A Simplified Universal Relation Assumption and Its Properties

RONALD FAGIN
IBM Research Laboratory
ALBERTO O. MENDELZON
IBM Research Center
and
JEFFREY D. ULLMAN
Stanford University

One problem concerning the universal relation assumption is the inability of known methods to obtain a database scheme design in the general case, where the real-world constraints are given by a set of dependencies that includes embedded multivalued dependencies. We propose a simpler method of describing the real world, where constraints are given by functional dependencies and a single join dependency. The relationship between this method of defining the real world and the classical methods is exposed. We characterize in terms of hypergraphs those multivalued dependencies that are the consequence of a given join dependency. Also characterized in terms of hypergraphs are those join dependencies that are equivalent to a set of multivalued dependencies.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; *trees*; H.2.1 [**Database Management**]: Logical Design—*normal forms*; *schema and subschema*; H.3.3. [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation*

General Terms: Algorithms, Design, Languages, Management, Theory

Additional Key Words and Phrases: Acyclic, hypergraph, database scheme, relational database, multivalued dependency, join dependency

1. INTRODUCTION

This paper is an attempt to deal with a number of issues concerning the "universal relation" concept from relational database theory. In particular, we discuss the following questions.

(1) Relational database design theory, as in [5, 25], for example, requires us to write down our assumptions regarding the real world in terms of functional

This work was partially supported by Air Force grant AFSOR-80-0212 supplemented by NSF agreement 1ST-80-21358.

Authors' present addresses: R. Fagin, IBM Research Laboratory, San Jose, CA 95193; A. O. Mendelzon, Department of Computer Science, University of Toronto, Toronto, Ont., Canada; J. D. Ullman, Stanford University, Stanford, CA 94305.

dependencies and multivalued dependencies. While it seems easy enough for the database designer to perceive and list the functional dependencies, multivalued dependencies are harder to write down correctly. Worse, the standard database design strategies may require us to deal with and infer new embedded multivalued dependencies from those given, and there is no known algorithm for such inferences. Could there be another way to describe the real world that is better tuned to what the designer can perceive or that leads to a simpler design theory?

(2) There are some hints [15, 16, 24] that not every collection of functional dependencies and multivalued dependencies describes a real world that could possibly be of interest to a database designer. Lien especially [15, 16] has taken more traditional forms of database description, such as Bachman diagrams and entity-relationship diagrams, and shown that reasonable diagrams of these sorts correspond to universal relations with special properties. If we suppose that earlier tools for database design, such as Bachman diagrams, were well tuned to the situations encountered in practice, might there not be a sufficiently general class of real worlds in the universal relation sense, for which database design can be done simply?

In answer to these questions we offer the following. Section 2 offers our class of real worlds, those that can be defined by predicates and functional dependencies alone. We feel this class plays a role analogous to the LALR(1) grammars among context-free languages in general (see [2] for a discussion of such grammars). Just as the LALR(1) grammars are sufficiently general to describe the syntax of essentially every known programming language feature and yet are sufficiently special that they have efficient parsers, we believe our class of real worlds is sufficiently general to cover most matters of interest to database designers and yet is sufficiently specific that database description and design are far more manageable than when the real world is described by functional dependencies and multivalued dependencies. This contention will no doubt itself cause controversy; yet we are sufficiently confident of its usefulness that we encourage the reader to search for a description of a *plausible* real world in standard relational terms, or any other terms, that cannot be modeled in the terms to be presented here.

Section 3 discusses database design in the new terms, and Section 4 relates our real-world descriptions to descriptions by multivalued dependencies. Particularly, we give an easy algorithm for finding the set of multivalued dependencies that follow logically from our description of the real world, and we characterize those of our real worlds that can be replaced by a set of multivalued dependencies and functional dependencies.

2. DEFINING UNIVERSAL RELATIONS BY PREDICATES

When doing database designs from a universal relation scheme, as in [10, 4, 25], for example, there is the implication that the selected relation schemes will, in the actual database, be given relations that are the projection of some universal relation.[1] Even though we reject this pure universal relation assumption, if we

[1] See [25] for the definition of the relational database terms used in this paper.

are to use the universal relation concept as a tool in selecting the relation schemes, there must be a universal relation, possibly with nulls, that we regard as reflective of the real world. Let us consider how we might define such a universal relation.

Example 1. Suppose our attributes in the universal scheme are C(ourse), T(eacher), R(oom), H(our), S(tudent), and G(rade). An informal description of what these attributes might mean is that T teaches course C, course C meets in room R at hour H, and S is getting grade G in C. If we had to define a universal relation over these attributes, we would therefore write

$$\{ctrhsg \mid t \text{ "teaches" } c,\ c \text{ "meets in" } r \text{ "at hour" } h, \text{ and } s \text{ "is getting" } g \text{ "in" } c\}.$$

In this definition we have used three undefined relationships, or predicates, that we presume have meaning in the real world: "teaches," "meets in ... at hour," and "is getting ... in." We do not insist that each c, t, r, h, s, and g take on real values in each tuple of the universal relation. For example, in some tuple, t could be a null standing for "the teacher of c," even if we did not know the correct value. Thus we avoid Codd's "anomalies" [12] in the universal relation and achieve a degree of "normalization." □

Our definition in Example 1 of the universal relation that reflects the current real world made use of three abstract predicates, such as "teaches," which we may view as "filters" that look at a specified set of components in each tuple and say whether or not these components reflect the current real world. The universal relation is the set of all and only those tuples that pass the test implied by each predicate.

While we do not know what the real world will say about various entities, for example, whether or not it is true that course CS101 meets Friday 9AM in room 321, we can assume that there are some three predicates $P_1(c, t)$, $P_2(c, h, r)$, and $P_3(c, s, g)$ that truly tell whether teacher t is teaching course c, and so on. The fact that legal universal relations for Example 1 must be of the form

$$\{ctrhsg \mid P_1(c, t) \text{ and } P_2(c, h, r) \text{ and } P_3(c, s, g)\}$$

for some predicates P_1, P_2, and P_3 puts a severe constraint on what universal relations may be. However, this constraint is not so severe as the general collections of multivalued dependencies that we imagine might be used to define universal relations. (We consider the role of functional dependencies momentarily.) In fact, we at once show that definition by predicates is equivalent to the requirement that the universal relation satisfy one join dependency.

Given a collection of predicates $\{P_1, \ldots, P_n\}$, each of which is defined over a subset of the attributes in some universal set U, we say relation r over relation scheme U is the *relation defined by* $P_1, \ldots, P_n$ if r consists of exactly those tuples that satisfy all these predicates. Thus, in the example above, each possible value of P_1, P_2, and P_3 defines a relation over the relation scheme *CTRHSG* in the way expressed by the set-formers shown above.

Recall that a join dependency [21], written $\bowtie(R_1, \ldots, R_k)$, is the requirement on relations r over relation scheme $R = R_1 \cup \cdots \cup R_k$ that the natural join of the projections of r onto the R_i's be r. If we abuse notation to the extent of identifying the name of a predicate P with the set of attributes over which P acts as a filter, then we can treat predicate names as if they were the R_i's (which we shall, following [23], call *objects*) of a join dependency. That is, we can state the following theorem.

THEOREM 1. *A relation r over attributes* $P_1 \cup \cdots \cup P_k$ *can be the relation defined by some values of the predicates* $P_1, \ldots, P_k$ *if and only if r satisfies the join dependency* $\bowtie(P_1, \ldots, P_k)$.

PROOF. *If:* Suppose r satisfies the join dependency, and let $p_i = \pi_{P_i}(r)$ for $i = 1, \ldots, k$. (π_S stands for the projection onto set of attributes S.) If we take p_i to be the value of predicate P_i, that is, the set of tuples for which the predicate is true, then surely every tuple in r satisfies the condition P_1 and $\cdots$ and P_k. Conversely, any tuple t that satisfies the predicates is such that $\pi_{P_i}(t)$ is in p_i. But then the join dependency implies that t is in r. Thus $r = \{t \mid P_1 \text{ and } \cdots \text{ and } P_k\}$ when we select the value p_i for predicate P_i, $i = 1, \ldots, k$.

Only if: Suppose r is the relation defined by the predicates when P_i has the value p_i for all i. Suppose t_i is in $\pi_{P_i}(r)$ for $i = 1, \ldots, k$, and let t be the natural join of the t_i's. Then t_i must satisfy P_i, else it could not be the projection of a tuple in the relation defined by the predicates. It follows that t satisfies all the predicates and is therefore in r. But since the t_i's were arbitrary projections of tuples in r, we have shown that r satisfies the join dependency. $\square$

COROLLARY 1. *A relation is defined by predicates* $P_1, \ldots, P_k$ *if and only if it is a fixed point of* $\{P_1, \ldots, P_k\}$ *in the sense of* [8].

2.1 Functional Dependencies

There are, in most situations, certain functional dependencies that hold, and these cannot be reflected in the definition of the universal relation by predicates in the style we have introduced above. Rather, let us say that the functional dependencies influence one or more predicates, in the sense that they may make it impossible for certain values of the predicates to reflect any instance of the real world. For example, if we assert the functional dependency $C \to T$ in our real world of Example 1, then we would not expect that the CT predicate would say yes both to (CS101, Jones) and (CS101, Smith). Functional dependencies may have subtle influences, ranging over several predicates, since it is not guaranteed that the attributes involved in a functional dependency will necessarily be checked as a group by any one predicate.

Example 2. In Example 1 we might assert the following five functional dependencies, following [26].

$$C \to T,\ TH \to R,\ SH \to R,\ HR \to C, \text{ and } CS \to G$$

The three predicates, CT, CHR, and CSG, embody (contain all the attributes in) the first, fourth, and fifth dependencies, respectively, but the second and third are "interrelational."

If one thinks about the universal relation produced by the alternative sets of predicates, it becomes apparent that we do not want to add clauses to the definition of the universal relation to the effect that "teacher t is in room r at hour h," and "student s is in room r at hour h." Intuitively, there are two kinds of functional dependencies. One, which we call *data structuring*, has a set of attributes that is coextensive with an object. These functional dependencies, such as $C \rightarrow T$, not only assert a uniqueness fact, but they tell us about a relationship that influences how we structure data in the underlying database. The second kind of functional dependency, which we shall call *incidental*, asserts a uniqueness fact, but does not apparently relate to the way data should be structured. The functional dependencies $SH \rightarrow R$ and $TH \rightarrow R$ are in this class. That is, although they express the physical fact that students and teachers cannot be in two places at once, they should have no bearing on how we store our data. There is no practical reason why we should store the locations of students and teachers at each hour explicitly (even though the information is deducible). □

Thus it appears that in general, our specification of the universal relation will include a definition by predicates, that is, the join dependency that the universal relation must satisfy, plus a collection of functional dependencies, some of which will be checked by the predicates and others that will not, but which will nevertheless influence the simultaneous values the predicates may assume. It is open to conjecture whether functional dependencies not embodied in a predicate play any useful role in database design. Let us observe that just because a functional dependency is true does not obligate us to check it or to design for it.

2.2 Hypergraph Representation of Join Dependencies

A portrayal of join dependencies that will prove useful later is the hypergraph. Formally, a *hypergraph* is a set of nodes and a set of hyperedges (we shall call them just "edges") which are nonempty sets of nodes. Hypergraphs are generalizations of ordinary graphs, in that a graph is a hypergraph in which every edge consists of exactly two nodes. To relate hypergraphs to join dependencies, let us use one node for each attribute, and let the edges be the objects of the join dependency, that is, the sets of attributes onto which projections are required.

Example 3. Assume a universal relation scheme with attributes Bank, Acct, Loan, Customer, Addr (address of customer), Bal (balance of account), and Amt (amount of loan). A natural definition for the universal relation over all seven of these attributes is

$\{a_1, \ldots, a_7 \mid a_2$ is an account at bank a_1, a_3 is the balance for a_2, a_4 is a loan made by bank a_1, a_5 is the amount of loan a_4, a_6 is a customer holding account a_2, a_6 is the customer borrowing loan a_4, and a_7 is the address of $a_6\}$.

The corresponding hypergraph is in Figure 1. Here we have only "ordinary" edges, that is, sets of two nodes, and we show these edges by lines in the usual way. Edges of other than two nodes are illustrated by dashed lines surrounding their nodes.

Although legal, there may be some problem with the use of attribute Customer in two different senses in the above definition of a universal relation. That is, if a query requests those banks associated with a given customer, it is not clear whether what is desired is the set of banks where the customer has loans, or the

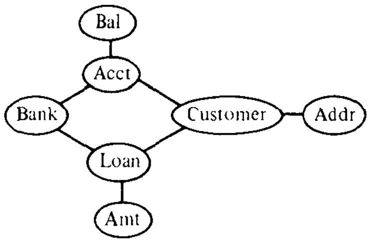

Fig 1. Hypergraph for banking example.

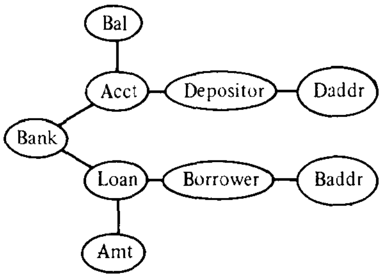

Fig. 2. Revised banking example with no cycles.

set of banks where the customer has accounts, or both.[2] In Section 4 we formally define the notion of cycles in a hypergraph, but intuitively, the cycle in Figure 1 should be apparent. We conjecture that it is possible in the real world always to define the universal relation so that cycles are absent (by renaming in some "natural" way). For example, Figure 2 shows the hypergraph for a database scheme similar to Figure 1, but that contains no cycles. Whether we wish always to do so is more doubtful. □

2.3 Other Examples of Universal Relation Definitions

Let us exhibit the hypergraphs for some other universal relations that illustrate additional points. First, we have not yet seen a case where an edge is other than the standard pair of nodes. The Courses, Teachers, etc., world from Example 1 provides such a situation.

Example 4. In Figure 3 we see the hypergraph for Example 1. Let us remark that this hypergraph is acyclic, as we shall see in Section 4 when we have a formal definition of "cycle" to match our intuition. □

Example 5. The next example is taken from [15], where it illustrates what is there called a "cyclic" Bachman diagram. Lien describes a database with information about employees, their children, their departments, the projects they work for (projects are unrelated to departments), the parts needed by projects, and the suppliers of parts for projects. Figure 4 shows an initial attempt at describing this information by a universal relation.

The role of the Project-Part-Supplier (*PrPaS*) predicate is threefold. First, it checks triples (p, a, s) to see if supplier s is currently supplying part a to project p. But the intent of Lien's example is that we should also record information about what parts a supplier *could* supply, and what parts a project needs, even though no one may be supplying the part currently.

[2] The problem as we see it is that the attribute Customer is "overloaded." It really should be two attributes, Depositor and Borrower. In that case, a query asking for Jones' bank would have to specify whether it meant Jones qua depositor or qua borrower. Note that the requirement that Customer be split into two attributes is similar to the requirement for attribute splitting that comes from the assumption of uniqueness of functional dependencies [10, 4]. Yet the latter does not formally require the attribute Customer to be split, since there is no functional relationship between Customer and Bank.

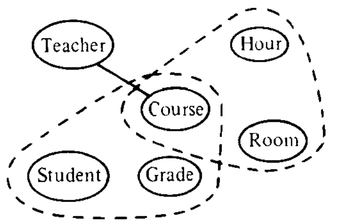

Fig. 3. Hypergraph for Example 1.

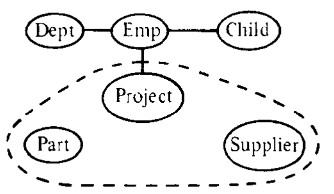

Fig. 4. Initial design of employee database.

There is no room in our model of universal relation definition for subpredicates *PrPa* and *PaS* of the ternary *PrPaS* predicate. In principle, the ternary predicate can check any subpredicate anyway. However, the problem here is that the meaning of the *PrPa* predicate is not: "there is a supplier *s* such that *PrPaS* is true." Thus the *PrPaS* and *PrPa* predicates are not consistent; neither are *PrPaS* and *PaS*.

Our way out of this dilemma is to recognize that Part is an overloaded attribute. It has three meanings, depending on whether it is a part that can be supplied, a part that is needed, or a part that is actually being supplied to a project. Figure 5 shows how the universal relation should be defined in our style.

A second approach is to assert that the physical database will have one relation, *PrPaS*. The pairs (p, a) and (a, s) in *PrPa* and *PaS*, respectively, will be represented by tuples with nulls (p, a, Λ) and (Λ, a, s) in *PrPaS*. □

2.4 The Formal Universal Relation Conjecture

The hypothesis that we would like to advance is that every plausible real world has a universal relation that can be described by

(1) one (full) join dependency and
(2) some number of functional dependencies.

We assume that the join dependency is *full*, meaning that the union of all its objects is the set of all attributes.

Further, we believe that it is possible to select a join dependency that is acyclic, in a sense to be defined formally. We do not wish to take a hard position on whether it is always necessary or desirable that the join dependency be acyclic. To permit cyclic join dependencies may cause certain queries to be ambiguous unnecessarily, while restricting ourselves to acyclic join dependencies may cause us to lose part of the advantage of querying the universal relation. For example, by splitting Part in Example 5 or Customer in Example 2, we force ourselves to remember the jargon for the varieties of Parts and Customers. Remembering the attribute name Part, Needed part, and Suppliable part is essentially the same as remembering that there are three relations in the database, and that we should refer to these attributes as *PrPaS*. Part, *PrPa*. Part, and *PaS*. Part, respectively.

An important consequence of our hypothesis is that there is no role in our database design process for explicitly declared multivalued dependencies. Rather, we see in Section 4 that the multivalued dependencies we would normally expect

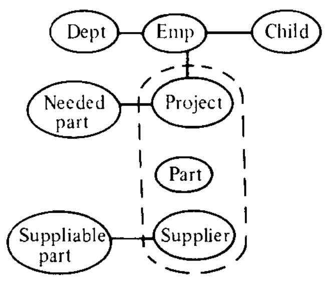

Fig. 5. Revised employee example.

to hold are deducible from the join dependency. In fact, one of the great advantages to this form of database specification is that we need not discover our multivalued dependencies by ourselves; they can be generated automatically from the join dependency. Of course, not all sets of multivalued dependencies can be generated by one join dependency, so our hypothesis may be viewed as an assumption about what sorts of multivalued dependencies one can expect to meet in practice. We shall see that our hypothesis bears a distinct resemblance to assumptions about multivalued dependencies made by [24, 15].

3. IMPLICATIONS OF THE HYPOTHESIS FOR DATABASE DESIGN

One of the first discoveries we make is that our hypothesis makes database design from dependencies straightforward. Ordinarily, to design a database from a given set of dependencies, one would start with a collection of attributes and decompose them into a suitable "normal form" [12, 13, 14]. Deciding whether further decomposition of a set of attributes is necessary requires us to make inferences about dependencies, and in some cases we have to infer embedded multivalued dependencies. As no algorithm for deciding whether such inferences hold is known, our capability to produce a design from the classical forms of information—functional and (possibly embedded) multivalued dependencies—is limited.

However, we can view the database design problem in general as one of selecting a database scheme, or set of sets of attributes. There are several properties the database scheme should have [5, 22, 25], but chief among these is the lossless join property [1], meaning that it should be possible, by taking the natural join, to recover the universal relation from its projections onto the selected relation schemes (sets of attributes in the database scheme). That is, if we select relation schemes $R_1, \ldots, R_k$ as our database scheme, the join dependency $\bowtie(R_1, \ldots, R_k)$ must hold. There are other desirable properties, principally the embedding of functional dependencies in the selected relation schemes [22] that we would like as well, but it appears that this *dependency preservation* property will very likely be satisfied automatically by our design procedures, at least for those functional dependencies that are represented by objects of the join dependency that is given by the design specifications (what we have called data structuring functional dependencies).

We can state very simply the conditions under which a given database scheme is a lossless join decomposition, assuming a definition of the real world by a join dependency and several functional dependencies. It is necessary and sufficient that

$$\{j_1, f_1, \ldots, f_m\} \vDash j_2$$

where j_1 is the join dependency given in the design specifications, $f_1, \ldots, f_m$ are the given functional dependencies, and j_2 is the join dependency whose objects are all the selected relation schemes. The symbol $\vDash$ means "logically implies." Interestingly, this exact problem was proved $\mathcal{NP}$-complete in [19], but at least we can answer the question in a finite amount of time. That is, unlike the classical methods that involve embedded multivalued dependencies, we can always decide whether a given database scheme has the lossless join property and therefore represents the universal relation correctly.

3.1 The Case of No Functional Dependencies

In one overly simple case (where there are no functional dependencies), we can characterize exactly those database schemes that have a lossless join, and the result has some implications for the more general case, in which there are given functional dependencies.

THEOREM 2. *If the legal universal relations are exactly those that satisfy some particular join dependency $\bowtie(R_1, \ldots, R_k)$, then a database scheme $S_1, \ldots, S_m$ has a lossless join if and only if for each R_i there is an S_j such that $R_i \subseteq S_j$.*

PROOF. A proof can be found in [8] or [9]. □

If we wish the join of all the relations in the database to be lossless, then Theorem 2 says it is necessary that the objects of the given join dependency be each contained in some relation scheme. Since we may have reason not to require a lossless join of all the objects, we do not view this containment as mandatory, however.

COROLLARY 2. *If our real world is defined by a join dependency and some functional dependencies, then the "if" direction of Theorem 2 still holds.*

3.2 When We Have Functional Dependencies Along with a Join Dependency

If we are given functional dependencies along with our join dependency, the relation schemes need not be subsets of the objects, nor do the objects need to be subsets of the relation schemes. There may be objects of the join dependency, such as AB and AC, that are properly contained in a relation scheme ABC; yet no violation of even as strong a normal form as projection/join normal form [14] occurs because $A \rightarrow BC$, and so A is a key.

We might also consider the possibility of two relation schemes AB and BC that are contained in an object ABC. As long as $B \rightarrow A$ or $B \rightarrow C$, the join of the relation schemes will still be lossless, because AB and BC join losslessly to produce the object ABC. In fact, when such functional dependencies exist, we are obliged to decompose relation ABC into AB and BC in order to achieve second normal form (unless $B \rightarrow A$ and $B \rightarrow C$ both hold).

4. INFERRING MULTIVALUED DEPENDENCIES FROM ONE JOIN DEPENDENCY

In this section we give a simple rule for discovering the multivalued dependencies that are implied by one join dependency. We also consider the question of when a join dependency is equivalent to the multivalued dependencies it implies, and show that the equivalence occurs only when the join dependency is "acyclic." The interest in this result comes from the argument in [24] that all "realistic" sets of multivalued dependencies are equivalent to one join dependency. Our result lends credence to the naturalness of the conjecture made in Section 3 that the attribute names for the universal relation should be chosen so that the join dependency resulting from definition of the universal relation by predicates yields an acyclic join dependency.

4.1 Definitions

To begin, let us give some terminology for hypergraphs. A *path* from node n to node m is a sequence of $k \geq 1$ edges $E_1, \ldots, E_k$ such that

(1) n is in E_1,
(2) m is in E_k, and
(3) for all $1 \leq i < k$, $E_i \cap E_{i+1}$ is nonempty.

We also say the above sequence of edges is an *edge-path* (or just *path* when no confusion arises) from E_1 to E_k.

Two nodes (or attributes) are *connected* if there is a path from one to the other. Similarly, two edges are *connected* if there is an edge-path from one to the other. A set of nodes or edges is connected if every pair is connected.

Let $(\mathcal{N}, \mathcal{E})$ be a hypergraph. Its *reduction* $(\mathcal{N}, \mathcal{E}')$ is obtained by removing from $\mathcal{E}$ each edge that is a proper subset of another edge. A hypergraph is *reduced* if it equals its reduction, that is, if no edge is a subset of another edge. In what follows, we shall assume that a hypergraph is reduced unless stated otherwise. Note that the join dependency of a hypergraph and the join dependency of its reduction are logically equivalent [8].

Let $\mathcal{M}$ be a set of nodes of the hypergraph $(\mathcal{N}, \mathcal{E})$. The set of *partial edges generated by* $\mathcal{M}$ is obtained by intersecting the edges in $\mathcal{E}$ with $\mathcal{M}$, that is, taking

$$\{(E \cap \mathcal{M}) \mid E \quad \text{is in} \quad \mathcal{E}\} - \{\emptyset\},$$

then taking the reduction of this set of edges. The set of partial edges generated from $(\mathcal{N}, \mathcal{E})$ by some set $\mathcal{M}$ is said to be a *node-generated set of partial edges.*

Let $\mathcal{F}$ be a connected, reduced set of partial edges, and let E and F be in $\mathcal{F}$. Let $Q = E \cap F$. We say that (E, F) is an *articulation pair*, and that Q is an *articulation set* of $\mathcal{F}$, if the result of removing Q from every edge in $\mathcal{F}$ is not a connected set of edges. Evidently, an articulation set in a hypergraph is a generalization of the concept of an articulation point in an ordinary graph.

A *block* of a reduced hypergraph is a connected, node-generated set of partial edges with no articulation set. A block is *trivial* if it consists of a single (partial) edge. A reduced hypergraph is *acyclic* if all its blocks are trivial; otherwise it is *cyclic.* A hypergraph is said to be cyclic or acyclic precisely if its reduction is cyclic or acyclic. We note that our definition of an acyclic hypergraph is different

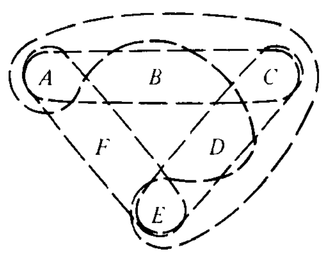

Fig. 6. Example of an acyclic hypergraph.

from that in [11], that is, our definition is strictly less restrictive. We believe that our definition is a better generalization of the usual definition of acyclic (ordinary) graphs. For a discussion of this point, along with a number of definitions equivalent to our definition of "acyclic hypergraph," see [7] and [18].

Example 6. It is straightforward to verify that Figure 6 shows an acyclic hypergraph. Its edges are ABC, CDE, EFA, and ACE.[3] An articulation set for the set of all edges is $ABC \cap ACE = AC$, since the result of removing A and C from each edge is to leave the set of edges B, DE, EF, and E, which is not connected (B is disconnected from the others). Note that the set of edges $\{ABC, CDE, EFA\}$ has no articulation set. However this set of edges is not node generated, so there is no contradiction of our assertion that Figure 6 is acyclic. □

4.2 Inferring Multivalued Dependencies from a Join Dependency

We shall now give a simple technique for computing the set of multivalued dependencies that follow from a join dependency. The method is amenable to computer implementation, and serves as an extremely easy way to deduce multivalued dependencies by "eyeballing" the hypergraph for a join dependency. The method can be viewed as a generalization of a theorem of [15] (although the dependencies inferred there were multivalued dependencies "with nulls," which are slightly different from our multivalued dependencies). It is also a special case of algorithms recently discovered by [19, 20, 26] for general inference of multivalued dependencies.

THEOREM 3. *Let* $j = \bowtie(R_1, \ldots, R_k)$ *be a join dependency. Suppose that* X *and* Y *are disjoint sets of attributes. Then the multivalued dependency* $X \rightarrow\rightarrow Y$ *follows logically from* j *if and only if* Y *is the union of some connected components of the hypergraph of* j *with the set of nodes* X *deleted.*

PROOF. *Only if*: Let us suppose that Z is all the attributes not in X or Y, and assume that there is some R_i that intersects both Y and Z, as suggested in Figure 7. If $X \rightarrow\rightarrow Y$ follows from j, then the tableau test of [17] will say so. That is, if we were to start with the tableau of Figure 8 and apply j to infer additional rows, we would eventually get a row of all a's. However, initially, there is no row with a's in the columns for all the attributes in R_i, and any application of j produces a row that agrees with some previous row on R_i. Thus an easy induction on the

[3] Conventionally, we denote sets of attributes, that is, nodes, by their concatenation. Thus ABC is short for $\{A, B, C\}$.

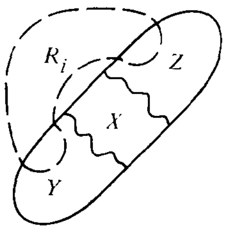

Fig. 7. Hypergraph that does not imply $X \to\to Y$.

$$\begin{array}{ccc} X & Y & Z \\ aa\cdots a & aa\cdots a & bb\cdots b \\ aa\cdots a & bb\cdots b & aa\cdots a \end{array}$$

Fig. 8. Initial tableau to test $X \to\to Y$.

number of added rows proves that no row with a's in all the columns for R_i can ever be added. Thus j does not imply $X \to\to Y$.

If: Suppose that Y is the union of some connected components of the hypergraph of j with X deleted, whereupon Z must be the union of the other connected components. Then we can apply the tableau test to the tableau of Figure 8 successfully; that is, we produce a row of all a's. In fact, we can do so in one step, since we may choose the first row of Figure 8 for those R_m's of j that are subsets of XY, and choose the second row for those R_m's that are subsets of XZ. It is not possible that an R_m is contained neither in XY nor in XZ, or else Y and Z would not be the union of connected components of the hypergraph with X removed. But the proposed mapping of the objects of the join dependency j into the rows of Figure 8 allows us immediately to infer the existence of a row with all a's and prove $j \models X \to\to Y$. □

COROLLARY 3. *The dependency basis* [6, 3] *for a left side X is obtained by deleting the nodes in X from the hypergraph for the given join dependency and finding the connected components that result.*

EXAMPLE 7. The dependency basis of Customer in Figure 1 is

Addr | Bank, Acct, Loan, Bal, Amt,

while the dependency basis of {Bank, Customer} is

Bal, Acct | Loan, Amt | Addr □

4.3 Characterization of Join Dependencies Equivalent to a Set of Multivalued Dependencies

Theorem 3 gives us the set of multivalued dependencies implied by a join dependency j, which we call MVD(j). We can use that result to show that j is logically equivalent to MVD(j) (i.e., the same relations satisfy both j and MVD(j)) exactly when j is acyclic. Before proving this result, we need some definitions and a lemma.

Let $(\mathcal{N}, \mathcal{E})$ be a hypergraph, and let $\mathcal{F}$ be a subset of $\mathcal{E}$. Let $\mathcal{M}$ be the set of nodes that is the union of the members of $\mathcal{F}$. We say $\mathcal{F}$ is *closed* if for each edge I of the hypergraph, there is an edge G in $\mathcal{F}$ such that $I \cap \mathcal{M} \subseteq G$. Note that every closed set of edges is a node-generated set of partial edges, generated by $\mathcal{M}$.

LEMMA 1. *Let $H = (\mathcal{N}, \mathcal{E})$ be a hypergraph, $\mathcal{G}$ a closed subset of $\mathcal{E}$, and $\mathcal{M}$ the set of nodes of $\mathcal{G}$. Let $\mathcal{F}$ be a closed subset of the hypergraph $(\mathcal{M}, \mathcal{G})$. Then $\mathcal{F}$ is also a closed subset of H.*

PROOF. Let I be an edge in $\mathcal{E}$, and let $\mathcal{P}$ be the union of the edges in $\mathcal{F}$. We must show there is some edge G in $\mathcal{F}$ such that $I \cap \mathcal{P} \subseteq G$. Since $\mathcal{G}$ is closed with respect to $\mathcal{E}$, there is some edge J in $\mathcal{G}$ such that $I \cap \mathcal{M} \subseteq J$. Then, since $\mathcal{F}$ is closed with respect to $\mathcal{G}$, there is some edge K in $\mathcal{F}$ such that $J \cap \mathcal{P} \subseteq K$. Thus, $I \cap \mathcal{P} \subseteq K$. $\square$

Let $\mathcal{F}$ be a closed connected set of two or more edges in a reduced acyclic hypergraph. Since $\mathcal{F}$ is a node-generated set of partial edges there is an articulation set Q for $\mathcal{F}$. The edges in $\mathcal{E}$ can thus be partitioned into nonempty sets $\mathcal{H}_1, \ldots, \mathcal{H}_k$, $k \geq 2$, such that

(1) each edge of $\mathcal{F}$ is in exactly one $\mathcal{H}_i$,
(2) $\mathcal{H}_i - Q$[4] is connected for each i, and
(3) if E_1 is an edge in $\mathcal{H}_i$ and E_2 is an edge in $\mathcal{H}_j$, $i \neq j$, then $E_1 \cap E_2 \subseteq Q$.

The fact that such a partition exists is exactly what we mean by saying that Q is an articulation set of $\mathcal{F}$. Each $\mathcal{H}_i$ is obtained by taking one of the connected components of $\mathcal{F} - Q$ and adding back to each edge its intersection with Q. We call each $\mathcal{H}_i$ a *component of $\mathcal{F}$ after articulation by Q*. We are now ready to prove the main theorem of the paper.

THEOREM 4. *The join dependency j is logically equivalent to a set of multivalued dependencies if and only if j's hypergraph is acyclic.*

PROOF. *If*: Evidently, j implies MVD(j), so we need only to prove that j follows from MVD(j). Without loss of generality, we assume that j's hypergraph is reduced. We use the tableau test of [17] and prove that if we chase the tableau of j with MVD(j), then we obtain a row of distinguished symbols ("a's") in all the attributes.

We can assume that the hypergraph is connected. In proof, note that if the theorem holds for connected components, then, since by Theorem 3, $\varnothing \twoheadrightarrow X$ for every connected component X, we can form a row of all a's from the rows with a's in the connected components.

We shall now define, inductively on i, certain closed subsets of edges to be *i-level components* and *i-level augmented components.* We start the induction by letting the set of all edges be the unique 0-level component and the unique 0-level augmented component.

Suppose that we have defined $(i - 1)$-level components and augmented components, $i \geq 1$. Assume inductively that each $(i - 1)$-level augmented component is closed. We now define the i-level components and augmented components. If $\mathcal{P}$ is an $(i - 1)$-level augmented component with exactly one edge, then $\mathcal{P}$ is also an i-level component and augmented component.

If $\mathcal{P}$ is an $(i - 1)$-level augmented component with more than one edge, then since j is acyclic, there is an articulation pair (E, F) of $\mathcal{P}$, since by the inductive

[4] We use $\mathcal{E} - X$ to mean the set of partial edges generated (as defined above) from set of edges $\mathcal{E}$ by the set of nodes consisting of all the nodes in members of $\mathcal{E}$ except those in Q.

hypothesis, $\mathscr{P}$ is closed, and therefore, node generated. Let $\mathscr{H}_1, \ldots, \mathscr{H}_k$ be the components of $\mathscr{P}$ after articulation by $E \cap F$. For each m, $1 \le m \le k$, define $\mathscr{H}'_m$ to be $\mathscr{H}_m$ if $\mathscr{H}_m$ contains one or both of the edges E or F,[5] and $\mathscr{H}'_m = \mathscr{H}_m \cup \{E\}$ otherwise. Each $\mathscr{H}_m$ is an i-level component, and each $\mathscr{H}'_m$ is an i-level augmented component.

By construction, each $\mathscr{H}'_m$ contains either E or F. We now show that each $\mathscr{H}'_m$ is closed. It is sufficient, by Lemma 1, to show that $\mathscr{H}'_m$ is a closed subset of the closed subset $\mathscr{P}$. Let $\mathscr{M}_m$ be the set of nodes appearing in $\mathscr{H}'_m$, and let I be an edge of $\mathscr{P}$. We must show that $I \cap \mathscr{M}_m \subseteq G$ for some G in $\mathscr{H}'_m$. If I is in $\mathscr{H}'_m$, then let $G = I$. Suppose I is not in $\mathscr{H}'_m$. We know that the unaugmented component $\mathscr{H}_m$ is a connected component of $\mathscr{P}$ after articulation by $E \cap F$. Thus, as I is in $\mathscr{P}$ but not in $\mathscr{H}'_m$, we know that its intersection with edges of $\mathscr{H}'_m$, if any, is limited to the articulation set, that is, $I \cap \mathscr{M}_m \subseteq E \cap F$. Thus we can take G to be whichever of E and F is in $\mathscr{H}'_m$.

We call each $\mathscr{H}'_m$ a *child* of $\mathscr{P}$, and we call $\mathscr{P}$ the *father* of $\mathscr{H}'_m$. The transitive closure of the father relation is the *ancestor* relation. It is straightforward that each child has strictly fewer edges than its father. It follows by induction on i that each i-level augmented component has at most $n - i$ edges, if n is the number of edges in the original hypergraph. In particular, the $(n - 1)$-level augmented components are precisely the singleton sets $\{G\}$ for each edge G of the original hypergraph.

Assume as before that $\mathscr{P}$ is an $(i - 1)$-level augmented component with (E, F) an articulation pair, and that $\mathscr{H}_1, \ldots, \mathscr{H}_k$, $k \ge 2$, are the components of $\mathscr{P}$ after articulation by $E \cap F$. We shall now show that

(*) (E, F) is an articulation pair for the whole hypergraph H, and each of $\mathscr{H}_1, \ldots, \mathscr{H}_k$ is contained in a distinct component of H after articulation by $E \cap F$.

If $i = 1$, that is, if $\mathscr{P}$ is the 0-level augmented component, which contains every edge, then (*) is immediate. Assume that $\mathscr{P}$ has a father $\mathscr{R}$. We show that (E, F) is an articulation pair for $\mathscr{R}$, and that each of $\mathscr{H}_1, \cdots, \mathscr{H}_k$ is contained in a distinct component after articulation of $\mathscr{R}$ by $E \cap F$. Then (*) follows easily by induction on the level of $\mathscr{P}$.

Assume by way of contradiction that $\mathscr{H}_1$ and $\mathscr{H}_2$ are in the same component after articulation of $\mathscr{R}$ by $Q = E \cap F$. Pick edge G_1 in H_1 and G_2 in H_2. We then know that there is a sequence of edges $X_1, \cdots, X_t$ of $\mathscr{R}$, where

(1) $G_1 = X_1$,
(2) $G_2 = X_t$, and
(3) $X_m \cap X_{m+1}$ is not wholly contained within Q, for $1 \le m < t$.

We know that some X_m is not in $\mathscr{P}$, since H_1 and H_2 are distinct components of $\mathscr{P}$ after articulation by Q. Let u be the minimum value of m, and v the maximum value of m such that X_m is not in $\mathscr{P}$. Then $1 < u \le v < t$.

[5] Note that just because the removal of $E \cap F$ disconnects the hypergraph does not mean that E is disconnected from F. It is possible that $E \cap F$'s removal disconnects certain other edges whose sole connection to the rest of the graph was through nodes in $E \cap F$. It is true in this case, however, that another articulation pair could have been chosen.

Let (C, D) be the articulation pair of $\mathscr{R}$ that led to the creation of $\mathscr{P}$. Then $\mathscr{P}$ contains at least one of C and D; by renaming if necessary, assume it is C. Consider the sequence of edges

$$X_1, X_2, \ldots, X_{u-1}, C, X_{v+1}, \ldots, X_t$$

in which we have "spliced" C in place of $X_u, \ldots, X_v$. Every edge in this sequence is in $\mathscr{P}$. To derive a contradiction, we need only show that this path is one in which each consecutive pair has a point in common that is not in $Q = E \cap F$, for we know that $X_1 = G_1$ and $X_t = G_2$, but G_1 and G_2 are supposed to be in distinct components of $\mathscr{P}$ after articulation by Q.

By (3) above, we already know that $X_m \cap X_{m+1} - Q$ is not empty for $1 \le m < u - 1$ and for $v < m < t$. Thus we need only show that $X_{u-1} \cap C - Q$ and $X_{v+1} \cap C - Q$ are nonempty. We know that there is some node d in $X_{u-1} \cap X_u - Q$, by condition (3) above. Now X_{u-1} and X_u are both in $\mathscr{R}$, but they are not in the same child of $\mathscr{R}$, since X_{u-1} is in $\mathscr{P}$, but X_u is not. So, since d is in $X_{u-1} \cap X_u$, we know that d is in $C \cap D$. We already know that d is in $X_{u-1} \cap X_u - Q$, so d is in $X_{u-1} \cap C - Q$. Hence $X_{u-1} \cap C - Q$ is nonempty. By an identical argument, $X_{v+1} \cap C - Q$ is nonempty, as was to be shown. We have now shown that if $\mathscr{P}$ is an $(i - 1)$-level augmented component with (E, F) as articulation pair and with $\mathscr{H}_1, \ldots, \mathscr{H}_k$ as the components of $\mathscr{P}$ after articulation by $E \cap F$, then (E, F) is an articulation pair for the whole hypergraph H, and that each of $\mathscr{H}_1, \ldots, \mathscr{H}_k$ is contained in a distinct component of H after articulation of H by $E \cap F$.

We now prove by reverse induction on i that if we chase the tableau of the join dependency j with MVD(j), then we obtain a row with distinguished symbols in all of the attributes of $\mathscr{P}$, for each i-level augmented component $\mathscr{P}$. The statement is true when $i = n - 1$, where n is the total number of nodes, since as we saw, every $(n - 1)$-level augmented component is a single edge. We assume the result for $i \ge 1$, and show it for $i - 1$.

Let $\mathscr{P}$ be an $(i - 1)$-level augmented component, and let $\mathscr{H}_1, \ldots, \mathscr{H}_k$ be its components after articulation as above, by $E \cap F$. Let $\mathscr{H}'_1, \ldots, \mathscr{H}'_k$ be the corresponding augmented components. By the inductive hypothesis, for each m, $1 \le m \le k$, there is a row r_m in the chased tableau with distinguished symbols in all the attributes of $\mathscr{H}'_m$ and hence of $\mathscr{H}_m$. We just proved that $E \cap F$ is an articulation set of the whole hypergraph, and that each $\mathscr{H}_m$ is in a distinct component after articulation of the hypergraph by $E \cap F$. It follows easily that chasing the rows $r_1, \ldots, r_k$ using multivalued dependencies with left-hand side $E \cap F$, we obtain a row with distinguished symbols in all the attributes of $\mathscr{P}$.

This completes the induction step. In particular, it follows that there exists in the chased tableau a row with distinguished symbols in every attribute of the 0-level augmented component, that is, in every attribute whatsoever. Hence MVD(j) implies j, as was to be shown.

Only if: Suppose that the hypergraph of j is cyclic. Then there is a set $\mathscr{M}$ of nodes such that the set $R_1, \ldots, R_m$ of partial edges generated by $\mathscr{M}$ has no articulation set, and $m \ge 2$. We show by induction on the number of rows added when chasing the tableau of j according to the set of multivalued dependencies MVD(j), that the projection of any row onto the columns for $\mathscr{M}$ is the projection of a row that existed at the outset of the chase procedure.

The basis, zero rows, is immediate. For the induction, suppose that at some time we can use the multivalued dependency $X \rightarrow\rightarrow Y$ in MVD(j) to produce a row whose projection onto set of attributes $\mathcal{M}$ differs from the projections onto $\mathcal{M}$ that already exist. Let Z be all the attributes not in X or Y. Let r_1 and r_2 be the pair of rows to which $X \rightarrow\rightarrow Y$ is applied to generate, for the first time, a new projection onto $\mathcal{M}$. So by the inductive assumption, $\pi_{\mathcal{M}}(r_1) = \pi_{\mathcal{M}}(s_1)$ and $\pi_{\mathcal{M}}(r_2) = \pi_{\mathcal{M}}(s_2)$ for some rows s_1 and s_2 of the original tableau. Now $\pi_{\mathcal{M}}(r_1) \neq \pi_{\mathcal{M}}(r_2)$, or else a new projection onto $\mathcal{M}$ could not be generated from these rows by a multivalued dependency. Thus $\pi_{\mathcal{M}}(s_1) \neq \pi_{\mathcal{M}}(s_2)$, and so $s_1 \neq s_2$.

Assume that s_1 corresponds to the set of attributes S_1 (that is, the distinguished symbols of s_1 appear in exactly the columns for the attributes in S_1) and, similarly, s_2 corresponds to S_2. Recall that all nondistinguished symbols are unique. So, s_1 and s_2 agree precisely on $S_1 \cap S_2$. We know that $S_1 \cap \mathcal{M} = R_p$ and $S_2 \cap \mathcal{M} = R_q$ for some p and q, because $R_1, \ldots, R_m$ is the node-generated set of partial edges, generated by $\mathcal{M}$. So $\pi_{\mathcal{M}}(s_1)$ and $\pi_{\mathcal{M}}(s_2)$ agree precisely on $S_1 \cap S_2 \cap \mathcal{M} = R_p \cap R_q$. Thus $\pi_{\mathcal{M}}(r_1)$ and $\pi_{\mathcal{M}}(r_2)$ agree precisely on $R_p \cap R_q$. To apply the multivalued dependency $X \rightarrow\rightarrow Y$ to r_1 and r_2 and get something new, rows r_1 and r_2 must at least agree on X. So $\pi_{\mathcal{M}}(r_1)$ and $\pi_{\mathcal{M}}(r_2)$ must agree on $\mathcal{M} \cap X$. Since we showed that $\pi_{\mathcal{M}}(r_1)$ and $\pi_{\mathcal{M}}(r_2)$ agree precisely on $R_p \cap R_q$, it follows that $\mathcal{M} \cap X \subseteq R_p \cap R_q$. Note that $R_p \neq R_q$, because if $R_p = R_q$ then $S_1 \cap \mathcal{M} = S_2 \cap \mathcal{M}$, whereupon $\pi_{\mathcal{M}}(s_1) = \pi_{\mathcal{M}}(s_2)$, a contradiction.

Since $R_1, \ldots, R_m$ forms a block, we know that deletion of $R_p \cap R_q$ does not disconnect these partial edges. But $\mathcal{M} \cap X \subseteq R_p \cap R_q$. Thus the deletion of X does not disconnect these partial edges. We may conclude from Theorem 3 that either $\mathcal{M} \subseteq XY$ or $\mathcal{M} \subseteq XZ$. As a result, applying the multivalued dependency $X \rightarrow\rightarrow Y$ produces only rows whose projection onto $\mathcal{M}$ already existed in another row, and it is not possible that this application of $X \rightarrow\rightarrow Y$ is the first to produce a row that has a's in a set of columns of $\mathcal{M}$ that is not a subset of the columns for any R_i.

We conclude that the chase process never introduces any new projections of rows onto $\mathcal{M}$. Thus, when we restrict our attention to $\mathcal{M}$, each row has a's only in a set of columns that is contained in some R_i, $1 \leq i \leq m$. Unless $\mathcal{M}$ is a subset of an edge of the hypergraph for j, in which case the block is trivial, contrary to hypothesis, none of these rows have a's in all the rows of $\mathcal{M}$. Thus certainly we do not, by chasing, produce a row with a's everywhere, and we conclude that j cannot be inferred from MVD(j). Hence, j is not equivalent to MVD(j), and so clearly, j is not equivalent to any set of multivalued dependencies. □

Note that if j is equivalent to MVD(j), then j is equivalent to the set of those multivalued dependencies in MVD(j) whose left-hand side is the intersection of two objects in j. This statement follows from the proof of the "if" portion of Theorem 4, in which we made use of only those members of MVD(j). As a consequence, for any acyclic join dependency j, we can find a subset of MVD(j) that is equivalent to MVD(j) and whose size is a polynomial in the size of j. We note that Beeri, Fagin, Maier, and Yannakakis [7] strengthen this result by showing that if j is acyclic then we can find a subset of MVD(j) that is equivalent to MVD(j) and is *linear* in the size of j.

5. CONCLUSIONS

We have presented a way to define universal relations and their semantic constraints (dependencies) that, while restricted, appears to offer enough power to describe real-world situations. Moreover, the method is sufficiently restricted that we can guarantee a database design into projection/join normal form [14] without encountering any possibly undecidable problems concerning whether a particular relation can or must be decomposed.

We also explored the class of universal relations defined by acyclic join dependencies. These were characterized as those join dependencies that are equivalent to the multivalued dependencies they imply. This result in turn shows that the class of universal relations defined by acyclic join dependencies includes those that are considered reasonable by [24, 15]. A number of other remarkable propeties of acyclic join dependencies have been shown in [7].

ACKNOWLEDGMENTS

The authors are grateful to Catriel Beeri, Maria Klawe, David Maier, and Moshe Vardi for useful suggestions.

REFERENCES

1. AHO, A.V., BEERI, C., AND ULLMAN, J.D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 297–314.
2. AHO, A.V., AND ULLMAN, J.D. *Principles of Compiler Design.* Addison-Wesley, Reading Mass.
3. BEERI, C. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Database Syst. 5*, 3 (Sept. 1980), 241–259.
4. BEERI, C., AND BERNSTEIN, P.A. An algorithmic approach to normalization of relational database schemas, CSRG-73, Univ. Toronto, 1975.
5. BEERI, C., BERNSTEIN, P.A., AND GOODMAN, N. A sophisticate's introduction to database normalization theory. In *Proc. 4th Int. Conf. Very Large Databases*, (West Berlin, Sept. 13–15), ACM, New York, 1978, pp. 113–124.
6. BEERI, C., FAGIN, R., AND HOWARD, J.H. A complete axiomatization for functional and multivalued dependencies in database relations. *ACM SIGMOD Int. Symp. Management of Data*, ACM, New York, 1977, pp. 47–61.
7. BEERI, C., FAGIN, R., MAIER, D., AND YANNAKAKIS, M. On the desirability of acyclic database schemes. To appear in *J. ACM.*
8. BEERI, C., MENDELZON, A.O., SAGIV, Y., AND ULLMAN, J.D. Equivalence of relational database schemes. *SIAM J. Comput. 10*, 2 (June 1981), 352–370.
9. BEERI, C. AND VARDI, M.Y. On the properties of joint dependencies. In *Proc. Workshop Formal Bases for Databases* (Toulouse, Dec. 1979).
10. BERNSTEIN, P.A. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst. 1*, 4 (Dec. 1976), 277–298.
11. BERGE, C. *Graphs and Hypergraphs.* Elsevier North-Holland, New York, 1973.
12. CODD, E.F. A relational model for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377–387.
13. FAGIN, R. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2*, 3 (Sept. 1977), 262–278.
14. FAGIN, R. Normal forms and relational database operators. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, (Boston, Mass., May 30–June 1), ACM, New York, 1979, pp. 153–160.
15. LIEN, Y.E. On the equivalence of database models. Database Research Report No. 3 (July 1980), Bell Laboratories, Holmdel, N.J.
16. LIEN, Y.E. On the semantics of the entity-relationship model. In *Entity-Relationship Approach*

to Systems Analysis and Design, P. P. Chen, Ed, Elsevier North-Holland, New York, 1980, pp. 155–167.

17. MAIER, D., MENDELZON, A.O., AND SAGIV, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455–469.
18. MAIER, D., AND ULLMAN, J.D. Connections in acyclic hypergraphs. In *Proc. ACM Symp. Principles of Database Systems* (Los Angeles, March 29-31), ACM, New York, 1982, pp. 34-39.
19. MAIER, D., SAGIV, Y., AND YANNAKAKIS, M. On the complexity of testing implications of functional and join dependencies. *J. ACM 28*, 4 (Oct. 1982), 680–695.
20. MENDELZON, A.O., AND MAIER, D. Generalized mutual dependencies and the decomposition of database relations. *In Proc. 5th Int. Conf. Very Large Databases*, (Rio de Janeiro, Oct. 3-5), ACM, New York, 1979, pp. 75-82.
21. RISSANEN, J. Theory of joins for relational databases—A tutorial survey. In *Proc. 7th Symp. Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 64, Springer-Verlag, pp. 537–551.
22. RISSANEN, J. Independent components of relations. *ACM Trans. Database Syst. 2*, 4 (Dec. 1977), 317-325.
23. SCIORE, E. Null values, updates, and normalization in relational databases. Doctoral dissertation, Princeton Univ., Princeton, N. J., 1980.
24. SCIORE, E. Real-world MVD's. In *ACM SIGMOD Int. Conf. Management of Data* (Ann Arbor, Mich., April 29-May 1), ACM, New York, 1981, pp. 121-132.
25. ULLMAN, J.D. Principles of Database Systems. Computer Science Press, Potomac, Md., 1980.
26. VARDI, M.Y. Inferring multivalued dependencies from functional and join dependencies. Dep. of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1980.

Received November 1980; revised July 1981; accepted August 1981