The following examples shows that this is not the case.

Example 5.4 Let I be the relation

A	B	C
0	0	0
1	0	1

Then $\pi_{AB}(I) * \pi_{BC}(I)$ is the relation

A	B	C
0	0	0
1	0	1
0	0	1
1	0	0

■

Example 5.5
Let I be the relation

A	B
0	0
0	1

Let J be the relation

B	C
0	0
0	1

Then $\pi_{AB}(I*J)$ is the relation

A	B
0	0

and $\pi_{BC}(I*J)$ is the relation

B	C
0	0
0	1

■

The following lemma asserts that the above examples are typical.

Lemma 5.1

(1) Let I be a relation on X, and let $X_1, \ldots, X_m$ be attribute sets such that $X = \bigcup_{j=1}^{m} X_j$. Then $I \subseteq *_{j=1}^{m} \pi_{X_j}(I)$.

(2) Let $I_1, \ldots, I_m$ be relations on $X_1, \ldots, X_m$, respectively. Then $\pi_{X_j}(*_{k=1}^{k} I_k) \subseteq I_j$. ■

The simultaneous projection of I on $X_1, \ldots, X_m$ (as in Lemma 5.1(1)) is called a ***decomposition***. If we are lucky to have that $I = *_{j=1}^{m} \pi_{X_j}(I)$, then we say that the decomposition of I to $\pi_{X_1}(I), \ldots, \pi_{X_m}(I)$ is *lossless,* since these relations can be joined to recover the original relation with no loss of information. If $I \subset *_{j=1}^{m} \pi_{X_j}(I)$, then we say that the decomposition is *lossy* we use $\subseteq$ to denote containment, and we use $\subset$

to denote *proper* containment).

5.3 Functional Dependencies

5.3.1 Motivation

When designing a relational database, we are often faced with a choice between alternative sets of relation scheme. Some choices are better than others for various reasons. Consider, for example, the database in Example 5.1. This database suffers from several problems, called *anomalies* by Codd [Co72a]:

(1) *Redundancy*. The information that Gauss is the manager of the Mathematics Department is repeated more than once.

(2) *Possible inconsistency*. Suppose now that we are asked to change Hilbert's manager from Gauss to von Neuman. Then we are faced with the situation that the Mathematics Department has two managers. It is doubtful whether that was the intention of the change.

Indeed, it seems that a better database design would be to store the data in relations as in Example 5.2, which does not suffer from the above anomalies.

A close look at the problem suggests that the source of the problem is that the first design ignores certain dependencies in the data, e.g., the fact that (as we suppose) every department has a unique manager. In other words, there is a *functional dependency* between DEPT and MGR. We start now with the study of functional dependencies, and shall come back later to the issue of database design.

5.3.2 Functional Dependencies

A *functional dependency* (abbr. fd) is a statement that describes a semantic constraint on data [Co72a]. Formally, an fd is an expression of the form $X \rightarrow Y$, read X *functionally determines* Y, where X and Y are attribute sets. $X \rightarrow Y$ is *over* an attribute set R if $XY \subseteq R$. $X \rightarrow Y$ is *satisfied* by a relation I on R if $X \rightarrow Y$ is over R and for all tuples $u, v \in I$, if $u[X] = v[X]$, then $u[Y] = v[Y]$. Intuitively, I satisfies $X \rightarrow Y$ if $I[XY]$ can be viewed as a function from tuples on X to tuples on Y. I satisfies a set Σ of fd's if I satisfies all fd's in Σ. A set Σ of fd's is said to be *over* an attribute set R if all fd's in Σ are over R, e.g., the set $\{A \rightarrow B, BC \rightarrow D\}$ is over the set $ABCDE$.

Example 5.6. Consider again the relation:

EMP	DEPT	MGR
Hilbert	Math	Gauss
Pythagoras	Math	Gauss
Turing	Computer Science	von Neumann

It is easy to verify that this relation satisfies the fd's EMP→DEPT, DEPT→MGR, and MGR→DEPT. It does not, however, satisfy the fd DEPT→EMP. ■

A set of fd's can be viewed as a semantic specification for the database. That is, if we are given a set Σ of fd's, then only relations that satisfy all fd's in Σ are considered "meaningful." Relations that do not satisfy some fd in Σ cannot be actual representation of the data for the application in mind. In order to allow for symbolic manipulation of semantic specification, it stands to reason that we should be able to test mechanically for *equivalence* and *redundancy*. Two sets Δ and Σ of fd's are *equivalent* if they are satisfied by the same relations, i.e., if I satisfies Δ if and only if I satisfies Σ, for all relations I. In that case we also say that Δ is a *cover* for Σ. A set Σ of fd's is *redundant* if there is a proper subet Δ of Σ, i.e, $\Delta \subset \Sigma$, such that Δ and Σ are equivalent.

It is easy to see that both equivalence and redundancy reduce to a more basic notion, that of *implication*. A set Σ of fd's *implies* an fd σ, denoted $\Sigma \models \sigma$, if I satisfies σ whenever I satisfies Σ, for all relations I. Clearly, Σ is redundant if and only if there is some fd $\sigma \in \Sigma$ such that $\Sigma - \{\sigma\} \models \sigma$. Also, Δ and Σ are equivalent if and only if $\Delta \models \sigma$ for all $\sigma \in \Sigma$ and $\Sigma \models \delta$ for all $\delta \in \Delta$.

The relevance of implication to database theory became apparent in Bernstein's work on database design [Ber75,Ber76], and was confirmed in later works, e.g., [BMSU81,Ri77]. Today implication is considered to be perhaps the most fundamental notion in dependency theory.

So far we have allowed both finite and infinite relations. We can try to be more realistic and consider only finite relations. Thus, we say that Σ *finitely implies* σ, denoted $\Sigma \models_f \sigma$, if I satisfies σ whenever I satisfies Σ for all *finite* relations I. Clearly, if $\Sigma \models \sigma$, then also $\Sigma \models_f \sigma$, but the reverse entailment does not seem to hold *a priori*. That is, it is conceivable that $\Sigma \models_f \sigma$, but there is some *infinite* relation I such that $I \models \Sigma$ and $I \not\models \sigma$.

As we said, we are interested in automatic manipulation of fd's. So we would like to solve the following decision problems. The *implication problem* is to decide, given a set Σ of fd's and an fd σ, whether $\Sigma \models \sigma$. The *finite implication problem* is to decide, given a set Σ, of fd's and an fd σ, whether $\Sigma \models_f \sigma$. Note that these are two independent decision problems. That is, a solution to any one of them does not solve the other one.

The reader may ask why we bother to deal with unrestricted implication at all, since finite implication seems to be the more interesting notion. The answer is that the relationship between implication and finite implication is a very significant one, as we shall see later.

5.3.3 Functional Dependencies and First-Order Logic

It is not hard to see that, as was observed by Nicolas [Ni78], fd's can be represented as sentences in first-order logic. There is a minor difficulty, since our definition of relations is different from the way relations are usually defined. For the sake of this translation we assume that the attributes are ordered. Thus, if $R=\{A_1, \ldots, A_k\}$, then rather than view a relation on R as a set of mappings from R to Dom, we can view it as a subset of $DOM(A_1)\times \cdots \times DOM(A_k)$. For example, if $R=ABCD$, then we can assume that the attributes A, B, C, and D label the first, second, third, and fourth columns, correspondingly.

Consider now the fd $AB\rightarrow C$. We can express it by the first-order sentence

$$(\forall abc_1c_2d_1d_2)((\mathbf{R}abc_1d_1\wedge\mathbf{R}abc_2d_2)\supset c_1=c_2)).$$

Here $(\forall abc_1c_2d_1d_2)$ is shorthand for $(\forall a\forall b\forall c_1\forall c_2\forall d_1\forall d_2)$, i.e., all variables are universally quantified. $\mathbf{R}$ is the predicate symbol referring to the relevant relation. (Note that we have used here individual variables as in [BV81,Fa82] rather than tuple variables as in [Ni78].) It is easy to see that we can express arbitrary fd's in this manner.

Since fd's are first-order sentences, we can apply certain operations to them. For example, if σ and τ are fd's, then we can consider $\sigma\wedge\tau$. Now $\sigma\wedge\tau$ is not an fd, but it is a perfectly legitimate first-order sentence. Similarly, we can consider $\neg\sigma$. That means that the implication and the finite implication problems can be reduced to classical decision problems for first-order logic. Let $\Sigma=\{\sigma_1, \ldots, \sigma_k\}$. Then $\Sigma\models\sigma$ if and only if $\sigma_1\wedge\cdots\wedge\sigma_k\wedge\neg\sigma$ is *unsatisfiable*, and $\Sigma\models_f\sigma$ if and only if $\sigma_1\wedge\cdots\wedge\sigma_k\wedge\neg\sigma$ is *finitely unsatisfiable*. (A sentence is (finitely) satisfiable if it has a (finite) model. It is (finitely) unsatisfiable if it has no (finite) model.)

By Godel's Completeness Theorem, unsatisfiability is *partially decidable*. That is, there is a procedure $\mathbf{P}_1$ with the following property. When given an unsatisfiable sentence τ, $\mathbf{P}_1$ will terminate and tell us that τ is unsatisfiable. When given a satisfiable sentence τ, $\mathbf{P}_1$ will either terminate and tell us that τ is satisfiable or it will not terminate. Since implication reduces to unsatisfiability, it follows that the implication problem is partially decidable.

On the other hand, it is easy to see that finite satisfiability and, therefore, finite *nonimplication* are also partially decidable. To find out whether τ is finitely satisfiable we just have to enumerate all finite structures and check whether they happen to be models of τ. Since the collection of finite models (up to

isomorphism) is clearly enumerable, if τ is finitely satisfiable, then our procedure will terminate and will tell us that τ is finitely satisfiable. If τ is not finitely satisfiable, then our procedure will not terminate. Let us call this procedure P_2.

Suppose now that satisfiability and finite satisfiability coincide, that is, whenever there is a model, then there is also a finite model. Then both satisfiability and nonsatisfiability are partially decidable. But then it follows that they are actually decidable [Ro67]. Given a sentence τ, we simply run both P_1 and P_2 in parallel. Now, one of these procedures is guaranteed to terminate with an answer, and of course we can never get contradictory answers. So we have a decision procedure for satisfiability. We know very well, however, that satisfiability of first-order logic is undecidable, which means that satisfiability and finite satisfiability do not coincide. But for many classes of first-order sentences satisfiability and finite satisfiability do coincide, and for such classes satisfiability is decidable. Indeed, the standard technique for proving decidability for classes of first-order sentences is by proving that satisfiability and finite satisfiability coincide [DG79]. Similarly, if we could prove that implication and finite implication coincide for fd's, then it would follows that the implication problem is decidable (and of course so also would be the finite implication problem, since it would be equivalent to the implication problem).

Let us see now what kind of first-order sentences we get when we reduce (finite) implication of fd's to first-order (finite) unsatisfiability. As we saw before, fd's are essentially universal sentences. When we reduce (finite) implication to (finite) unsatisfiability, we take the conjunction of several fd's with the negation of an fd. The resulting sentence $\sigma_1 \wedge \cdots \wedge \sigma_k \wedge \neg\sigma$ can be written as an $\forall^* \exists^*$ sentence, that is, a sentence in prenex normal form whose quantifier prefix consists of a string of universal quantifiers followed by a string of existential quantifiers. The class of such sentences is known as the *initially extended Bernayes-Schonfinkel class*. We abbreviate this long name and call this class the BS class. The BS class has been studied by logicians in the 1920s, and they have shown that satisfiability and finite satisfiability for this class coincide [DG79]. Thus, we have proven the following theorem.

Theorem 5.1 For fd's, implication and finite implication coincide, and the corresponding decision problems (which also coincide) are decidable. ■

This was the good news. The bad news is that though the decision problem for the BS class is decidable, it is highly intractable; even nondeterministic algorithms for this problem require exponential time [Le80]. Thus, reducing implication of fd's to unsatisfiability of BS sentences does not give us a *practical* algorithm. Since the sentences that arise from the reduction of implication to unsatisfiability form a proper subclass of the BS class, we still can hope to find a better algorithm for the implication problem for fd's. To develop such an algorithm we have to study the properties of fd's in more detail.

5.3.4 Formal System for Functional Dependencies

We now try to gain better understanding of fd implication by studying *formal systems*. Formal systems consists of *axiom schemes* and *inference rules* (axiom schemes can be viewed as inference rules with no premises). Given a formal system **F**, a *derivation* of a dependency σ from a set Σ of fd's is a sequence $\sigma_1, \ldots, \sigma_n$, where σ_n is σ and each σ_i is either an instance of an axiom scheme or follows from preceding dependencies in the sequence by one of the inference rules. $\Sigma \vdash \sigma$ denotes that there is a derivation of σ from Σ. We say that **F** is *sound* if $\Sigma \vdash \sigma$ entails $\Sigma \models \sigma$, and it is *complete* if $\Sigma \models \sigma$ entails $\Sigma \vdash \sigma$. (Note that we are now talking solely about unrestricted implication, since we know that implication and finite implication coincide for fd's.)

Since fd's can be expressed as first-order sentences, it might be argued that there is no need to develop formal systems for fd's, because any formal system for first-order logic will do. However, fd's are just a fragment of first-order logic, a fragment that seems to be suitable to expressing integrity constraints of databases. As we are trying here to gain better understanding into implication of fd's, we would like to have a formal system which would enable us to infer only fd's and not general first-order sentences, unlike a formal system for first-order logic.

Formal systems for implication of fd's were first studied by Armstrong [Arm74], even though the importance of implication was not yet realized when his paper appeared in 1974. We present here a formal system **FD** that is somewhat different from Armstrong's system. **FD** consists of one axiom scheme and two inference rules:

FD0 (reflexivity axiom): $\vdash X \rightarrow \emptyset$.

FD1 (transitivity): $X \rightarrow Y, Y \rightarrow Z \vdash X \rightarrow Z$.

FD2 (augmentation): $X \rightarrow Y \vdash XZ \rightarrow YZ$.

Note that the above axiom and rules are schemes; any attribute set can be substituted for X, Y, and Z.

Example 5.7 Let $U = ABCDE$, let Σ consists of the fd's $A \rightarrow B$, $A \rightarrow C$, and $BC \rightarrow DE$, and let σ be $A \rightarrow E$. We show that $\Sigma \vdash \sigma$. By **FD2**, we have $A \rightarrow B \vdash A \rightarrow AB$ and $A \rightarrow C \vdash AB \rightarrow BC$. By **FD1**, we have $A \rightarrow AB, AB \rightarrow BC \vdash A \rightarrow BC$, and $A \rightarrow BC, BC \rightarrow DE \vdash A \rightarrow DE$. Now, by **FD0**, we have $\vdash DE \rightarrow E$, so by another application of **FD1**, we get $\Sigma \vdash A \rightarrow E$. ■

Before proving soundness and completeness, we need a technical lemma.

Lemma 5.2 $X \rightarrow Y, X \rightarrow Z \vdash X \rightarrow YZ$.

Proof By **FD2**, $X \rightarrow Y \vdash X \rightarrow XY$, and $X \rightarrow Z \vdash XY \rightarrow YZ$. The claim follows by **FD1**. ■

Theorem 5.2 The system **FD** is sound and complete.

Proof

Soundness: To prove soundness it suffices to show that the axiom and the inference rules are sound. **FD0** is vacuously sound. Let I be a relation, and let $u,v \in I$. Suppose that I satisfies $X \rightarrow Y$ and $Y \rightarrow Z$. If $u[X]=v[X]$, then $u[Y]=v[Y]$, so $u[Z]=v[Z]$. Thus, I satisfies $X \rightarrow Z$, and **FD1** is sound. Finally, suppose that I satisfies $X \rightarrow Y$. If $u[XZ]=v[XZ]$, then $u[X]=v[X]$, so $u[Y]=v[Y]$, and therefore $u[YZ]=v[YZ]$. Thus, I satisfies $XZ \rightarrow YZ$, and **FD2** is sound.

Completeness: We have to show that if $\Sigma \models \sigma$, then $\Sigma \vdash \sigma$. We prove the contrapositive: if $\Sigma \not\vdash \sigma$, then $\Sigma \not\models \sigma$.

Let σ be $X \rightarrow Y$. Define an attribute set $X^+ = \{A : \Sigma \vdash X \rightarrow A\}$. By **FD0** and **FD2**, we have that $\vdash X \rightarrow A$, for $A \in X$, so $X \subseteq X^+$. By Lemma 5.2, we have that $\Sigma \vdash X \rightarrow X^+$. We claim that Y is not a subset of X^+. Suppose it is. Then by **FD0**, $\Sigma \vdash X^+ \rightarrow Y$, and since we know that $\Sigma \vdash X \rightarrow X^+$, it follows, by **FD1**, that $\Sigma \vdash X \rightarrow Y$ - a contradiction. Thus, there must be some attribute B that is a member of Y but not a member of X^+.

We construct a relation I as follows. I consists of two tuples u and v, such that u and v agree precisely on X^+. For example, $u[A]=0$ for each attribute A, $v[A]=0$ for each attribute $A \in X^+$, and $v[A]=1$ for each attribute $A \in U - X^+$. Clearly, $u[A]=v[X]$ but $u[Y] \neq v[Y]$. So I does not satisfy $X \rightarrow Y$.

We now claim that I satisfies Σ. Let $S \rightarrow T$ be an fd in Σ. Suppose that $u[S]=v[S]$. But then we must have that $S \subseteq X^+$. Thus, by **FD2**, we have that $S \rightarrow T \vdash X^+ \rightarrow T$. Now, by **FD0** and **FD2**, we have that $\vdash T \rightarrow A$ for $A \in T$. Therefore, by **FD1**, we have that $\Sigma \vdash X^+ \rightarrow A$, for all attributes $A \in T$. But then $T \subseteq X^+$ and $u[T]=v[T]$.

Thus, I satisfies Σ and it does not satisfy σ, so $\Sigma \not\models \sigma$. ■

Note that the above proof is also a direct proof that for fd's implication and finite implication coincide, since the *counterexample* relation that we have constructed is finite (a counterexample relation for an implication $\Sigma \models \sigma$ is a relation that satisfies Σ but not σ).

5.3.5 Functional Dependencies and Propositional Logic

What is perhaps the most important fact about implication of fd's is buried in the proof of the completeness of the system **FD**. Before stating it, we need some definitions. A *two-tuple relation* is a relation that contains at most two tuples. To make things simpler we assume that a two-tuple relation consists of precisely two, possibly identical, tuples. We say that a set Σ of fd's *implies* an fd σ *with respect to two-tuple relations*, denoted $\Sigma \models_2 \sigma$, if if I satisfies σ whenever I satisfies Σ, for all two-tuple relations I. Clearly, if $\Sigma \models_f \sigma$, then also $\Sigma \models_2 \sigma$. The proof of Theorem 5.2 shows that the reverse entailment also

holds.

Theorem 5.3 Let Σ be a set of fd's, and σ be an fd. Then $\Sigma \models \sigma$ if and only if $\Sigma \models_f \sigma$ if and only of $\Sigma \models_2 \sigma$. ■

The reader may wonder why the sudden interest in two-tuple relations. After all, most real databases usually contain more than just two tuples. Before explaining the importance of Theorem 5.3, we define a new notion of implication, seemingly unrelated to our previous definition. This definition is based on viewing fd's as formulas in propositional logic. According to this view, each attribute can be viewed as a propositional symbol that can be assigned truth values. Thus, we define a *relational truth assignment* ψ as a mapping $\psi: U \rightarrow \{0,1\}$. We extend ψ to give truth assignment to attribute sets by defining $\psi(X) = \prod_{A \in X} \psi(A)$. That is, ψ assigns the value 1 to X if and only if it assigns the value 1 to all attributes in X. We can now extend relational truth assignments to assign truth values to fd's in the following manner: ψ assigns the value 1 to $X \rightarrow Y$ if only if whenever $\psi(X)=1$, then also $\psi(Y)=1$. For example if $\psi(A)=1$, $\psi(B)=1$, and $\psi(C)=0$, then $\psi(A \rightarrow B)=1$ and $\psi(AB \rightarrow C)=0$. We can also extend ψ to sets of fd's by defining $\psi(\Sigma) = \prod_{\sigma \in \Sigma} \psi(\sigma)$. That is, ψ assigns the value 1 to the set Σ if and only if it assigns the value 1 to all fd's in Σ. We say that that a set Σ of fd's *propositionally implies* an fd σ, denoted $\Sigma \models_p \sigma$, if $\psi(\Sigma)=1$ entails $\psi(\sigma)=1$ for any relational truth assignment ψ. This notion of implication for dependencies may seem to be quite unmotivated. Surprisingly enough, propositional implication turns out to coincide with implication.

Theorem 5.4 [Fa77b] Let Σ be a set of fd's, and σ be an fd. Then $\Sigma \models \sigma$ if and only if $\Sigma \models_p \sigma$.

Proof By Theorem 5.3, it suffices to show that $\Sigma \models_2 \sigma$ iff $\Sigma \models_p \sigma$. The basis for this equivalence is a simple correspondence between relational truth assignments and two-tuple relations.

Let ψ be a relational truth assignment. Then $I_\psi = \{u,v\}$ is a two tuple relation such that $u[A]=v[A]$ iff $\psi(A)=1$. Let $I=\{u,v\}$ be a two-tuple relation. Then ψ_I is a relational truth assignment such that $\psi(A)=1$ iff $u[A]=v[A]$. Let τ be any fd. The reader can verify that $\psi(\tau)=1$ iff I_ψ satisfies τ, and I satisfies τ iff $\psi_I(\tau)=1$. Similar correspondence holds for sets of fd's. Suppose now that $\Sigma \models_2 \sigma$. We want to show that $\Sigma \models_p \sigma$. Let ψ be a relational truth assignment such that $\psi(\Sigma)=1$. Then I_ψ satisfies Σ, and since $\Sigma \models_2 \sigma$, it must be the case that I_ψ satisfies σ. But then $\psi(\sigma)=1$. We have shown that $\psi(\Sigma)=1$ entails $\psi(\sigma)=1$ for any relational truth assignment σ, so $\Sigma \models_p \sigma$. An analogous argument shows that if $\Sigma \models_p \sigma$, then also $\Sigma \models_2 \sigma$. ■

The significance of Theorem 5.4 is that it enables us to view fd's as formulas in propositional logic. Namely, the fd $A_1 \cdots A_k \rightarrow B_1 \cdots B_l$ can be viewed as the formula $A_1 \wedge \cdots \wedge A_k \supset B_1 \wedge \cdots \wedge B_l$, where the

A's and the B's are viewed as propositional symbols. Furthermore, such a formula is equivalent to a set of *Horn* formulas, and so is its negation. (Horn formulas are formulas in one of the following forms: $A_1 \wedge \cdots \wedge A_k \rightarrow B$, B, or $\neg A_1 \vee \cdots \vee \neg A_k$, where the A's and B are propositional symbols.) For example, the formula $A \supset BC$ is equivalent to the set $\{A \supset B, A \supset C\}$, and the formula $\neg(A \supset BC)$ is equivalent to the set $\{A, \neg B \vee \neg C\}$. Thus, the conjunction $\sigma_1 \wedge \cdots \wedge \sigma_k \wedge \neg\sigma$ is equivalent to a set of Horn formulas. While no polynomial algorithm is known for satisfiability of propositional logic, for Horn formulas satisfiability can be tested in polynomial time. So, as a consequence of Theorem 5.4, we obtain a polynomial time algorithm for the implication problem for fd's.

Interestingly, a polynomial time algorithm for fd implication was discovered by Beeri and Bernstein [BB79] independently of the discovery of the correspondence between fd's and propositional logic. Their algorithm is based on a fast construction of *closure* sets with respect to a given set of fd's. Formally, the *closure* of an attribute set X with respect to a set Σ of fd's, denoted $cl_\Sigma(X)$, is the set $\{A : \Sigma \models X \rightarrow A\}$ (so the set X^+ in the proof of Theorem 5.2 is exactly $cl_\Sigma(X)$). Note that $X \subseteq cl_\Sigma(X)$. The proof of the following lemma is easy and is left to the reader.

Lemma 5.3 $\Sigma \models X \rightarrow Y$ if and only if $Y \subseteq cl_\Sigma(X)$. ■.

Thus, to test implication of $X \rightarrow Y$ by Σ it suffices to construct $cl_\Sigma(X)$ and then to test whether $Y \subseteq cl_\Sigma(X)$.

Algorithm 5.1

Input: A set Σ of fd's and an attribute set X.

Output: $cl_\Sigma(X)$.

```
CLOSURE(Σ,X)
begin
    Y:=X;
    while there exists an fd S→T in Σ such that S⊆Y and T⊈Y do Y:=YT
    end while
return(Y)
end.
```

We have to show that CLOSURE(Σ,X) terminates and indeed returns $cl_\Sigma(X)$.

Lemma 5.4 CLOSURE(Σ,X) terminates and returns $cl_\Sigma(X)$.

Proof The algorithm clearly terminates. Let Z be the set constructed by the algorithm. We show first that $Z \subseteq cl_\Sigma(X)$ by induction on the steps of the algorithm. That is, we show that $Y \subseteq cl_\Sigma(X)$ at all stages of the algorithm. Originally, the claim is true, since $Y = X$. Suppose now that the claim is true for Y, and that there is an fd $S \rightarrow T$ in Σ such that $S \subseteq Y$. It is easy to see that in this case $\Sigma \models X \rightarrow A$ for all $A \in YT$ (use

the formal system **FD**). Thus, $YT \subseteq cl_\Sigma(X)$.

It remains to show that it is impossible that Z is a proper subset of $cl_\Sigma(X)$. Suppose it is, and let $B \in cl_\Sigma(X) - Z$. Since $B \in cl_\Sigma(X)$, it means that $\Sigma \models X \rightarrow B$. Let $I = \{u,v\}$ be a two-tuple relation such that $u[A] = v[A]$ iff $A \in Z$. Clearly, I does not satisfy $X \rightarrow B$. We claim now that I satisfies Σ. Suppose it does not. Then there must be some fd $S \rightarrow T$ in Σ such that $S \subseteq Z$ but $T \not\subseteq Z$. But in this case the algorithm could not have terminated. Thus, I satisfies Σ, but it does not satisfies $X \rightarrow B$, so $\Sigma \not\models X \rightarrow B$ - a contradiction. It follows that $Z = cl_\Sigma(X)$. ■

The alert reader may have noticed the similarity in the proofs of Theorem 5.2 and Lemma 5.4. Indeed, one can view an execution of the algorithm as an organized derivation in the formal system **FD**.

Example 5.8 Let $U = ABCDEF$, and let Σ consists of the fd's $A \rightarrow B$, $A \rightarrow C$, and $BC \rightarrow DE$. Let us calculate $cl_\Sigma(A)$. Initially $Y = A$. We then apply the fd $A \rightarrow B$, since $A \subseteq Y$, and set $Y = AB$. We then apply the fd $A \rightarrow C$ and set $Y = ABC$. Finally, we apply the fd $BC \rightarrow DE$ and set $Y = ABCDE$. So $cl_\Sigma(A) = ABCDE$. ■

It is not hard to implement CLOSURE to run in polynomial time. Beeri and Bernstein showed how to implement it to run in *linear* time.

Theorem 5.5 [BB79] The implication problem for fd's can be solved in time that is linear in the length of the input. ■

5.3.6 Covers

Now that we can test implication efficiently, we can also test equivalence and redundancy efficiently. In particular, we can find efficiently *nonredundant covers*. (Recall the Δ is a cover for Σ if Δ and Σ are equivalent.)

Algorithm 5.2

Input: A set Σ of fd's.

Output: A nonredundant cover of Σ.

```
NONREDUN(Σ)
begin
    Δ:=Σ;
    for each fd σ in Δ do
        if Δ−{σ}⊨σ then Δ:=Δ−{σ}
    end for
return(Δ)
end.
```

Example 5.9 Let $U=ABC$, and let Σ consist of the the fd's $A \rightarrow B$, $B \rightarrow A$, $B \rightarrow C$, and $A \rightarrow C$. Then NON-REDUN(Σ) returns the set $\{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$. Note that the set returned by the algorithm depends on the order in which Σ is presented. If Σ is presented in the order $\{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C\}$, then NON-REDUN($\Sigma$) returns the set $\{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$. Note that $\{A \rightarrow B, B \rightarrow A, AB \rightarrow C\}$ is also a nonredundant cover of Σ. ■

The above example shows that a given set of fd's can have many nonredundant covers, some of which may not be a subset of the given set. For economical reasons it is desirable to find the cover with the smallest number of fd's. Formally, Δ is a *minimum cover* of Σ if Δ is a cover of Σ and whenever Γ is also a cover of Σ then $|\Delta| \leq |\Gamma|$. Algorithm 5.2 supplies us with nonredundant covers, but these are not necessarily minimum covers.

Theorem 5.6 [Ma80] There is an algorithm for finding minimum covers in time that is quadratic in the length of the input. ■

At this point the reader may suspect that everything we care to know about fd's can be found in polynomial time. Unfortunately, this does not seem to be the case, as is demonstrated by a slight variant of the minimum cover problem, which is to find a *minimum contained cover of* Σ. A cover Δ of Σ is a *contained cover* if $\Delta \subseteq \Sigma$. Δ is a minimum contained cover of Σ if it is a contained cover of Σ, and there is no smaller contained cover of Σ. Formally, the minimum contained cover problem is: given a set Σ of fd's and an integer k, is there a contained cover Δ of Σ such that $|\Delta| \leq k$?

Theorem 5.7 [Ber75] The minimum contained cover problem is NP-complete. ■

NP-complete problems are problems that can be solved in polynomial time using "guesses" (in the minimum cover problem we can guess a subset of Σ and check that it is a cover of Σ and has at most k fd's), but probably cannot be solved in polynomial time by any deterministic algorithm. We assume familiarity with the theory of NP-completeness. A good textbook in the subject is [GJ79].

Notice that if we could actually find minimum contained covers in polynomial time, then we could clearly solve the minimum cover problem in polynomial time. Thus, Theorem 5.7 can be viewed as strong evidence to the intractability of finding minimum contained covers.

5.4 Database Schema Design

5.4.1 Normal Forms

We have argued in §3.1 that certain ways of storing the data might be better than other ways. In the example studied, it is better to store the data in two relations, one on {EMP,DEPT} and one on

{DEPT,MGR}, rather than store it in one relation on {EMP,DEPT,MGR}. The decisions on the organization of the data are taken during the *database design* process. We concentrate here on one aspect of that process, which is the design of the *database schema*. The database schema specifies a list of relation schemes and a set of relations that are meaningful for each relation scheme. The latter is specified by means of dependencies. Formally, a *relation schema* is a pair (R,Σ) where R is a relation scheme, i.e., a set of attributes, and Σ is a set of fd's over R. (Note that we distinguish between a relation *scheme*, which is just an attribute set, and a relation *schema*, which consists also of a set of fd's.) A *database schema* is a collection $\mathbf{D}=((R_1,\Sigma_1), \ldots ,(R_k,\Sigma_k))$ of relation schemas, where $\bigcup_{i=1}^{k} R_i = U$. A database **B** over **D** is an assignment of a relation to each relation schema in **D**, that is, each relation schema (R_i,Σ_i) in **D** is assigned a relation on R_i that satisfies Σ_i.

Example 5.10 A database schema for the database of Example 5.1 is

$$(\{EMP,DEPT,MGR\},\{EMP \rightarrow DEPT, DEPT \rightarrow MGR\})).$$

A database schema for the database of Example 5.2 is

$$(\{EMP,DEPT\},\{EMP \rightarrow DEPT\}),$$

$$(\{DEPT,MGR\},\{DEPT \rightarrow MGR\})).$$

■

The problems described in §3.1 follow from the fact that an fd is more than just a semantic constraint; it is also a description of a basic piece of data. The fd DEPT→MGR says that every department has a unique manager, and it also intuitively says that the relationship between departments and managers is an independent semantic relationship. Without this functional dependency, the relationship between department and managers would have been merely a projection of the relationship between employees, departments, and managers, and there would be no anomalies. Thus, the source of the problem is the embedding of an independent semantic relationship in a bigger context. Note, however, that the mere presence of an fd does not necessarily cause a problem.

Example 5.11 Consider the database schema $(\{EMP,DEPT,SAL\},\{EMP \rightarrow DEPT, EMP \rightarrow SAL\}))$, with the database

EMP	DEPT	SAL
Hilbert	Math	$20000
Pythagoras	Math	$25000
Turing	Computer Science	$40000

For this database there is no problem of redundancy and possible inconsistency, even though the relationship between, say, employees and their departments is embedded in a bigger context, that of {EMP,DEPT,SAL}. The difference between this example and the problematic one is that here the bigger relationship, that of {EMP,DEPT,SAL}, is not independent from the smaller one, that of {EMP,DEPT}, since EMP functionally determines both DEPT and SAL. ■

The above observations on the problem led Codd to the definition of the several *normal forms* [Co72a,Co74]. We jump immediately to the strongest of them, the so-called *Boyce-Codd Normal Form* (BCNF). We first need a few definitions. Let $\mathbf{D}=((R_1,\Sigma_1),\ldots,(R_k,\Sigma_k))$ be a database schema, and let $\Sigma=\bigcup_{i=1}^{k}\Sigma_i$. We say that an attribute set X is a *determinant* of R_i if $X\subseteq R_i$ and there is an attribute $A\in R_i-X$ such that $\Sigma\models X\rightarrow A$. X is a *superkey* of R_i if $X\subseteq R_i$ and $\Sigma\models X\rightarrow R_i$. We say that **D** is in BCNF if whenever X is a determinant of R_i, then X is a superkey of R_i. The intuition behind this definition is that if X is a determinant of R_i but not a superkey of R_i, then we have a semantic relationship embedded within a bigger independent context.

Example 5.12 The database schema

$$(\{\{EMP,DEPT,MGR\},\{EMP\rightarrow DEPT,DEPT\rightarrow MGR\}\})$$

is not in BCNF, since DEPT is a determinant but not a superkey. The database schema

$$(\{\{EMP,DEPT\},\{EMP\rightarrow DEPT\}\},$$

$$\{\{DEPT,MGR\},\{DEPT\rightarrow MGR\}\})$$

is in BCNF. ■

5.4.2 Normalization through Decomposition

The solution suggested in §3.1 to the problem of anomalies was to change the database schema from

$$(\{\{EMP,DEPT,MGR\},\{EMP\rightarrow DEPT,DEPT\rightarrow MGR\}\})$$

to

$$((\{EMP,DEPT\},\{EMP \rightarrow DEPT\}),$$

$$(\{DEPT,MGR\},\{DEPT \rightarrow MGR\})),$$

and at the same time replace the relation on {EMP,DEPT,MGR} by its projections on {EMP,DEPT} and {DEPT,MGR}.

We noted before, however, that a decomposition of a relation into its projection can be lossy. Fortunately, the presence of fd's can guarantee the losslessness of the decomposition.

Theorem 5.8 [He71] Let I be a relation on R, and let X, Y, and Z be attribute sets such that $XYZ=R$. If I satisfies $X \rightarrow Y$, then the decomposition of I into $\pi_{XY}(I)$ and $\pi_{XZ}(I)$ is lossless.

Proof Let $J=\pi_{XY}(I)*\pi_{XZ}(I)$. We know that $I \subseteq J$, so we have to show that $J \subseteq I$. Let $u \in J$. Then $u[XY] \in \pi_{XY}(I)$ and $u[XZ] \in \pi_{XZ}(I)$. Thus, there are tuples $v,w \in I$ such that $u[XY]=v[XY]$ and $u[XZ]=w[XZ]$. Consequently, $v[X]=w[X]$, and since I satisfies $X \rightarrow Y$, it follows that $w[XY]=v[XY]=u[XY]$. Thus $w[XYZ]=u[XYZ]$, so $w=u$, since $XYZ=R$. ■

This suggests the process of *normalization* through *decomposition*. We need first some notation. Let Σ be a set of fd's, and let X be an attribute set. Then $\pi_X(\Sigma)=\{S \rightarrow T : S \rightarrow T \in \Sigma \text{ and } ST \subseteq X\}$. Let $\mathbf{D}=((R_1,\Sigma_1), \ldots ,(R_k,\Sigma_k))$ be database schema that is not in BCNF. Then there is a relation schema (R_i,Σ_i) and an attribute set X such that X is a determinant of R_i but is not a superkey of R_i. That is, there is an attribute $A \in R_i - X$ such that $\Sigma \models X \rightarrow A$, where $\Sigma = \bigcup_{i=1}^{k} \Sigma_i$. We then replace (R_i,Σ_i) by two relation schemas (R_i^1,Σ_i^1) and (R_i^2,Σ_i^2), where $R_i^1=XA$, $R_i^2=R_i-\{A\}$, and $\Sigma_i^j=\pi_{R_i^j}(\Sigma_i)$.

Corresponding to the decomposition of the relation schema (R_i,Σ_i) there is the decomposition of the relation on R_i into relations on R_i^1 and R_i^2. This decomposition is justified because of Theorem 5.8.

Since the process of normalization through decomposition produces smaller and smaller relation schemes it is clear that it must terminate producing a database schema in BCNF.

Example 5.13 [Ul82] Let $U=CTHRSG$, where C stands for "Course," T stands for "Teacher," H stands for "Hour," R stands for "Room," S stands for "Student," and G stands for "Grade." Σ consists of the following fd's:

$C \rightarrow T$ each course has only one teacher,

$HR \rightarrow C$ only one course can meet in a room at one time,

$HT \rightarrow R$ a teacher can be only in one room at one time,

$CS \rightarrow G$ each student has only one grade in each course, and

$HS \rightarrow R$ a student can be in only one room at one time.

We start with the database schema (U,Σ). This schema is not in BCNF, since CS is not a superkey of U, but $CS \rightarrow G$ is in Σ. Thus, we decompose U into CSG and $CTHRS$. The resulting database schema is $((CSG,\Sigma_1),(CTHRS,\Sigma_2))$, where $\Sigma_1 = \pi_{CSG}(\Sigma) = \{CS \rightarrow G\}$, and $\Sigma_2 = \pi_{CTHRS}(\Sigma) = \{C \rightarrow T, HR \rightarrow C, HT \rightarrow R, HS \rightarrow R\}$.

This schema is still not in BCNF, since C is not a superkey of $CTHRS$, but $C \rightarrow T$ is in Σ_2. So we decompose $CTHRS$ into CT and $CHRS$. The resulting database schema is $((CSG,\Sigma_1),(CT,\Sigma_3),(CHRS,\Sigma_4))$, where $\Sigma_3 = \pi_{CT}(\Sigma_2) = \{C \rightarrow T\}$, and $\Sigma_4 = \pi_{CHRS}(\Sigma_2) = \{HR \rightarrow C, HS \rightarrow R\}$.

This schema is still not in BCNF, since HR is not a superkey of $CHRS$, but $HR \rightarrow C$ is in Σ_4. So we decompose $CHRS$ into HRC and HRS. The resulting database schema is

$((CSG,\{CS \rightarrow G\}),$

$(CT,\{C \rightarrow T\}),$

$(HRC,\{HR \rightarrow C\})$

$(HRS,\{HS \rightarrow R\}))$

The four relation schemas of this database schema tabulate respectively:

- grades for students in courses,
- the teacher of each course,
- the hours and rooms where each course meet, and
- the rooms in which students can be found at given hours.

This design is not a very good one. Rather than keep information about the rooms in which students can be found at given hours, it would make more sense to keep the information about the courses that students take at given hours. If we added to Σ the fd $CH \rightarrow R$ (which is redundant, since it follows from the other fd's), then we could have decomposed $CHRS$ into HRC and HSC, yielding a more intuitive database schema. ■

5.4.3 Problems with BCNF

In order to do normalization through decompositions we have to be able to check for violation of BCNF. Unfortunately, checking violations of BCNF is probably an intractable problem.

Theorem 5.9 [BB79] The following problem is NP-complete: determine for a given database schema **D** whether **D** is not in BCNF.

Proof To prove that the problem is NP-complete we first have to show that it is in NP, and then we have to show that it is NP-hard, i.e., we have to reduce a known NP-complete problem to it.

We first show that the problem is in NP, i.e., can be solved in polynomial time using "guesses." Let $\mathbf{D}=((R_1,\Sigma_i), \ldots ,(R_k,\Sigma_k))$, and let $\Sigma=\bigcup_{i=1}^{k}\Sigma_i$. If **D** is not in BCNF, then there is some relation scheme R_i, an attribute set $X\subseteq R_i$, and an attribute $A\in R_i-X$ such that $\Sigma\models X\rightarrow A$ but $\Sigma\not\models X\rightarrow R_i$. To check that **D** is not in BCNF we guess R_i, X, and A, and check that the above conditions are satisfied. Clearly, this can be done in polynomial time.

To show that the problem is NP-hard we reduce to it the *hitting set problem*, which was shown to be NP-complete in [Ka72]. The hitting set problem is formulated as follows. Given a family $B_1, \ldots ,B_m$ of subsets of a set $T=\{A_1, \ldots ,A_n\}$, one has to decide if there exists a set $W\subseteq T$ such that for each i, $1\leq i\leq m$, W and B_i have precisely one element in common. Such a W is called a *hitting set*. We now show a polynomial time algorithm that maps each instance the hitting set problem to a database schema, such that there exists a hitting set if and only if the produced schema is not in BCNF.

We take the set U of attributes to be $\{A_1, \ldots ,A_n,B_1, \ldots ,B_n,C,D\}$. The database schema consists of the following relation schemas. For each A_i and B_j such that $A_i\in B_j$, we have a relation schema $(A_iB_j,\{A_i\rightarrow B_j\})$. We also have a relation schema $(B_1 \cdots B_mC,\{B_1 \cdots B_m\rightarrow C\})$. Finally, we also have a relation schema

$$(A_1 \cdots A_nCD,\{CD\rightarrow A_1 \cdots A_n\}\cup\{A_iA_j\rightarrow CD : i\neq j \text{ and there is some } B_k \text{ such that } A_i\in B_j \text{ and } A_j\in B_k\}).$$

Clearly, we can generate this schema in polynomial time. Let Σ consists of all the fd's in the relation schema.

Let $W\subseteq T$ be a hitting set. Then for each B_j there is some A_i such that $A\in W\cap B_j$. Therefore, $\Sigma\models W\rightarrow B_j$. It follows that $\Sigma\models W\rightarrow C$. That is, W is a determinant of $A_1 \cdots A_nCD$. We claim that $cl_\Sigma(W)=WB_1 \cdots B_mC$, so W is not a superkey of this relation schema. To prove this we have to show that for every fd τ in Σ either the left-hand side of τ is not a subset of $cl_\Sigma(W)$ or the right-hand side of τ is a subset of $cl_\Sigma(W)$. This can be shown by a case analysis.

Conversely, suppose that **D** is not in BCNF. A case analysis shows that the violation can be only in the last relation scheme. Suppose that $W \subseteq A_1 \cdots A_n CD$ is a determinant but not a key. This can happen only if W has precisely one element in common with each of the B_i's. So $W \cap T$ is a hitting set. ■

Theorem 5.9 indicates one problem with normalization through decomposition. There is another problem with normalization, which goes beyond the issue of computational tractability. The basic idea underlying normalization is that of replacing a problematic database schema with a nonproblematic one. But this would clearly be meaningless unless the new schema in some sense represents the old one. This issue, the relationship between database schemas, is a significant issue in database theory, and we shall not go into it here ([BBG78] is a good introduction to that subject). Rather we consider here normalization of database schemas of the form (U,Σ) (i.e., with a single relation schema). Such schemas are called *universal* schemas.

Let $\mathbf{D}=(U,\Sigma)$ be a universal schema, and let $\mathbf{D}'=((R_1,\Sigma_1), \ldots ,(R_k,\Sigma_k))$ be a database schema. We want to find conditions under which it makes sense to say that $\mathbf{D}'$ *represents* $\mathbf{D}$. We have already implicitly referred to one condition: that there is no loss of information in storing the data as a database over $\mathbf{D}'$ rather than a database over $\mathbf{D}$. There is, however, another natural condition: that all the Σ_i's be implied by Σ and that the Σ_i's together imply all the dependencies in Σ. To see why this is necessary, consider the following example.

Example 5.14 Let $U=CAZ$, where C stands for "City," A stands for "Address," and Z stands for "Zipcode." Σ consists of the following dependencies

$CA \rightarrow Z$ city and address together determine the zipcode, and

$Z \rightarrow C$ the zipcode determines the city.

Let $\mathbf{D}=(U,\Sigma)$, and let $\mathbf{D}'=((ZC,\{Z \rightarrow C\}),(ZA,\varnothing))$. Consider the following database **B** on $\mathbf{D}'$:

Z	C
10017	New York
10018	New York

Z	A
10017	33 1st Ave.
10018	33 1st Ave.

Does this database make sense? We claim it does not. Let us see why.

Consider the tuples of the relation on ZA in **B**. This tuples represent data about addresses and their zipcodes. But this data make sense only if with each pair (address, zipcode) we can associate a city. Let c_1 and c_2 be the associated cities. That is, with (10017, 33 1st Ave.) we associate the city c_1, and with (10018, 33 1st Ave.) we associate the city c_2. But we know that the zipcode determines the city, so from the data in the relation on ZC it follows that both c_1 and c_2 must be "New York." But then we get that there are two zipcodes associated with the the same address in New York, which violates the fd $CA \rightarrow Z$.

The problem with **D'** is that it represents the fd $Z \rightarrow C$, but it does not represent the fd $CA \rightarrow Z$. ■

Thus, in order for a database schema **D'** to represent a universal schema, the decomposition associated with **D'** should be lossless, and **D'** should *preserve* all dependencies in the universal schema. We now formalize these conditions. Let $\mathbf{D} = (U, \Sigma)$ be a universal schema, and let $\mathbf{D}' = ((R_1, \Sigma_1), \ldots, (R_k, \Sigma_k))$ be a database schema. We say that **D'** *represents* **D** if the following conditions hold:

(1) For every relation I on U that satisfies Σ, the decomposition of I into $\pi_{R_1}(I), \ldots, \pi_{R_k}(I)$ is lossless.

(2) $\Sigma \models \Sigma'$ and $\Sigma' \models \Sigma$, where $\Sigma' = \bigcup_{i=1}^{k} \Sigma_i$.

Looking again at the decomposition process we see that starting with a database schema $\mathbf{D} = (U, \Sigma)$ it produces a database schema **D'** such that condition (1) and half of condition (2) above are satisfied and **D'** is in BCNF. What we would like is to strengthen that to get a database schema that represents **D** and is in BCNF. Unfortunately, this is impossible.

Theorem 5.10 Let $U = CAZ$, and let $\Sigma = \{CA \rightarrow Z, Z \rightarrow C\}$. There is no database schema in BCNF that represents $\mathbf{D} = (U, \Sigma)$.

Proof We have seen in Example 5.14 that the schema $((ZC, \{Z \rightarrow C\}), (ZA, \varnothing))$ does not represent **D**. An exhaustive analysis shows that no database schema in BCNF represents **D**. ■

Since not every universal database schema can be represented by a database schema in BCNF, this raises a natural decision problem: given a universal database schema (U, Σ), decide if there exists a database schema in BCNF that represents **D**. We call this the *BCNF normalizability problem*.

Theorem 5.11 [BB79] The BCNF normalizability problem is NP-hard. ■

5.4.4 Third Normal Form and Normalization via Synthesis

We saw in the previous section that BCNF is too stringent. Given a database schema $\mathbf{D}=(U,\Sigma)$ it is not always possible to get a database schema in BCNF that represents **D**, and even when it is possible, it might be computationally intractable. One solution to the problem is to consider a somewhat weaker normal form that would be easier to use even though it may not solve all the anomalies. The *Third Normal Form* (3NF) is such a normal form [Co72a]. (Historically, 3NF came before BCNF.)

We need a few definitions. Let $\mathbf{D}=((R_1,\Sigma_1),\ldots,(R_k,\Sigma_k))$ be a database schema, and let $\Sigma=\bigcup_{i=1}^{k}\Sigma_i$. We say that X is a *key* of R_i if X is a superkey of R_i and no proper subset of X is a superkey of R_i. An attribute $A\in R_i$ is *prime* in R_i if there is some key X of R_i such that $A\in X$. X is a *strong determinant* of R_i if $X\subseteq R_i$ and there is a nonprime attribute $A\in R_i-X$ such that $\Sigma\models X\rightarrow A$. We say that **D** is in 3NF if whenever X is a strong determinant of R_i, then X is a superkey of R_i.

Example 5.15 Consider the universal schema $(ABCD,\{AB\rightarrow C\})$. $ABCD$ is a superkey but is not a key. ABD is a key. A, B, and D are prime attributes. AB is a strong determinant but is not a key, so this schema is not in 3NF. Consider now the schema $((ABC,\{AB\rightarrow C\}),(ABD,\varnothing))$. This schema is in 3NF and it represents the previous schema. ■

Example 5.16 Consider the universal schema $(CAZ,\{CA\rightarrow Z,Z\rightarrow C\})$. CA is a key, so C is a prime attribute. Z is a determinant, but it is not a strong determinant. This schema is in 3NF but is not in BCNF. Consider the following database over this schema.

C	A	Z
New York	33 1st Ave.	10017
New York	34 1st Ave.	10017
New York	33 5th Ave.	10100

This database still suffers from the anomalies described in §3.1. ■

3NF is a relaxation of BCNF that is designed to make decomposition work. When we have an fd $X\rightarrow A$ in a relation scheme R_i, we want to decompose R_i into XA and R_i-A. If, however, A is prime, then there some key $Y\subseteq R_i$ such that $A\in Y$. When we decompose R_i as suggested, the fd $Y\rightarrow R_i$ (since Y is a key) is lost, so the resulting schema does not represent the original one. If A is not prime, no such problem arises. Thus, 3NF does not solve all the anomalies, but, as we shall soon see, it has the property that every universal schema can be represented by a schema in 3NF. In fact, we even have a polynomial time algorithm that produces such a schema. The algorithm, [Ber76], works by "synthesizing" the

relation schemas rather than refine the universal schema by successive decompositions. Thus, the method is called *normalization through synthesis*.

Before describing the synthesis approach, we show that normalization through decomposition is not a practical approach even when we try to achieve 3NF.

Theorem 5.12 [Bee85] The following problem is NP-hard: determine for a given database scheme **D** whether **D** is not in 3NF.

Proof To show that the problem is NP-hard we reduce to it the *prime attribute problem*, which was shown to be NP-complete in [LO78]. The prime attribute problem is to determine for a universal schema $\mathbf{D}=(U,\Sigma)$ and an attribute $A \in U$ whether A is prime in U. Let $\mathbf{D}=(U,\Sigma)$ be a universal schema, and let $A \in U$. Let F be a new attribute. We define a universal schema $\mathbf{D}'=(U',\Sigma')$, where $U'=UF$, and $\Sigma'=\Sigma \cup \{U \rightarrow F\} \cup \{BF \rightarrow U : B \neq A\} \cup \{F \rightarrow A\}$. We claim that A is prime in U if and only if $\mathbf{D}'$ is in 3NF.

The argument is as follows. Because of the fd $U \rightarrow F$, every key of U is also a key of U'. In addition, BF is a key of U' for every $B \neq A$. Thus, A is prime in U' if and only if it is prime in U. Furthermore, all other attributes are prime in U'. Thus, F is a strong determinant of U' if and only if A is not prime in U. Since F is not a superkey of U', $\mathbf{D}'$ is in 3NF if and only if A is prime in U. ■

To describe the synthesis algorithm we need some technical machinery.

Lemma 5.5 $X \rightarrow A_1 \cdots A_k \models X \rightarrow A_i$, $1 \leq i \leq k$, and $\{X \rightarrow A_i : 1 \leq i \leq k\} \models X \rightarrow A_1 \cdots A_k$. ■

A a consequence of the above lemma, we can always convert our fd's to fd's have a single attribute on the right-hand side. Such fd's are said to be in *canonical form*. For a set Σ of fd's, let CANONICAL(Σ) a *canonical cover* of Σ, i.e., a cover of Σ where all fd's are in canonical form.

Let Σ be a set of fd's, and let $X \rightarrow Y$ be an fd in Σ. We say that $X \rightarrow Y$ is *reduced* in Σ if there is no proper subset Z of X such that $\Sigma \models Z \rightarrow Y$. Clearly, if $X \rightarrow Y$ is not reduced, then we can replace it by an fd with a smaller left-hand side. Σ is reduced if all fd's in Σ are reduced. The following algorithm uses our efficient test for implication of fd's to construct a reduced cover of Σ.

Algorithm 5.3

Input: A set Σ of fd's.

Output: A reduced cover of Σ.

```
REDUCE(Σ)
begin
    Δ:=Σ;
    for each fd X→Y in Δ do
```

```
        for each attribute A in X do
            if Δ⊨X−A→Y then
                remove A from X in X→Y in Δ
        end for
    end for
return(Δ)
end.
```

Lemma 5.6 Let Σ be a nonredundant set of fd's in canonical form. Then REDUCED(Σ) is a reduced nonredundant canonical cover of Σ.

Proof Clearly all fd's stay in canonical form, and it is easy to see that the algorithm does not introduce any redundancy. ■

Let Σ be a set of fd's, and let R be a relation scheme. A *key* of R with respect to Σ is an attribute set X such that $\Sigma \models X \rightarrow R$ and for no proper subset Y of X we have that $\Sigma \models Y \rightarrow R$. The following algorithm construct keys.

Algorithm 5.4

Input: A set Σ of fd's and a relation scheme R.

Output: A key of R with respect to Σ.

```
KEY(R,Σ)
begin
    X:=R;
    for each attribute A in R do
        if Σ⊨X−A→R then X:=X−A
    end for
return(X)
end.
```

We can now describe the synthesis algorithm.

Algorithm 5.5

Input: A universal schema (U,Σ).

Output: A database scheme **D** in 3NF that represents (U,Σ).

```
3NF(U,Σ)
begin
    Δ:=REDUCED(NONREDUN(CANONICAL(Σ)))
```

X:=KEY(U,Δ)
D:=∅
for each fd $Y \rightarrow A$ in Δ **do**
D:=D∪(YA,π_{YA}(Δ))
end for
D:=D∪(X,∅)
return(D)
end.

Informally, the algorithm starts by finding a reduced nonredundant canonical cover of Σ. Then for every fd in that cover a relation schema is created. Finally, a key for U is added as another relation scheme. It is clear that the algorithm terminates in polynomial time. We now prove its correctness.

Theorem 5.13 [Ul82] Let (U,Σ) be a universal schema. Then 3NF(U,Σ) is in 3NF and it represents (U,Σ).

Proof Let 3NF(U,Σ)=((R_1,Σ$_1$), . . . ,(R_k,Σ$_k$)), where (R_k,Σ$_k$) is the last relation schema to be added, i.e., R_k is a key of U with respect to Σ and Σ$_k$=∅. We first show that 3NF(U,Σ) is a database schema, i.e., $\bigcup_{i=1}^{k} R_i = U$. We know that $\Delta \models R_k \rightarrow U$, so $cl_\Delta(R_k)=U$. Therefore, for every attribute $A \in U - R_k$, there must be some fd $Y \rightarrow A$ in Δ, so $A \in R_i$ for some i, $1 \le i \le k-1$. Let Δ be a reduced nonredundant canonical cover of Σ. Let (R_i,Σ$_i$) be a relation schema of 3NF(U,Σ).

We now show that 3NF(U,Σ) is in 3NF. There are two cases: either $1 \le i \le k-1$ and R_i came from an fd $Y \rightarrow A$ in Δ, or $i=k$ and R_k is a key of U with respect to Σ. Consider the first case, where $R_i = YA$. We claim that Y is a key of R_i. Suppose it is not. Then there is a proper subset Z of Y such that $\Sigma \models Z \rightarrow YA$. But that means that $Y \rightarrow A$ is not reduced in Δ - contradiction. Thus, the only attribute in R_i that is possibly not prime is A. If there is a violation of 3NF in YA, then that means that there is a proper subset Z of Y such that $\Sigma \models Z \rightarrow A$. But that means that $Y \rightarrow A$ is not reduced in Δ - contradiction. Consider now the second case. If there is a violation of 3NF in R_k, then that means that there is a proper subset Z of R_k and an attribute $B \in R_k - Z$ such that $\Sigma \models Z \rightarrow A$. But then $\Sigma \models R_k - A \rightarrow U$, so R_k is not a key of U with respect to Σ - contradiction.

We now show that 3NF(U,Σ) represents (U,Σ). We first have to show that the decomposition associated with this schema is lossless, that is, for every relation I on U that satisfies Σ, the decomposition of I into $\pi_{R_1}(I)$, . . . ,$\pi_{R_k}(I)$ is lossless. The proof of this claim is out of the scope of this article. The reader is referred to [BDB79,Va84b]. It remains to show that 3NF(U,Σ) preserves the dependencies in

Σ. But this is obvious, since $\bigcup_{i=1}^{k}\Sigma_i = \Delta$, and Δ is a cover of Σ. ■

The synthesis algorithm presented above is actually a simplified version of the algorithm in [BDB79] (which is an improvement of the algorithm in [Ber76]). The algorithm there has the property that it synthesize a database schema with a minimal number of relation schema, namely, given a universal schema (U,Σ), the algorithm synthesizes a database schema **D** that is in 3NF and represents (U,Σ), such that no database schema **D'** that is in 3NF and represents (U,Σ) has fewer relation schemas than **D**.

5.5 Multivalued Dependencies

5.5.1 Motivation

We have seen in the previous sections that the presence of certain functional dependencies in a database schema can cause certain problems, called *anomalies*. We show now that anomalies can also occur in the absence of fd's.

Example 5.17 Consider the following relation:

EMP	CHILD	SKILL
Hilbert	Hilda	Math
Hilbert	Hilda	Physics
Pythagoras	Peter	Math
Pythagoras	Paul	Math
Pythagoras	Peter	Philosophy
Pythagoras	Paul	Philosophy
Turing	Peter	Computer Science

This relation does not obey any fd besides trivial ones such as EMP→EMP. Nevertheless, it still suffers from the anomalies:

(1) *Redundancy*. Several pieces of information, e.g., the information that Pythagoras is skillful in Mathematics, is repeated more than once.

(2) *Possible inconsistency*. Suppose now that we are asked to delete the tuple <Pythagoras, Peter, Math>. Since there seems to be no connection between employees' children and employees' skills, it is not clear whether the absence of this tuple is consistent with the rest of the tuples, as Pythagoras is still skillful in Mathematics and Peter is still his son. ■

A close look in the above example suggest that even though we have no fd's here, there is nevertheless some dependency among the data, since every employee determines a *set* of children and a

set of skills. Such dependency is called a ***multivalued dependency***.

5.5.2 Multivalued Dependencies

A ***multivalued dependency*** (abbr. mvd) is a statement that describes a semantic constraints on data [Fa77a,Za76]. Formally, an mvd is an expression of the form $X \twoheadrightarrow Y$, read X ***multi-determines*** Y, where X and Y are attribute sets. $X \twoheadrightarrow Y$ is *over* an attribute set R if $XY \subseteq R$. $X \twoheadrightarrow Y$ is *satisfied* by a relation I on R if $X \twoheadrightarrow Y$ is over R and for all tuples $u, v \in I$, if $u[X] = v[X]$, then there exists a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[R - XY] = v[R - XY]$. (Of course, if I satisfies $X \twoheadrightarrow Y$, then there must also exists a tuple $w' \in I$ such that $w'[X] = u[X] = v[X]$, $w'[Y] = v[Y]$, and $w'[R - XY] = u[R - XY]$.) Intuitively, the mvd $X \twoheadrightarrow Y$ says that every "X-value" determines a set of "Y-values," independently of the entries in the attributes of $R - XY$. For most of this section, we deal only with relations on U, so the term "$R - XY$" in the above definition can be replaced by "$\overline{XY}$."

Example 5.18 The relation in Example 5.17 satisfies the mvd's EMP$\twoheadrightarrow$CHILD and EMP$\twoheadrightarrow$SKILL. It does not satisfy the mvd CHILD$\twoheadrightarrow$EMP. Intuitively, every employee has a set of children and a set of skills, and these two sets are independent of each other. ■

Several notions that have been defined for fd's can be analogously defined for mvd's. The reader is asked to go through the definitions of §3.2, §3.3, and §3.4, substituting "mvd" for "fd" in the definitions.

Mvd's, like fd's, can be expressed in first-order logic. For example, suppose that $U = ABCD$ and that the attributes A, B, C, and D label the first, second, third, and fourth columns, correspondingly. Then the mvd $AB \twoheadrightarrow C$ can be expressed by the first-order sentence

$$(\forall abc_1c_2d_1d_2)((\mathbf{R}abc_1d_1 \wedge \mathbf{R}abc_2d_2) \supset \mathbf{R}abc_1d_2).$$

This makes the discussion of §3.3 relevant to mvd's. In particular, (finite) implication of mvd's is reducible to (finite) unsatisfiability of the BS class. Thus, we get:

Theorem 5.14 For mvd's, implication and finite implication coincide, and the corresponding decision problems (which also coincide) are decidable. ■

As with fd's, this does not give us a practical algorithm for testing implication. In the next section, we study formal systems for mvd's in order to get such an algorithm.

5.5.3 Formal System for Multivalued Dependencies

A formal system for mvd's was studied in [BFH77] and proven to be sound and complete. Several systems have been investigated in [Bis78, Bis80, Fa76, Men79]. The system **MVD** that we present here is somewhat different from the systems studied in those articles. **MVD** consists of one axiom scheme and two inference rules:

MVD0 (complementation axiom): $\vdash X \twoheadrightarrow \bar{X}$.

MVD1 (augmentation): $X \twoheadrightarrow Y \vdash XZ \twoheadrightarrow YZ$.

MVD2 (difference): $X \twoheadrightarrow Y, S \twoheadrightarrow T \vdash X \twoheadrightarrow Y - T$, if $S \cap Y = \varnothing$.

Example 5.19 Let $U = ABCD$, and let Σ consists of the mvd's $A \twoheadrightarrow BC$ and $D \twoheadrightarrow B$, and let τ be $AB \twoheadrightarrow BD$. We show that $\Sigma \vdash \tau$. By **MVD2**, we have $A \twoheadrightarrow BC, D \twoheadrightarrow B \vdash A \twoheadrightarrow C$. By **MVD0**, we have $\vdash A \twoheadrightarrow BCD$. By **MVD2**, we have $A \twoheadrightarrow BCD, A \twoheadrightarrow C \vdash A \twoheadrightarrow BD$. By **MVD1**, we have $A \twoheadrightarrow BD \vdash AB \twoheadrightarrow BD$. ■

Before proving soundness and completeness we need two lemmas.

Lemma 5.7

(1) $\vdash X \twoheadrightarrow U$.

(2) $\vdash X \twoheadrightarrow X$.

(3) $X \twoheadrightarrow Y \vdash X \twoheadrightarrow \bar{Y}$.

(4) $X \twoheadrightarrow Y, S \twoheadrightarrow T \vdash X \twoheadrightarrow Y \cap T$, if $Y \cap S = \varnothing$.

(5) $X \twoheadrightarrow Y, X \twoheadrightarrow Z \vdash X \twoheadrightarrow Y \cap Z$.

(6) $\vdash X \twoheadrightarrow A$, if $A \in X$.

Proof

(1) By **MVD0**, we have $\vdash \varnothing \twoheadrightarrow U$. By **MVD1**, we have $\varnothing \twoheadrightarrow U \vdash X \twoheadrightarrow U$.

(2) By **MVD0**, we have $\vdash \varnothing \twoheadrightarrow U$. By **MVD2**, we have $\varnothing \twoheadrightarrow U, \varnothing \twoheadrightarrow U \vdash \varnothing \twoheadrightarrow \varnothing$. By **MVD1**, we have $\varnothing \twoheadrightarrow \varnothing \vdash X \twoheadrightarrow X$.

(3) By **MVD0**, $\vdash X \twoheadrightarrow \bar{X}$. By **MVD2**, $X \twoheadrightarrow \bar{X}, X \twoheadrightarrow Y \vdash X \twoheadrightarrow \bar{X} - Y$. By **MVD1**, $X \twoheadrightarrow \bar{X} - Y \vdash X \twoheadrightarrow \bar{Y}$.

(4) By (3), $S \twoheadrightarrow T \vdash S \twoheadrightarrow \bar{T}$. By **MVD2**, $X \twoheadrightarrow Y, S \twoheadrightarrow \bar{T} \vdash X \twoheadrightarrow Y - \bar{T}$. But $Y - \bar{T} = Y \cap T$.

(5) By **MVD0**, $\vdash X \twoheadrightarrow \bar{X}$. By (4), $X \twoheadrightarrow \bar{X}, X \twoheadrightarrow Y \vdash X \twoheadrightarrow \bar{X} \cap Y$. By (4), $X \twoheadrightarrow \bar{X} \cap Y, X \twoheadrightarrow Z \vdash X \twoheadrightarrow \bar{X} \cap Y \cap Z$. By **MVD1**, $X \twoheadrightarrow \bar{X} \cap Y \cap Z \vdash X \twoheadrightarrow Y \cap Z$.

(6) By **MVD0**, $\vdash X \twoheadrightarrow \bar{X}$. By **MVD1**, $X \twoheadrightarrow \bar{X} \vdash X \twoheadrightarrow \bar{X}A$. By (2), $\vdash X \twoheadrightarrow X$. By (5), $X \twoheadrightarrow X, X \twoheadrightarrow \bar{X}A \vdash X \twoheadrightarrow A$. ■

Lemma 5.8 Let W be an attribute set, and let I be a two tuple relation $\{u, v\}$ such that u and v agree precisely on $\bar{W}$, i.e, $u[\bar{W}] = v[\bar{W}]$ and $u[A] \neq v[A]$ for all $A \in W$. Then I satisfies the mvd $S \twoheadrightarrow T$ if and only if

either $S \cap W \neq \emptyset$, $W \subseteq T$, or $W \cap T = \emptyset$.

Proof Suppose first that $S \cap W \neq \emptyset$. Then $u[S] \neq v[S]$, and I satisfies $S \twoheadrightarrow T$ vacuously. Suppose now that $S \cap W = \emptyset$. Then $u[S] = v[S]$, so there should be a tuple $w \in I$ such that $w[S] = u[S] = v[S]$, $w[T] = u[T]$, and $w[\overline{ST}] = v[\overline{ST}]$. If $W \subseteq T$, then take w to be u, and if $W \cap T = \emptyset$, then take w to be v. Conversely, suppose that I satisfies $S \twoheadrightarrow T$. Then, either $u[S] \neq v[S]$, in which case $S \cap W \neq \emptyset$, or there is a tuple $w \in I$ such that $w[S] = u[S] = v[S]$, $w[T] = u[T]$, and $w[\overline{ST}] = v[\overline{ST}]$. But w must be either u, in which case $W \subseteq T$, or it must be v, in which case $W \cap T = \emptyset$. ■

We can now prove soundness and completeness.

Theorem 5.15 The system **MVD** is sound and complete.

Proof Soundness: We show that the axiom and the inference rules are sound. Let I be a relation on U, and let $u, v \in I$. Suppose that $u[X] = v[X]$. To show that **MVD0** is sound, we need a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[\overline{X}] = u[\overline{X}]$, and $w[\emptyset] = v[\emptyset]$ (since $\overline{X\overline{X}} = \emptyset$). Clearly, u is the desired tuple.

Suppose now that I satisfies $X \twoheadrightarrow Y$ and that $u[XZ] = v[XZ]$. Then $u[X] = v[X]$, so there is a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[\overline{XY}] = v[\overline{XY}]$. Since $u[XZ] = v[XZ]$, it follows that $w[XZ] = u[XZ] = v[XZ]$ and $w[YZ] = u[YZ]$. Now $\overline{XYZ} \subseteq \overline{XY}$, so $w[\overline{XYZ}] = v[\overline{XYZ}]$. Thus, **MVD1** is sound.

Finally, suppose that I satisfies $X \twoheadrightarrow Y$ and $S \twoheadrightarrow T$, and that $u[X] = v[X]$. Then there is a tuple w such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[Z] = v[Z]$, where $Z = \overline{XY}$. Thus, $w[XZ] = v[XZ]$. Since $Y \cap S = \emptyset$, we have that $S \subseteq XZ$, so $w[S] = v[S]$. Therefore, there is a tuple $p \in I$ such that $p[S] = v[S] = w[S]$, $p[T] = v[T]$, and $p[\overline{ST}] = w[\overline{ST}]$. Since $u[x] = v[X] = w[X]$, clearly $p[X] = v[X] = w[X]$. Since $S \cap Y = \emptyset$, we have that $Y - T \subseteq \overline{ST}$, so $p[Y - T] = w[Y - T] = u[Y - T]$. Finally, $\overline{X(Y-T)} = Z \cup (T - X)$, so $p[\overline{X(Y-T)}] = v[\overline{X(Y-T)}]$, since $p[T] = v[T]$ and $p[Z] = v[Z]$. Thus, **MVD2** is sound.

Completeness: We have to show that if $\Sigma \models \sigma$, then $\Sigma \vdash \sigma$. We prove the contrapositive: if $\Sigma \not\vdash \sigma$, then $\Sigma \not\models \sigma$.

Let σ be $X \twoheadrightarrow Y$. Suppose that $\Sigma \vdash X \twoheadrightarrow Y - X$. Then, by **MVD2**, $\Sigma \vdash X \twoheadrightarrow Y$. Thus, we can assume without loss of generality that X and Y are disjoint. Consider the collection of sets that are provably multidetermined by X:

$$rhs_{\Sigma}(X) = \{Z : \Sigma \vdash X \twoheadrightarrow Z\}.$$

By Lemma 5.7, $rhs_{\Sigma}(X)$ is a Boolean algebra, since it contains a maximal element and is closed under complement and intersection. An *atom* in a Boolean algebra is a minimal nonzero element. Since $rhs_{\Sigma}(X)$ is a field of finite sets, it is clear that every nonempty set in $rhs_{\Sigma}(X)$ includes an atom. That is,

$rhs_\Sigma(X)$ is an *atomic* Boolean algebra. In an atomic Boolean algebra every element is the union of the atoms that it contains. The collection of these atoms is called the *dependency basis* of X with respect to Σ, denoted $dep_\Sigma(X)$. Thus,

$$dep_\Sigma(X)=\{Z : Z\neq\emptyset, \Sigma\vdash X\twoheadrightarrow Z, \text{ and if } \Sigma\vdash X\twoheadrightarrow R, R\subseteq Z, \text{ and } R\neq\emptyset, \text{ then } R=Z\}.$$

Note that the sets in $dep_\Sigma(X)$ form a partition of U, by Lemma 5.7.

Since $\Sigma\nvdash X\twoheadrightarrow Y$, it follows that Y is not in $rhs_\Sigma(X)$. Thus, Y is not a union of sets in $dep_\Sigma(X)$. Consequently, there is a set W in $dep_\Sigma(X)$, such that W *intersects* Y *nontrivially*, i.e., $W\not\subseteq Y$ and $W\cap Y\neq\emptyset$. Since $W\in rhs_\Sigma(X)$, we have $\Sigma\vdash X\twoheadrightarrow W$. We claim that $W\cap X=\emptyset$. Suppose not, and let $A\in W\cap X$. By Lemma 5.7, $A\in dep_\Sigma(X)$, so we must have that $W=A$. But then W cannot intersect Y nontrivially.

We construct a relation I as follows. I consists of two tuples u and v, such that u and v agree precisely on $\overline{W}$. For example, $u[A]=0$ for each attribute A, $v[A]=0$ for each attribute $A\in\overline{W}$, and $v[A]=1$ for each attribute $A\in W$.

By Lemma 5.8, I satisfies an mvd $S\twoheadrightarrow T$ if and only if either $S\cap W\neq\emptyset$, or $W\subseteq T$, or $W\cap T=\emptyset$. Since X is disjoint from W and W intersects Y nontrivially, it follows that I does not satisfy $X\twoheadrightarrow Y$.

We now claim that I satisfies Σ. Let $S\twoheadrightarrow T$ be an mvd in Σ. If I does not satisfy $S\twoheadrightarrow T$, then $S\cap W=\emptyset$, $W\not\subseteq T$, and $W\cap T\neq\emptyset$, by Lemma 5.8. But then, by **MVD2**, $X\twoheadrightarrow W, S\twoheadrightarrow T\vdash X\twoheadrightarrow W\cap T$, so $W\cap T$ should be in $dep_\Sigma(X)$. But $W\cap T\subset W$, so we cannot have both W and $W\cap T$ in $dep_\Sigma(X)$. It follows that I satisfies $S\twoheadrightarrow T$.

Thus, I satisfies Σ and it does not satisfy σ, so $\Sigma\not\models\sigma$. ■

As with fd's, the completeness proof is also a direct proof that for mvd's implication and finite implication coincide, since the *counterexample* relation that we have constructed is finite.

5.5.4 Testing Implication of Multivalued Dependencies

As with fd's, the most important fact about implication of mvd's is buried in the proof of the completeness of the system **MVD**. We recall the definition of the *dependency basis*, where now, by the completeness of **MVD**, we can replace $\vdash$ by $\models$:

$$dep_\Sigma(X)=\{Z : Z\neq\emptyset, \Sigma\models X\twoheadrightarrow Z, \text{ and if } \Sigma\models X\twoheadrightarrow R, R\subseteq Z, \text{ and } R\neq\emptyset, \text{ then } R=Z\}.$$

The following properties of the dependency basis follow from the proof of Theorem 5.15.

Lemma 5.9 [BFH77]

(1) $dep_\Sigma(X)$ is a partition of U,

(2) for each $A \in X$, we have $A \in dep_\Sigma(X)$, and

(3) $\Sigma \models X \twoheadrightarrow Y$ if and only if Y is a union of sets in $dep_\Sigma(X)$. ■

It follows from Lemma 5.9 that if we can efficiently construct dependency bases, then we can efficiently test implication of mvd's. Beeri [Bee80] was the first to describe a polynomial time algorithm for the construction of the dependency basis. We describe here a better algorithm, Hagihara et al. [HITK79].

Algorithm 5.6

Input: The set U of all attribute, an attribute set X, and a set Σ of mvd's.

Output: The dependency basis of X with respect to Σ.

```
DEP(U,X,Σ)
begin
    DEP={X̄}
    for each attribute A in X do DEP:=DEP∪{A}
    end for
    while there is some W in DEP and some S↠T in Σ such that
        S∩W=∅ and T∩W≠∅ and W⊈T do
            DEP:=DEP∪{W∩T,W−T}−{W}
    end while
return(DEP)
end.
```

We have to show that DEP(U,X,Σ) terminates and indeed returns the dependency basis of X with respect to Σ.

Lemma 5.10 DEP(U,X,Σ) terminates and returns $dep_\Sigma(X)$.

Proof The algorithm starts by constructing a partition of U and then goes on to refine that partition. Thus, it must terminate. We now show that the algorithm returns the dependency basis of X.

We first show that if at any stage of the algorithm W is a set in DEP, then $\Sigma \models X \twoheadrightarrow W$. Before the main loop of the algorithm, DEP gets the value $\{A : A \in X\} \cup \{\bar{X}\}$. By Lemma 5.7, for each W in this partition we have $\Sigma \models X \twoheadrightarrow W$. In the main loop of the algorithm we replace a set W in DEP by $W \cap T$ and $W - T$, provided there is an mvd $S \twoheadrightarrow T$ in Σ such that S is disjoint from W and W intersects T nontrivially. But in that case, by MVD2 and Lemma 5.7, $\Sigma \models X \twoheadrightarrow W - T$ and $\Sigma \models X \twoheadrightarrow W \cap T$.

Suppose now that DEP(U,X,Σ) is not the dependency basis of X. Then there is a set Y in DEP(U,X,Σ) and a set W in $dep_\Sigma(X)$ such that W is a proper subset of Y. We claim that $Y \cap X = \emptyset$. Suppose not, and assume $A \in Y \cap X$. But since A is in DEP(U,X,Σ), it must be the case that $Y = A$. But then W cannot be a proper subset of Y.

We construct a relation I as follows. I consists of two tuples u and v, such that u and v agree precisely on $\bar{Y}$. By Lemma 5.8, I satisfies an mvd $S \twoheadrightarrow T$ if and only if either $S \cap Y \neq \emptyset$, or $Y \subseteq T$, or $Y \cap T = \emptyset$. Since X is disjoint from Y and W intersects Y nontrivially, it follows that I does not satisfy $X \twoheadrightarrow W$.

We now claim that I satisfies Σ. Let $S \twoheadrightarrow T$ be an mvd in Σ. If I does not satisfy $S \twoheadrightarrow T$, then $S \cap Y = \emptyset$, $Y \not\subseteq T$, and $Y \cap T \neq \emptyset$, by Lemma 5.8. But then Y should be replaced in DEP by $Y \cap T$ and $Y - T$. It follows that I satisfies $S \twoheadrightarrow T$. Thus, I satisfies Σ and it does not satisfy $X \twoheadrightarrow W$. But we have shown that $\Sigma \models X \twoheadrightarrow W$ - a contradiction. It follows that DEP(U,X,Σ) must be $dep_\Sigma(X)$. ■

The alert reader may have noticed the similarity in the proofs of Theorem 5.15 and Lemma 5.10. Indeed, one can view an execution of the algorithm as an organized derivation in the formal system **MVD**.

Example 5.20 Let $U = ABCD$, and let Σ consists of the mvd's $A \twoheadrightarrow BC$ and $D \twoheadrightarrow B$. Let us calculate $dep_\Sigma(AB)$. Initially DEP is assigned the partition $\{A,B,CD\}$. We then apply the mvd $A \twoheadrightarrow BC$ and replace CD by C and D. So $dep_\Sigma(AB) = \{A,B,C,D\}$. ■

It is not hard to implement DEP to run in polynomial time. Galil [Ga82] has shown how to implement it to run in almost linear time.

Theorem 5.16 [Ga82] The implication problem for mvd's can be solved in time $O(n \log n)$, where n is the length of the input. ■

5.5.5 Multivalued Dependencies and Propositional Logic

Looking again in the proofs of Theorem 5.15 and Lemma 5.10, we see that the counterexample relation constructed there is a two-tuple relation.

Theorem 5.17 Let Σ be a set of mvd's, and σ be an mvd. Then $\Sigma \models \sigma$ if and only if $\Sigma \models_f \sigma$ if and only if $\Sigma \models_2 \sigma$. ■

As in §3.5, this suggests a correspondence between mvd's and propositional logic. With fd's the correspondence shed some light on fd's. Here, surprisingly, the correspondence sheds some light on propositional logic. In order to define propositional implication for mvd's, we have to extend relational truth assignments to assign truth values to mvd's: ψ assigns the value 1 to $X \twoheadrightarrow Y$ if and only if whenever $\psi(X) = 1$, then $\psi(Y) = 1$ or $\psi(\overline{XY}) = 1$.

Theorem 5.18 [SDPF81] Let Σ be a set of mvd's, and σ be an mvd. Then $\Sigma \models \sigma$ if and only if $\Sigma \models_p \sigma$.

Proof By Theorem 5.17, it suffices to show that $\Sigma \models_2 \sigma$ iff $\Sigma \models_p \sigma$. The basis for this equivalence is a simple correspondence between relational truth assignments and two-tuple relations.

Let ψ be a relational truth assignment. Then $I_\psi = \{u,v\}$ is a two-tuple relation such that $u[A] = v[A]$ iff $\psi(A) = 1$. Let $I = \{u,v\}$ be a two-tuple relation. Then ψ_I is a relational truth assignment such that $\psi(A) = 1$ iff $u[A] = v[A]$. Let τ be an mvd $X \twoheadrightarrow Y$. We claim that $\psi(\tau) = 1$ iff I_ψ satisfies τ, and I satisfies τ iff $\psi_I(\tau) = 1$.

Suppose first that $\psi(\tau) = 1$. Let $R = \{A : \psi(A) = 1\}$. If $\psi(X) = 0$, then $X \not\subseteq R$, so $u[X] \neq v[X]$ and I_ψ satisfies τ. Suppose now that $\psi(X) = 1$. Then $X \subseteq R$, so $u[X] = v[X]$. For I_ψ to satisfy τ there should be a tuple $w \in I_\psi$ such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[\overline{XY}] = v[\overline{XY}]$. If $\psi(Y) = 1$, then $Y \subseteq R$, so we can take w to be v. If $\psi(\overline{XY}) = 1$, then $\overline{XY} \subseteq R$, so we can take w to be u. So I_ψ satisfies τ.

Suppose now that I satisfies τ. Let $R = \{A : u[A] = v[A]\}$. If $u[X] \neq v[X]$, then $X \not\subseteq R$, so $\psi_I(X) = 0$, and $\psi_I(X \twoheadrightarrow Y) = 1$. If $u[X] = v[X]$, then there should be a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[\overline{XY}] = v[\overline{XY}]$. If w is u, then $u[\overline{XY}] = v[\overline{XY}]$, so $\overline{XY} \subseteq R$ and $\psi_I(\overline{XY}) = 1$. If w is v, then $u[Y] = v[Y]$, so $Y \subseteq R$ and $\psi_I(R) = 1$. At any case $\psi_I(X \twoheadrightarrow Y) = 1$.

A similar correspondence between ψ and I_ψ and between I and ψ_I holds for sets of fd's. Suppose now that $\Sigma \models_2 \sigma$. We want to show that $\Sigma \models_p \sigma$. Let ψ be a relational truth assignment such that $\psi(\Sigma) = 1$. Then I_ψ satisfies Σ, and since $\Sigma \models_2 \sigma$, it must be the case that I_ψ satisfies σ. But then $\psi(\sigma) = 1$. We have shown that $\psi(\Sigma) = 1$ entails $\psi(\sigma) = 1$ for any relational truth assignment σ, so $\Sigma \models_p \sigma$. An analogous argument shows that if $\Sigma \models_p \sigma$, then also $\Sigma \models_2 \sigma$. ■

Theorem 5.18 enables us to view mvd's as formulas in propositional logic. Let $X = A_1 \cdots A_k$, $Y = B_1 \cdots B_l$, and $\overline{XY} = C_1 \ldots C_m$. Then the mvd $X \twoheadrightarrow Y$ can be viewed as the formula

$$A_1 \wedge \cdots \wedge A_k \supset (B_1 \wedge \cdots \wedge B_k) \vee (C_1 \wedge \cdots \wedge C_k).$$

For example, if $U = ABC$, then the mvd $A \twoheadrightarrow B$ can be viewed as the formula $A \supset B \vee C$. This formula is not equivalent to a set of Horn formulas. Thus, the results of §5.4 together with the correspondence between mvd's and propositional logic yield a polynomial time algorithm for a subclass of propositional logic.

5.5.6 Functional and Multivalued Dependencies

So far we have considered functional and multivalued dependencies separately. Since both types of dependencies express natural semantic constraints, their interaction is also of importance. As for fd's

alone and mvd's alone, the (finite) implication problem for fd's and mvd's is reducible to the (finite) unsatisfiability problem for the BS class. Thus, the problems coincide and they are decidable. As before, we resort to formal systems in order to gain a better understanding.

The interaction of fd's and mvd's was first studied in [BFH77]. We present here a formal system **FD−MVD** that is somewhat different from the system in [BFH77]. The system consists of the system **MVD**, an axiom and an inference rules for fd's, and two rules that describe the interaction between fd's and mvd's.

FD3 (reflexivity): $\vdash X \rightarrow A$, if $A \in X$.

FD4 (union): $X \rightarrow Y, X \rightarrow Z \vdash X \rightarrow YZ$.

FD5 (decomposition): $X \rightarrow Y \vdash X \rightarrow A$, if $A \in Y$.

FD−MVD0 (translation): $X \rightarrow Y \vdash X \twoheadrightarrow Y$.

FD−MVD1 (intersection): $X \twoheadrightarrow Y, S \rightarrow T \vdash X \rightarrow Y \cap T$, if $S \cap Y = \emptyset$.

MVD0 (complementation axiom): $\vdash X \twoheadrightarrow \overline{X}$.

MVD1 (augmentation): $X \twoheadrightarrow Y \vdash XZ \twoheadrightarrow YZ$.

MVD2 (difference): $X \twoheadrightarrow Y, S \twoheadrightarrow T \vdash X \twoheadrightarrow Y - T$, if $S \cap Y = \emptyset$.

Example 5.21 Let $U = ABCD$. We show that $A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$. (This derivation is done by one step using **FD1** in the system **FD**, but this rule does not belong to the system **FD−MVD**.) By **FD−MVD0**, $A \rightarrow B \vdash A \twoheadrightarrow B$. By Lemma 5.5, $A \twoheadrightarrow B \vdash A \twoheadrightarrow CD$ (since the system **MVD** is a subset of the system **FD−MVD**). By **FD−MVD1**, $A \twoheadrightarrow CD, B \rightarrow C \vdash A \rightarrow C$. ■

Theorem 5.19 The system **FD−MVD** is sound and complete.

Proof

Soundness: We leave it to the reader to prove that the rules **FD3** and **FD4** are sound. Consider the rule **FD−MVD0**. Let I be a relation that satisfies $X \rightarrow Y$. Let $u, v \in I$ such that $u[X] = v[X]$. Then there should be a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = v[Y]$, and $w[\overline{XY}] = v[\overline{XY}]$. But $u[Y] = v[Y]$, since I satisfies $X \rightarrow Y$, so take w to be v. Consider now the rule **FD−MVD1**. Let I be a relation that satisfies $X \twoheadrightarrow Y$ and $S \rightarrow T$, where $S \cap Y = \emptyset$. Let $u, v \in I$ such that $u[X] = v[X]$. There is a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = v[Y]$, and $w[\overline{XY}] = v[\overline{XY}]$. Since $S \cap Y = \emptyset$, $w[S] = v[S]$, so $w[T] = v[T]$. Since $w[Y] = u[Y]$, we have that $u[Y \cap T] = v[Y \cap T]$, as desired.

Completeness: Again we prove that if $\Sigma \not\vdash \sigma$, then $\Sigma \not\models \sigma$. Consider first the case that σ is an mvd $X \twoheadrightarrow Y$. As in the proof of Theorem 5.15, we define the dependency basis of X with respect to Σ (which now

contains both fd's and mvd's).

$$dep_\Sigma(X)=\{Z : Z\neq\emptyset, \Sigma\vdash X\twoheadrightarrow Z, \text{ and if } \Sigma\vdash X\twoheadrightarrow R, R\subseteq Z, \text{ and } R\neq\emptyset, \text{ then } R=Z\}$$

As in the proof of Theorem 5.15, since $\Sigma\nvdash X\twoheadrightarrow Y$, there is a set W in $dep_\Sigma(X)$, such that W intersects Y nontrivially, $\Sigma\vdash X\twoheadrightarrow W$, and $W\cap X=\emptyset$. We construct a relation I with two tuples u and v, such that u and v agree precisely on $\overline{W}$.

By Lemma 5.8, I satisfies an mvd $S\twoheadrightarrow T$ if and only if either $S\cap W\neq\emptyset$, $W\subseteq T$, or $W\cap T=\emptyset$. Consequently I does not satisfy $X\twoheadrightarrow Y$ and I satisfies all mvd's in in Σ. It remain to show that I satisfies all fd's in Σ.

Let $S\rightarrow T$ be an fd in Σ. If $u[S]=v[S]$, then $S\cap W=\emptyset$. If $T\cap W=\emptyset$, then $u[T]=v[T]$, so assume that $T\cap W\neq\emptyset$. Let $A\in T\cap W$. By **FD5**, $S\rightarrow T\vdash S\rightarrow A$; by **FD−MVD1**, $X\twoheadrightarrow W, S\rightarrow A\vdash X\rightarrow A$; and by **FD−MVD0**, $X\rightarrow A\vdash X\twoheadrightarrow A$. Thus, $A\in dep_\Sigma(X)$. But since $A\in W$, it follows that $W=A$, contradicting the fact that W nontrivially intersects Y. Thus, I satisfies Σ and it does not satisfy σ, so $\Sigma\not\models\sigma$.

We now consider the case that σ is an fd $X\rightarrow Y$. If $\Sigma\nvdash X\rightarrow Y$, then there is some attribute $B\in Y$ such that $\Sigma\nvdash X\rightarrow B$ (by **FD4** and **FD5**). Again let $X^+=\{A : \Sigma\vdash X\rightarrow A\}$. Since $\Sigma\nvdash X\rightarrow B$, we have that $B\notin X^+$. Let W be the set in $dep_\Sigma(X)$ such that $A\in W$ (since, as we observed in the proof of Theorem 5.15, $dep_\Sigma(X)$ is a partition of U). As before, we construct a relation I with two tuples u and v that agree precisely on W.

We claim that $X\cap W=\emptyset$. Suppose not, and let $A\in X\cap W$. By Lemma 5.7, $A\in dep_\Sigma(X)$, so we must have $W=A$. But then $A=B$, and $\vdash X\rightarrow B$ by **FD3**. It follows that $u[X]=v[X]$ and $u[B]\neq v[B]$, so I does not satisfy σ. It remain to show that I satisfy Σ. This is done as in the case that σ is an mvd. Thus, I satisfies Σ and it does not satisfy σ, so $\Sigma\not\models\sigma$. ■

As with fd's alone and mvd's alone, the ideas in the completeness proof can be used to devise an efficient algorithm for testing implication of fd's and mvd's.

Theorem 5.20 [Ga82] The implication problem for fd's and mvd's can be solved in time $O(n\log n)$, where n is the length of the input. ■

5.5.7 Database Schema Design and Embedded Mvd's

In §5.1 we saw that multivaled dependencies cause certain anomalies. Again, *decomposition* seems to be the solution.

Example 5.22 The solution to the problems describe in Example 5.17 is to decompose the relation into the following relations:

EMP	CHILD
Hilbert	Hilda
Pythagoras	Peter
Pythagoras	Paul
Turing	Peter

EMP	SKILL
Hilbert	Math
Hilbert	Physics
Pythagoras	Math
Pythagoras	Philosophy
Turing	Computer Science

Intuitively, this decomposition seems to be lossless. ■

Our intuition in decomposing a relation that has multivalued dependencies is justified by the following theorem.

Theorem 5.21 [Fa77] Let I be a relation on R, let $X \twoheadrightarrow Y$ be an mvd, and let $Z = R - XY$. If I satisfies $X \twoheadrightarrow Y$, then the decomposition of I into $\pi_{XY}(I)$ and $\pi_{XZ}(I)$ is lossless.

Proof Let $J = \pi_{XY}(I) * \pi_{XZ}(I)$. We know that $I \subseteq J$, so we have to show that $J \subseteq I$. Let $w \in J$. Then $w[XY] \in \pi_{XY}(I)$ and $w[XZ] \in \pi_{XZ}(I)$. Thus, there are tuples $u, v \in I$ such that $w[XY] = u[XY]$ and $w[XZ] = v[XZ]$. In particular, $u[X] = v[X]$. But then there is a tuple $w' \in I$ such that $w'[X] = u[X] = v[X]$, $w'[Y] = u[Y]$, and $w'[Z] = v[Z]$. Consequently, $w'[XYZ] = w[XYZ]$, so $w = w'$. ■

Based on the above ideas a schema design theory was developed for mvd's (e.g., [De78, Fa77, Li81, ZM81]). This theory, however, suffers from a basic difficulty that did not arise with fd's. This difficulty stems from an essential difference between fd's and mvd's. An fd $X \rightarrow Y$ is "context independent," that is, its satisfaction in a relation depends only on the entries for the attributes in XY. More formally:

Theorem 5.22 Let I be a relation on R, and let $X \rightarrow Y$ be an fd such that $XY \subseteq R$. Then I satisfies $X \rightarrow Y$ if and only if $\pi_{XY}(I)$ satisfies $X \rightarrow Y$. ■

Mvd's, in contrast are "context dependent," since satisfaction of an mvd $X \twoheadrightarrow Y$ in a relation on R depends also on the entries for the attributes in $R - XY$. (In fact the relation on XY always satisfies $X \twoheadrightarrow Y$.) Thus, while with fd's we could mix together fd's that come from different relation schemas, we cannot do the same with mvd's, since these mvd's may have different "contexts." The solution to this difficulty is to change the notation for mvd's, so as to make them "context independent." Such mvd's are called *embedded mvd's* [Fa77], since their "context" can be properly embedded in the global "context" (we shall see in a minute what these contexts are).

A *embedded mvd* (abbr. emvd) is an expression of the form $X \twoheadrightarrow Y|Z$, where X, Y, and Z are attribute sets. $X \twoheadrightarrow Y|Z$ is *over* an attribute set R if $XYZ \subseteq R$. $X \twoheadrightarrow Y|Z$ is *satisfied* by a relation I on R if $X \twoheadrightarrow Y|Z$ is over R and for all tuples $u, v \in I$, if $u[X] = v[X]$, then there exists a tuple $w \in I$ such that $w[X] = u[X] = v[X]$, $w[Y] = u[Y]$, and $w[Z] = v[Z]$. Intuitively, the mvd $X \twoheadrightarrow Y|Z$ says that every "X-value"

determines a set of "Y-values" and a set of "Z-values" and these two sets are independent of each other. Here the "global context" is the relation scheme R, and the "context" of $X \twoheadrightarrow Y|Z$ is XYZ. Since XYZ can be a proper subset of R, $X \twoheadrightarrow Y|Z$ is said to be embedded.

Example 5.23 Consider the following relation:

EMP	CHILD	BIRTHDATE	SKILL
Hilbert	Hilda	June 1, 1899	Math
Hilbert	Hilda	June 1, 1899	Physics
Pythagoras	Peter	July 4, 320BC	Math
Pythagoras	Paul	July 4, 322BC	Math
Pythagoras	Peter	July 4, 320BC	Philosophy
Pythagoras	Paul	July 4, 322BC	Philosophy
Turing	Peter	Feb 28, 1937	Computer Science

This relation does not satisfies the mvd EMP↠CHILD. It does, however, satisfy the emvd EMP↠CHILD|SKILL. (It also satisfies the mvd EMP↠SKILL.) ■

The following theorem formalizes the "context independence" of emvd's.

Theorem 5.23 Let I be a relation on R, and let $X \twoheadrightarrow Y|Z$ be an emvd over R. Then I satisfies $X \twoheadrightarrow Y|Z$ if and only if $\pi_{XYZ}(I)$ satisfies $X \twoheadrightarrow Y|Z$. ■

At this point the reader probably expects us to present a sound and complete formal system for emvd's, to get an efficient algorithm for testing implication, and to develop schema design theory. Unfortunately, none of this has been done. The (finite) implication problem for emvd's has stayed open since emvd's were defined in 1977. This is one of the major open problems in dependency theory.

To see why emvd's differ from mvd's in such a radical way, consider the way emvd's are expressed in first-order logic. For example, suppose that $U=ABCD$ and that the attributes A, B, C, and D label the first, second, third, and fourth columns, correspondingly. Then the mvd $A \twoheadrightarrow B|C$ can be expressed by the first-order sentence

$$(\forall abc_1c_2d_1d_2)(\exists e)((\mathbf{R}ab_1c_1d_1 \wedge \mathbf{R}ab_2c_2d_2) \supset \mathbf{R}ab_1c_2e)).$$

The difference between this sentence and the sentences that express fd's and mvd's is that this sentence has an existential quantifier that follows the universal quantifiers. This existential quantifier makes a big difference. In particular, the sentence $\sigma_1 \wedge \cdots \wedge \sigma_k \wedge \neg\sigma$, where the σ_i's are emvd's, is not in the BS class any more.

While several inference rules for emvd's have been investigated (e.g., [ITK83]), no complete formal system is known. The following theorem says that for emvd's solving the implication problem and

finding a complete formal system are equivalent.

Theorem 5.24 [BV84] The implication problem for emvd's is solvable if and only if there is a sound and complete formal system for emvd's.

Proof

Only if: Suppose that the implication problem for emvd's is solvable, and consider the formal system consisting of one inference rule:

$$\sigma_1, \ldots, \sigma_k \vdash \sigma, \text{ if } \{\sigma_1, \ldots, \sigma_k\} \models \sigma.$$

Clearly, this formal system is sound and complete for emvd's.

If: Suppose that **EMVD** is a sound and complete formal system for emvd's. Let Σ be a set of emvd's, and let σ be an emvd. To decide whether $\Sigma \models \sigma$ we list every possible sequence (without repetitions) of emvd's over the attributes that occur in Σ and σ and check whether the sequence is a derivation of σ from Σ by **EMVD**. Since there is a finite number of such sequences, this process must terminate. Hence, the implication problem for emvd's is solvable. ■

In Theorem 5.24 we implicitly assume that a formal system has to be effective, that is, it can be effectively checked whether a given sequence of dependencies is a derivation in the system. The formal systems **FD**, **MVD**, and **FD−MVD** are, of course, effective. In [PP80,SW82] another way in which formal systems can be restricted is considered. A formal system is said to be *k-ary* if all inference rules in the system are of the form $\sigma_1, \ldots, \sigma_n \vdash \sigma$, where $n \leq k$. For example, the formal systems **FD**, **MVD**, and **FD−MVD** are all 2-ary.

Theorem 5.25 [PP80,SW82] For all $k>0$, there is no sound and complete k-ary formal system (possibly noneffective) for implication and finite implication of emvd's. ■

5.6 More Data Dependencies

5.6.1 Motivation

So far we have introduced fd's, mvd's, and emvd's. Do we need any more? Indeed we do, for several reasons.

The first reason is that life is not simple. There are semantic constraints that cannot be described by any of the dependencies introduced so far. We mention two examples.

Example 5.24

(1) Consider a database with two relations: the relation *ES*(Employee, Salary) and the relation *DM*(Department, Manager). Each manager, however, is also an employee. So we would like to express the constraint that every manager entry in the *DM* relation appears as an employee entry in the *ES* relation. Such a constraint is called an *inclusion dependency* ([Fa81]), and is written $DM[M] \subseteq ES[E]$.

(2) Let I be a relation on X, and let $X_1, \ldots, X_m$ be attribute sets such that $X = \bigcup_{j=1}^{m} X_j$. We have observed in §2.2 that the decomposition of I to $\pi_{X_1}(I), \ldots, \pi_{X_m}(I)$ could be lossy. We can impose on I the constraint that the above decomposition be lossless. Such a constraint is called a *join dependency* ([ABU79,Ri78]), and is written $*[X_1, \ldots, X_m]$. ■

Another reason to introduce more dependencies is in the hope of solving the (finite) implication problem for emvd's. It is conceivable that for a larger class of dependencies a decision procedure would be apparent, while the specialization of the algorithm for emvd's is too murky to be visible. Also, a larger class of dependencies may have an elegant formal system that the narrower class of emvd's lacks. Even if one only cares about fd's, there is a technical reason to introduce more general dependencies, since fd's do not have enough expressive power when dealing with projections of relations. We now make the latter comment more precise.

We have mentioned that we view the language of dependencies as a semantic specification language. From this point of view, dependencies are means to specify classes of "semantically meaningful" relations. Given a set Σ of dependencies over an attribute set R, let $SAT_R(\Sigma)$ be the class of all relations on R that satisfy Σ. Thus, $SAT_R(\Sigma)$ is the class of all "semantically meaningful" relations on R with respect to Σ. A set Ψ of relations on R is an *fd-class* if there is a set Σ of fd's such that $\Psi = SAT_R(\Sigma)$. The definition of an *mvd-class* is analogous. Given a set Ψ of relations on R, and an attribute set $S \subseteq R$, let $\pi_S(\Psi)$ be the projection on S of the relations in Ψ, i.e., $\pi_S(\Psi) = \{\pi_S(I) : I \in \Psi\}$. Such a class of relations is called a *projective class*.

Projective classes arise very naturally in the context of *user views*. Very often certain users are not allowed to see the whole database but only a portion of it, a portion that may be defined by projection. For example, very often most users would not be allowed to see the salary data in a personnel database. If the class of "meaningful" relations in the database is $SAT(\Sigma)$, the class of "meaningful" relations for these users might be $\pi_S(SAT_R(\Sigma))$, for an appropriate attribute set S. One would like our specification language to be able to specify also projective classes, i.e., one would like the projection of an fd-class to be an fd-class and the projection of an mvd-class to be an mvd-class. Unfortunately, this is not the case.

Theorem 5.26 [Fa82,GZ82] There is a projection of an fd-class that is not an fd-class.

Proof Let Σ be a set of fd's over R, let $\Sigma^+ = \{\sigma : \Sigma \models \sigma\}$, let $\Psi = SAT_R(\Sigma)$, and let $\Omega = \pi_S(\Psi)$. We claim that Ω is an fd-class if and only if $\Omega = SAT_S(\pi_S(\Sigma^+))$. Clearly the latter is a sufficient condition, so we prove that it is also necessary.

Suppose that $\Omega = SAT_S(\Delta)$ for some set Δ of fd's over S. Let I be a relation on S. If I satisfies Δ, then $I \in \Omega$, so $I = \pi_S(J)$ for some $J \in \Psi$. But then J satisfies Σ^+, so I satisfies $\pi_S(\Sigma^+)$. Consequently, $\Delta \models \pi_S(\Sigma^+)$. Suppose that $\pi_S(\Sigma^+) \not\models \Delta$. That is, there is an fd $X \rightarrow Y$ in Δ such that $\pi_S(\Sigma^+) \not\models X \rightarrow Y$. If $\Sigma \models X \rightarrow Y$, then $X \rightarrow Y \in \pi_S(\Sigma^+)$, since $XY \subseteq S$ - a contradiction. Thus, $\Sigma \not\models \Delta$, which means that there is a relation I that satisfies Σ but not Δ. But then $\pi_S(I) \in \Omega$ and $\pi_S(I)$ does not satisfy Δ - contradiction. Therefore, $\Omega = SAT_S(\Delta) = SAT(\pi_S(\Sigma^+))$.

Let $R = ABCDE$, let $S = ABCD$, and let Σ consists of the fd's $A \rightarrow E$, $B \rightarrow E$, and $CE \rightarrow D$. It can be verified that $\Delta = \pi_S(\Sigma^+) = \{AC \rightarrow D, BC \rightarrow D\}$. We claim that $SAT_S(\Delta) \neq \pi_S(SAT_R(\Sigma))$.

Consider the relation I on S:

	A	B	C	D
u_1	1	3	5	7
u_2	2	4	5	8
u_3	1	4	6	8

Clearly, I satisfies Δ. Nevertheless, I is not in $\pi_S(SAT_R(\Sigma))$. Suppose it is, that is $I = \pi_S(J)$, where J satisfies Σ. Then there are tuples $v_1, v_2, v_3 \in J$ such that $u_1[S] = v_1[S]$, $u_2[S] = v_2[S]$, and $u_3[S] = v_3[S]$. Since J satisfies $A \rightarrow E$, and $v_1[A] = u_1[A] = u_3[A] = v_3[A]$, we must have $v_1[E] = v_3[E]$. Similarly, since J satisfies $B \rightarrow E$, and $v_2[B] = u_2[B] = u_3[B] = v_3[B]$, we must have $v_2[E] = v_3[E]$. Also $v_1[C] = u_1[C] = u_2[C] = v_2[C]$. But then $v_1[CE] = v_2[CE]$, and, since J satisfies $CE \rightarrow D$, we must have $v_1[D] = v_2[D]$. But then $u_1[D] = u_2[D]$ - which is not the case. ■

In the next section, we define a class of dependencies that generalizes the classes that we have seen so far.

5.6.2 Dependencies

Studying the dependencies that we have defined so far (fd's, mvd's, emvd's, and, informally, inclusion dependencies), we see that they all have a common structure: they say "if you see a certain pattern of tuples in the database, then you must also see this." In the case of fd's, "this" refers to the equality of certain entries, while for the other dependencies, "this" is another tuple that must also be in the database. For example, if $U = ABC$, then the fd $A \rightarrow B$ says that if you see two tuples that agree on A, then

they must also agree on B. The mvd $A \twoheadrightarrow B$ says that if you see two tuples that agree on A, then there should be a tuple that agree with these tuples on A, agree with the first tuple on B, and agree with the second tuple on C. Thus, fd's can be called *equality-generating dependencies*, while mvd's can be called *tuple-generating dependencies*.

We now generalize these ideas and define a general class of semantic constraints, which we simply call *dependencies*. We choose to describe dependencies as first-order sentences as in [BV81,Fa82], rather then use the equivalent formalisms of [BV84] and [YP82]. To use first-order logic, we assume that the attribute are ordered, so we do not have to refer to them explicitly. We start by allowing databases with many relations. The atomic formulas are those that are either of the form $\mathbf{P}x_1 \cdots x_d$, where $\mathbf{P}$ is the name of a d-ary relation and the x_i's are individual variables, or of the form $x=y$, where x and y are individual variables. Formulas of the former type are called *relational formulas* (because they say that a certain tuple exists in the relation), and formulas of the latter type are called *equalities*. A *dependency* is a first-order sentence

$$(\forall y_1 \cdots y_k)(\exists x_1 \cdots x_l)(A_1 \wedge \cdots \wedge A_p \supset B_1 \wedge \cdots \wedge B_q)$$

where the A_i's and B_i's are atomic formulas. To capture our intuition about what this sentences should say, we put more syntactic restrictions. First, we want the dependency to say "if you see a certain pattern of tuple then ... ," so we require all the A_i's to be relational formulas in the variables $y_1, \ldots, y_k$, that all the y_j's occur in the A_i's, and that $p \geq 1$ and $k \geq 1$. We want the B_i's to talk about existence of tuples or about equalities among entries of tuples, so we require that $q \geq 1$. We do not require that $l \geq 1$; if $l=0$, then there are no existential variables.

Example 5.25 The dependency

$$(\forall y y_1 y_2 y_3 y_4)(\exists x x_1 x_2)(\mathbf{R}y y_1 y_2 \wedge \mathbf{R}y y_3 y_4 \supset \mathbf{R}x x_1 x_2 \wedge x=y \wedge x_1=y_2 \wedge x_2=y_4 \wedge y_1=y_2),$$

says that if there are two tuples that agree on the first argument, then there should be a tuple that agrees with these tuples on the first argument, agrees with the first tuple on the second argument, and agrees with the second tuple on the third argument. Note that this dependency can also be written as

$$(\forall y y_1 y_2 y_3 y_4)(\mathbf{R}y y_1 y_2 \wedge \mathbf{R}y y_3 y_4 \supset \mathbf{R}y y_1 y_4 \wedge y_1=y_2).$$

■

Example 5.26 The dependency

$$(\forall y_1 y_2)(\exists x_1)(\mathbf{P}y_1 y_2 \supset \mathbf{R}y_2 x_1)$$

says that if a value occurs as an entry in the second argument of a tuple in the **P** relation, then that value occurs also as the first argument of a tuple in the **R** relation. We have termed such dependencies as *inclusion dependencies*. ■

We now consider classification of dependencies to several subclasses. If the dependency talks about more than one relation, e.g., the dependency in Example 5.26 talks about the relation **P** and the relation **R**, then it is an *interrelational* dependency. If it talks only about one relation, e.g., the dependency in Example 5.25 talks only about the relation **R**, then it is an *intrarelational* or *unirelational* dependency. For simplicity we deal from now on only with unirelational dependencies.

The second classification has to do with the pattern of occurrence of variables in the dependencies. Suppose that the dependency talks about the relation **R**. If no variable occurs in two different argument positions of **R** and we have an equality $y_1=y_2$ only if y_1 and y_2 occur in the same argument position of **R**, then the dependency is *typed*, otherwise it is *untyped*. The intuition behind this classification is that in a typed relation the domains that underly distinct columns are disjoint and constitute distinct *types*. Thus, a typed dependency does not require any interaction between values in different columns.

Example 5.27 The dependency

$$(\forall yy_1y_2y_3y_4)(\mathbf{R}yy_1y_2 \wedge \mathbf{R}yy_3y_4 \supset \mathbf{R}yy_1y_4),$$

is typed. The dependency

$$(\forall y_1y_2)(\exists x_1)(\mathbf{R}y_1y_2 \supset \mathbf{R}y_2x_1)$$

is untyped. Thus, fd's, mvd's, emvd's, and join dependencies are all typed dependencies, while inclusion dependencies are untyped dependencies. ■

The following classification has to do with the structure of the dependencies. If $l=0$, i.e., there are no existential quantifiers, then the dependency is *full*, otherwise it is *embedded* (the term "embedded" is not quite appropriate here, but it is borrowed from emvd's). If all the B_i's are equalities, then the dependency is an *equality-generating dependency* (egd). If all the B_i's are relational formula, then the dependency is a *tuple-generating dependency* (tgd). For example, an fd is a full egd, an emvd is a tgd, and an mvd is a full tgd.

The above forms can be viewed as certain syntactical normal forms.

Theorem 5.27

(1) A (typed) dependency is logically equivalent to a set of (typed) egd's and tgd's.

(2) A (typed) full dependency is logically equivalent to a set of (typed) full egd's and tgd's. ■

Thus, from now on, we use the term "dependencies" to refer to egd's and tgd's.

We conclude this section by observing that egd's have the expressive power that fd's lack. We define egd-class analogously to fd-class. That is, a class Ψ of relations is an egd-class if there is a set Σ of egd's such that $\Psi = SAT(\Sigma)$.

Theorem 5.28 [Fa82] The projection of an egd-class is an egd-class. ■

Note that since every fd-class is an egd-class, it follows from the theorem that the projection of an fd-class is an egd-class.

5.6.3 The Implication Problem

We can now recast the discussion of §3.3 in more general terms: for full dependencies the (finite) implication problem reduces to the (finite) unsatisfiability problem of the BS class.

Theorem 5.29 For full dependencies, implication and finite implication coincide, and the corresponding decision problems are decidable. ■

Following our standard course, we should now come up with a formal system for full dependencies, and then with an efficient implication testing algorithm. A formal system was indeed developed in [BV84], but unlike the case with fd's and mvd's, that formal system does not lead to an efficient decision procedure. The best decision procedure for testing implication of full dependencies runs in exponential time. Surprisingly, this bound is the best possible.

Theorem 5.30 [CLM81] The implication problem for typed full dependencies is EXPTIME-complete. ■

A problem is EXPTIME-complete if it can be solved in exponential time and it is also as hard as any problem that can be solved in exponential time. Since it is known that there are problems that can be solved in exponential time and do require exponential time, it follows that EXPTIME-complete problems require exponential time. Thus, an EXPTIME-completeness is a proof that the problem is intractable. In contrast, NP-completeness is a strong suggestion rather than a proof that the problem is intractable, since it proves intractability only under the assumption that there are problems that can be solved in nondeterministic polynomial time but not in deterministic polynomial time.

The reduction of (finite) implication to (finite) unsatisfiability of the BS class depends crucially on the fact that full dependencies are universal sentences, and therefore, the reduction does not hold for embedded dependencies.

Theorem 5.31 [BV81] For embedded dependencies, implication and finite implication differ.

Proof Let **R** be a binary relation name. Let Σ consists of the dependency σ_1:

$$\forall y_1 y_2 \exists x (\mathbf{R} y_1 y_2 \supset \mathbf{R} y_2 x)$$

and σ_2:

$$\forall y_1 y_2 y_3 (\mathbf{R} y_1 y_2 \wedge \mathbf{R} y_2 y_3 \supset \mathbf{R} y_1 y_3).$$

σ_1 says that **R** is *serial*, i.e., every node has an outgoing edge. σ_2 says that **R** is a *transitive* relation. Let σ be the dependency

$$\forall y_1 y_2 \exists x (\mathbf{R} y_1 y_2 \supset \mathbf{R} x x).$$

σ says that if **R** is nonempty, then it must be *reflexive* on some node.

We claim that $\Sigma \not\models \sigma$ but $\Sigma \models_f \sigma$. We first show that $\Sigma \not\models \sigma$. Let **R** be interpreted by the binary relation $I = \{(i,j) : i \leq j\}$. It is easy to check that I satisfies Σ but not σ. Suppose, however, that **R** is interpreted by a finite nonempty binary relation I and that I that satisfies Σ. Because of σ_1, the relation I must contain a cycle, i.e., a sequence of nodes $a_1, \ldots, a_n$ such that $(a_i, a_{i+1}) \in I$, for $1 \leq i \leq n-1$, and also $(a_n, a_1) \in I$. But because of σ_2, we must have $(a_1, a_1) \in I$, so I satisfies σ. ■

Theorem 5.31 suggests that the (finite) implication problem for dependencies might be undecidable, which is indeed the case. (Note, however, that there are classes of dependencies for which implication and finite implication differ, but both the implication problem and the finite implication problem are decidable [KCV83].)

Theorem 5.32 [BV81,CLM81] The implication and the finite implication problems for dependencies are undecidable. ■

Well, so much for our hope to prove decidability for emvd's by extending the class of dependencies. But all may not be lost yet. It is still conceivable that we can find a decidable class of dependencies that that contains the class of emvd's. Let us look more closely at the first-order syntax of emvd's (see §5.7). Emvd's have the following four properties:

(1) they are typed,

(2) they are tgd's,

(3) they have a single relational formula on the right-hand side of the implication, and

(4) they have at most two formulas on the left-hand side of the implication.

Dependencies that satisfy properties (1)-(3) are called *template dependencies* [SU82]. The class of template dependencies seems to be quite a natural class; in fact, it is the smallest class of dependencies that contains the class of emvd's and is known to have a sound and complete formal system [BV84,SU82]. Thus, one might hope that this class is decidable. Unfortunately, this is not the case.

Theorem 5.33 [GL82,Va84a] The implication and finite implication problems for template dependencies are undecidable. ■

In fact, Vardi [Va84a] proved undecidability for even a smaller class of dependencies, the class of *projected join dependencies*. Nevertheless, the implication problem for emvd's remains tantalizingly open.

5.6.4 Global Decision Problems

So far we have concentrated almost exclusively on the implication problem. Our interest in the implication problem was originally, however, rather secondary. Our primary interest was in properties of sets of dependencies, such as equivalence and redundancy, which happen to reduce to implication. Now that we know that the implication problem is undecidable, we have to reconsider equivalence and redundancy. For simplicity we consider here only finite relations. Recall that two sets Δ and Σ of dependencies are *equivalent* if they are satisfied by the same relations, and a set Σ of dependencies is *redundant* if there is a proper subet Δ of Σ such that Δ and Σ are equivalent. The *redundancy problem* is to determine whether a given finite set of dependencies is redundant (we concentrate on redundancy, since undecidability of equivalence follows easily from undecidability of implication).

More generally, we would like to be able to check other properties of sets of dependencies. This gives rise to decision problems that we call *global*, to contrast it with the implication problem, which we view as a *local* decision problem. (This terminology is borrowed from the theory of finitely presented groups [Bo68]. The problem whether two words in a finitely presented group are equal is a local problem, while the problem whether a finitely presented group is, say, simple is global.) Our interest in global decision problem comes from the fact that we view a set of dependencies as a semantic specification for a database. The ability to recognize properties of such specifications seems to be essential to the task of verifying their consistency and correctness. We now describe several global properties of interest. We assume that the language contains a relation name **R** of some unspecified arity, and that **R** will be interpreted by a relation I.

A relation I is *trivial* if $I=\{(a,a,\ldots,a)\}$ for some element a, that is, I consists of a single tuple with the same entries in all columns. It is easy to verify that a trivial relation satisfies all dependencies. A set Σ is *inconsistent* if a relation I satisfies Σ only if I is trivial. If Σ is inconsistent, then it is probably not a meaningful semantic specification. (In general, inconsistency means having no model. But, since

every set of dependencies is satisfiable by the trivial relation, we define inconsistency as having no nontrivial models.)

Let $\Sigma^+ = \{\sigma : \Sigma \models_f \sigma\}$, i.e., Σ^+ is the set of dependencies implied by Σ. Σ is *complete* if $\Sigma \cup \{\sigma\}$ is inconsistent for all $\sigma \notin \Sigma^+$. If Σ is complete, then there is no point in trying to extend it, since every dependency σ is either a consequence of Σ, or yields an inconsistent specification if added to Σ.

For any relation I, let $DEP(I)$ be the set of all dependencies satisfied by I, i.e., $DEP(I) = \{\sigma : I \text{ satisfies } \sigma\}$. I is an *Armstrong relation* for a set Σ of dependencies, if $DEP(I) = \Sigma^+$. Σ is *Armstrong* if it has an Armstrong relation. (This terminology, [Fa82,Mak81], is suggested by the approach in [Arm74].) The motivation for this property is as follows. Suppose that I satisfies Σ. In general, I satisfies not only Σ^+, but also other dependencies as well. These dependencies are satisfied "accidentally." If, on the other hand, I is an Armstrong relation for Σ, then it does not satisfy any "accidental" dependencies. Thus, I can be viewed as a representative instance for the collection of databases specified by Σ, namely $SAT(\Sigma)$. Such representative instances seems to be useful in the process of database design [SM81,MR85].

The last property that we consider is *decidability*. Undecidability of the implication problem means that the set $\{(\Sigma,\sigma) : \Sigma \models_f \sigma\}$ is not recursive. It is possible, however, that for a particular set Σ of dependencies, the set Σ^+ is recursive. This means that for this particular Σ the implication problem is decidable, i.e., we can check whether $\Sigma \models_f \sigma$ for any given σ. In this case we say that Σ is *decidable*, clearly, a desirable property.

A property P *implies* a property Q, if P is a subset of Q, i.e., if a set Σ of dependencies is P, then it is also Q. The relationship between the above defined properties is as follows.

Theorem 5.34 Inconsistency implies completeness, which implies Armstrongness, which implies decidability.

Proof

(1) Suppose that Σ is inconsistent. Then it has only trivial models. But the trivial relation satisfies all dependencies, so Σ^+ is the set of all dependencies. Thus, the condition of completeness is satisfied vacuously.

(2) Suppose that Σ is complete. We have to consider two cases. First, Σ might be inconsistent. In that case, Σ^+ is the set of all dependencies, so any trivial relation is an Armstrong relation for Σ. If Σ is not inconsistent, then it has a nontrivial model, i.e., there is a nontrivial relation I such that I satisfies Σ. We claim that I is an Armstrong relation for Σ. Indeed, suppose that I

satisfies σ, but σ is not in Σ^+. Since I is nontrivial and it satisfies $\Sigma \cup \{\sigma\}$, it follows that $\Sigma \cup \{\sigma\}$ is consistent, which contradicts the completeness of Σ.

(3) Suppose that Σ is Armstrong. Then there is a relation I such that $DEP(I) = \Sigma^+$. Thus, σ is in Σ^+ if and only if I satisfies σ. Since it is decidable whether I satisfies σ, it follows that Σ^+ is recursive. ■

In general, a *property* P is a set of finite sets of dependencies. We usually say that a set Σ *is* P instead of saying that it is *in* P. For example, the property of completeness is the set of all complete finite sets of dependencies. Of course, we would like our properties to be *decidable*, i.e., recursive. We are going to state a general negative result about decidability of properties, but we need first some definitions.

A property P is *trivial* if either all finite sets of dependencies are P or none is. Clearly, only nontrivial properties are of interest. The following definition is inspired by Theorem 5.34. We say that a property P is *well-behaved* if it contains the property of inconsistency, i.e., if every inconsistent set is P. For example, decidability, Armstrongness, completeness, and, of course, inconsistency are all well-behaved properties.

To define the next notion we need to redefine the operation of *projection* in accordance with our current convention of ordered columns. Let I be an n-ary relation, and let $i_1, \ldots, i_k$ be a sequence such that $1 \le i_1 < \cdots < i_k \le n$. The *projection* of I on the arguments $i_1, \ldots, i_k$ is

$$\pi_{<i_1, \ldots, i_k>}(I) = \{(a_{i_1}, \ldots, a_{i_k}) : (a_1, \ldots, a_n) \in I\}.$$

Let Σ be a set of dependencies on an n-ary relation **R**, and let Δ be a set of dependencies on a k-ary relation **P**. We say that Δ is a *projection* of Σ if there is a sequence $1 \le i_1 < \cdots < i_k \le n$ such that $SAT(\Delta) = \pi_{<i_1, \ldots, i_k>} SAT(\Sigma)$. In other words, Δ is a projection of Σ if it is the specification of the projections of the relations specified by Σ. We say that a property P is *hereditary* if it is inherited by projections, that is, if Σ is P and Δ is a projection of Σ, then Δ is also P.

We can now state the general result.

Theorem 5.35 [Va81] Let P be a nontrivial, well-behaved, and hereditary property. Then P is undecidable. In particular, inconsistency, completeness, Armstrongness, and decidability are undecidable. ■

Looking back at the properties studied here, we see that redundancy is different from, say, completeness. Completeness is a semantic property; that is, if Σ is complete and Δ is equivalent to Σ, then Δ is also complete. This is not the case with redundancy, which is a syntactic property. In particular,

redundancy is not an hereditary property, so it is not covered by Theorem 5.35. Nevertheless, the reader should not get too hopeful.

Theorem 5.36 [Va81] Redundancy is undecidable. ■

5.7 Concluding Remarks

We have gone through a whole span of dependency theory, from the "meek" functional dependencies, for which most decision problems are efficiently solvable, to the general "mean" family of dependencies, for which almost nothing of interest seems to be solvable. Our voyage have been motivated by the desire to automate the process of database design. So what is the moral?

Perhaps it would be instructive to draw an analogy with another discipline. Twenty years ago it was widely believed that a powerful and efficient proof procedure that can create logical demonstrations would be a major step in getting a machine to behave intelligently. Years passed, the desired proof procedure continued to elude researchers, and this approach was rendered naive and simplistic. Current research in artificial intelligence is detailed and nitty-gritty rather than vague and general.

We believe that the hope of fully automating the process of database design is similarly naive and simplistic. Modelling the real world is an immensely complicated task for which perhaps no elegant algorithm exists. One should view the theoretical foundations that have been laid as a basis on which to develop heuristics and practical methodologies.

Acknowledgements I am grateful to Ron Fagin and Pierre Wolper for many valuable comments on previous drafts of this article.

REFERENCES

[ABU79] Aho, A.V., Beeri, C., Ullman, J.D., "The theory of joins in relational databases," *ACM Trans. on Database Systems* 4(1979), pp. 297-314.

[Arm74] Armstrong, W.W., "Dependency structure in data base relationships," *Proc. IFIP 74*, North-Holland, 1974, pp. 580-583.

[BB79] Beeri, C., Bernstein P.A., "Computational problems related to the design of normal form relational schemas," *ACM Trans. on Database Systems* 4(1979), pp. 30-59.

[BBG78] Beeri, C., Bernstein, P.A., Goodman, N., "A sophisticate's introduction to database normalization theory," *Proc. 4th Int'l. Conf. on Very Large Data Bases*, Berlin, 1978, pp. 113-124.

[BDB79] Biskup, J., Dayal, U., Bernstein, P.A., "Synthesizing independent database schemas," *Proc. ACM-SIGMOD Int'l Conf. on Management of Data*, 1979, pp. 143-152.

[Bee80] Beeri, C., "On the membership problem for multivalued dependencies," *ACM Trans. on Database Systems* 5(1980), pp. 241-259.

[Bee85] Beeri, C., Private communication.

[Ber75] Bernstein, P.A., "*Normalization and functional dependencies in the relational model*," Ph.D. Dissertation, Tech. Rep. CSRG-60, University of Toronto, 1975.

[Ber76] Bernstein, P.A., "Synthesizing third normal form relations from functional dependencies," *ACM Trans. on Database Systems* 1(1976), pp. 277-298.

[BFH77] Beeri, C., Fagin, R., Howard, J.H., "A complete axiomatization for functional and multivalued dependencies in database relations," *Proc. ACM-SIGMOD Int'l Conf. on Management of Data*, Toronto, 1977, pp. 47-61.

[Bi78] Biskup, J., "On the complementation rule for multivalued dependencies in data base relations," *Acta Informatica* 10(1978), pp. 297-305.

[Bi80] Biskup, J., "Inferences of multivalued dependencies in fixed and undetermined universe," *Theoretical Computer Science* 10(1980), pp. 93-105.

[BMSU81] Beeri, C., Mendelzon, A.O., Sagiv, Y, Ullman, J.D., "Equivalence of relational database schemes," *SIAM J. Comput.* 10(1981), pp. 647-656.

[Bo68] Boone, W.W., "Decision problems about logical systems as a whole and recursively enumerable degrees of unsolvability," In *Contributions to Mathematical Logic* (H.A. Schmidt, K. Schutte, and H.J. Thiele, eds.), North-Holland, 1968, pp. 13-33.

[BV81] Beeri, C., Vardi, M.Y., "The implication problem for data dependencies," *Proc. 8th Int'l Colloq. on Automata, Language, and Programming*, Acre, Israel, 1981, Lecture Notes in Computer Science 115, Springer-Verlag, 1981, pp. 73-85.

[BV84] Beeri, C., Vardi, M.Y., "Formal system for tuple and equality generating dependencies," *SIAM J. Computing* 13(1984), pp. 76-98.

[CLM81] Chandra, A.K., Lewis, H.R., Makowsky, J.A., "Embedded implicational dependencies and their inference problem," *Proc. 13th ACM Ann. Symp. on Theory of Computing*, 1981, pp. 342-354.

[Co70] Codd, E.F., "A relational model of data for large shared data banks," *Comm. ACM* 13(1970), pp. 377-387.

[Co72a] Codd, E.F., "Further normalization of the data base relational model," in *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, N.J., 1972, pp. 33-64.

[Co72b] Codd, E.F., "Relational completeness of database sublanguages," in *Data Base Systems* (R. Rustin, ed.), Prentice-Hall, N.J., 1972, pp. 65-98.

[Co74] Codd, E.F., "Recent investigations in relational database systems," *Proc. IFIP 74*, North-Holland, 1974, pp. 1017-1021.

[De78] Delobel, C., "Semantics of relations and the decomposition process in the relational data model," *ACM Trans. on Database Systems* 3(1978), pp. 201-222.

[DG79] Dreben, B.S., Goldfarb, W.D., "*The decision problem - solvable classes of quantificational formulas*," Addison Wesley, 1979.

[Fa76] Fagin, R., "*Multivalued dependencies and a new normal form for relational databases*," IBM Research Report RJ1812, IBM Research Laboratory, San Jose, 1976.

[Fa77a] Fagin, R., "Multivalued dependencies and a new normal form for relational databases," *ACM Trans. on Database Systems* 2(1977), pp. 262-278.

[Fa77b] Fagin, R., "Functional dependencies in a relational database and propositional logic," *IBM J. Research and Development* 21(1977), pp. 534-544.

[Fa81] Fagin, R., "A normal form for relational databases that is based on domains and keys," *ACM Trans. on Database Systems* 6(1981), pp. 387-415.

[Fa82] Fagin, R., "Horn clauses and database dependencies," *J. ACM* 29(1982), pp. 952-985.

[FV86] Fagin, R., Vardi, M.Y., "The theory of data dependencies - a survey," in *Mathematics of Information Processing*, Proc. Symp. in Applied Mathematics 34, American Mathematical Society, 1986, pp. 19-72.

[Ga82] Galil, Z., "An almost linear-time algorithm for computing a dependency basis in a relational database," *J. ACM* 29 (1982), pp. 96-102.

[GJ79] Garey, M.R., Johnson, D.S., "*Computers and intractability - a guide to the theory of NP-completeness*," Freeman, San Francisco, 1979.

[GL82] Gurevich, Y., Lewis, H.R., "The inference problem for template dependencies," *Proc. ACM Symp. on Principles of Database Systems*, Los Angeles, 1982, pp. 221-229.

[GZ82] Ginsburg, S., Zaiddan, S.M., "Properties of functional-dependency families," *J. ACM* 29(1982), pp. 678-698.

[He71] Heath, I.J., "Unacceptable file operations in a relational database," *Proc. ACM-SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, 1971.

[ITK83] Ito, M., Taniguchi, K., Kasami. T., "Membership problem for embedded multivalued dependencies under some restricted conditions," *Theoretical Computer Science* 23(1983), pp. 175-194.

[Ka72] Karp, R.M., "Reducibility among combinatorial problems," In *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85-104.

[KCV83] Kannelakis, P.C., Cosmadakis, S.S., Vardi, M.Y., "Unary inclusion dependencies have polynomial-time inference problems," *Proc. 15th ACM Symp. on Theory of Computing*, Boston, 1983, pp. 264-277.

[Le80] Lewis, H.R., "Complexity results for classes of quantificational formulas," *J. Computer and System Sciences* 21(1980), pp. 317-353.

[Li81] Lien, Y.E., "Hierarchical schemata for relational databases," *ACM Trans. on Database Systems* 6(1981), pp. 48-69.

[LO78] Lucchesi, C.L., Osborn, S.L., "Candidate keys for relation," *J. Computer and System Sciences* 17(1978), pp. 270-279.

[Ma80] Maier, D., "Minimum covers in the relational database model," *J. ACM* 27(1980), pp. 664-674.

[Ma83] Maier, D., "*The theory of relational databases*," Computer Science Press, Rockville, 1983.

[Mak81] Makowsky, J.A., "Characterizing database dependencies," *Proc. 8th Int'l Colloq. on Automata, Language, and Programming*, Acre, Israel, 1981, Lecture Notes in Computer Science 115, Springer-Verlag, 1981, pp. 86-97.

[Me79] Mendelzon, A.O., "On axiomatizing multivalued dependencies in relational databases," *J. ACM* 26(1979), pp. 37-44.

[MR85] Mannila, H., Raiha, K.J., "Small Armstrong relations for database design," *Proc. 4th ACM Symp. on Principles of Database Systems*, Portland, 1985, pp. 245-250.

[Ni78] Nicolas, J.M., "First order logic formalization for functional, multivalued and mutual dependencies," *Proc. ACM-SIGMOD Int'l Conf. on Management of Data*, 1978, pp. 40-46.

[PP80] Parker, D.S., Parsaye-Ghomi, K., "Inference involving embedded multivalued dependencies and transitive dependencies," *Proc. ACM-SIGMOD Int'l Conf. on Management of Data*, 1980, pp. 52-57.

[Ri77] Rissanen, J., "Independent components of relations," *ACM Trans. on Database Systems* 2(1977), pp. 317-325.

[Ri78] Rissanen, J., "Theory of relations for databases - a tutorial survey," *Proc. 7th Symp. on Math. Found. of Computer Science*, Poland, 1978, Lecture Notes in Computer Science 64, Springer-Verlag, pp. 537-551.

[Ro67] Rogers, H., "*Theory of recursive functions and effective computability*," McGraw-Hill, 1967.

[SDPF81] Sagiv, Y., Delobel, C., Parker, D.S., Fagin, R., "An equivalence between relational database dependencies and a subclass of propositional calculus," *J. ACM* 28(1981), pp. 435-453.

[SM81] Silva, A.M., Melkanoff, M.A., "A method for helping discover the dependencies of a relation," In *Advances in Database Theory* (H. Gallaire, J. Minker, and J.M. Nicolas, eds.), Plenum Press, 1981, pp. 115-133.

[SU82] Sadri, F., Ullman J.D., "A complete axiomatization for a large class of dependencies in relational databases," *J. ACM* 29(1982), pp. 363-372.

[SW82] Sagiv, Y., Walecka, S., "Subset dependencies as an alternative to embedded multivalued dependencies," *J. ACM* 29(1982), pp. 103-117.

[Ul82] Ullman, J.D., "*Principles of database systems*," Computer Science Press, Rockville, 1982.

[Va81] Vardi, M.Y., "Global decision problems for relational databases," *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, Nashville, 1981, pp. 198-202.

[Va84a] Vardi, M.Y., "The implication and the finite implication problems for typed template dependencies," *J. Computer and System Sciences* 28(1984), pp. 3-28.

[Va84b] Vardi, M.Y., "A note on lossless database decomposition," *Information Processing Letters* 18(1984), pp. 257-260.

[YP82] Yannakakis, M., Papadimitriou, C., "Algebraic dependencies," *J. Computer and System Sciences* 21(1982), pp. 2-41.

[Za76] Zaniolo, C., "*Analysis and design of relational schemata for database systems*," Technical Report UCLA-ENG-7769, Department of Computer Science, UCLA, July 1976.

[ZM81] Zaniolo, C., Melkanoff, M.A., "On the design of relational database schemata," *ACM Trans. on Database Systems* 6(1981), pp. 1-47.