HORN TABLES - AN EFFICIENT TOOL FOR HANDLING INCOMPLETE INFORMATION IN DATABASES

Gösta Grahne

University of Helsinki, Department of Computer Science
Teollisuuskatu 23 SF-00510 Helsinki FINLAND

1. INTRODUCTION

The basic semantic assumption is that an incomplete information database is a *set of possible worlds* (i.e. a set of complete databases). The main issue is then the problem of representing the set of possible worlds in a fashion that is suitable for storing and processing the database. There now exist a variety of representations, including the so called logical databases [15, 20] and algebraic generalizations of ordinary relations [2, 4, 10, 12]. In particular, [2] presents a hierarchy of representations obtained by allowing variables and conditions on variables as entries in relations.

In [2] the representation hierarchy is characterized w.r.t. the data complexity of querying incomplete information databases. We shall however see that taking *dependencies* into account complicates the picture somewhat. Thus we complement the representation hierarchy with a new construct called *Horn tables*. This class of tables is algebraically closed w.r.t. total dependencies, meaning that the information contained in the dependencies can be incorporated into a Horn table in such a way that the resulting table is still a Horn table. If the set of dependencies consists of one join dependency and a set of equality generating dependencies, the resulting table is no larger than a polynomial of the size of the original table (in some special cases any set of total dependencies is allowed). We shall also see that Horn tables allow *positive existential* queries to be evaluated in polynomial time (data complexity). If one is interested in *certain answers* only, then any *datalog* query can be used without increasing the complexity.

The rest of this paper is organized as follows. In Section 2 we give the definition of Horn tables and show how they fit into the representation hierarchy. Section 3 considers the problem of query evaluation. We define the certain and possible answers and give the central complexity results for Horn tables. We also recall some results from [2] so that we can give a completed picture of the complexity of querying incomplete information databases. In Section 4 we define the *completion* of an incomplete information database w.r.t. a set of dependencies, and we characterize the complexity of computing the completion for all representations in the hierarchy.

Throughout this paper we make the *closed world assumption* [14], and the computational characterizations are in terms of *data complexity* [18], i.e. the complexity measure is a function of the database size, and not of the query size, or the size of the dependency set. Data complexity is considered as a reasonable measure since the the number of tuples in the database typically is the dominating factor in an application.

2. HORN TABLES AND THE REPRESENTATION HIERARCHY

We assume familiarity with the notions of relational databases, queries and dependencies (see e.g. [17]), in order to use a minimum of notation and space.

Let the *domain* be a countably infinite set of constants {0, 1, 2,..., c, ...}. A *relation* r of arity (a) is a finite subset of $(\text{domain})^a$, where a is an integer ≥0. A member of a relation is

called a *fact*. A *complete information database* I of arity $(a_1, ..., a_n)$ is a n-vector of relations $(r_1, ..., r_n)$, such that each r_i is of arity a_i.

Let *variables* be a countably infinite set {x, y, z, u, v, ...} that is disjoint from the domain. The variables are used to express null values, i.e. *values present but unknown*. A *table* T of arity (a) is obtained by replacing some occurrences of constants in a relation of arity (a) by *distinct* variables, so that each variable occurs at most once in the table. A *tuple* t of a table T is a tuple of variables and constants appearing as a row of T.

An *equality* is an expression of the form $x=\delta$, where x is a variable and δ is a variable or a constant. A *literal* is an equality $x=\delta$ or the negation *not*$(x=\delta)$ of an equality. A *condition* is a finite boolean combination of equalities. The boolean *true* and *false* are encoded as x=x and *not*(x=x), respectively. Conditions are associated with a table T in two ways:

(i) a finite set of conditions $\Phi(T)$, called the *global condition*, is associated with the entire table,

(ii) a *local condition* $\phi(t)$ is associated with each tuple t of T.

A *universal table* (u-table for short) is a table T together with the associated conditions. An example of a u-table T appears in Figure 1. The two tuples are $t_1 = (1, x, 2)$ and $t_2 = (2, y, z)$. The local conditions are $\phi(t_1)$ = *not*(x=1 *or* z=2), and $\phi(t_2)$ = (z=5). The global condition $\Phi(T)$ is {(z=2 *or* y=x), *not*(z=9)}.

	{(z=2 *or* y=z), *not*(z=9)}
1 x 2	*not*(x=1 *or* z=2)
2 y z	z=5

FIGURE 1. A u-table

The set of instances represented by a u-table is obtained through valuations. A *valuation* h is a mapping from variables and constants to constants, such that h(c)=c for each constant. A valuation h naturally extends to tuples t and tables T, producing facts h(t) and relations h(T). A valuation h *satisfies* a condition $\phi(t)$ or $\Phi(T)$, if its assignement of constants to variables makes formulas $\phi(t)$ resp. $\Phi(T)$ true. The set of instances $\mathfrak{I}$ represented by a U-table T is defined as {I : there is a valuation h satisfying $\Phi(T)$, such that instance I consists exactly of those facts h(t) for which h satisfies $\phi(t)$}. (Cf. Figure 2.)

1 3 2
2 3 5

1 2 2

FIGURE 2. Two instances represented by the u-table.

The above definitions easily generalize to n-vectors of u-tables, as opposed to 1-vectors, and I's of arity $(a_1, ...,a_n)$ as opposed to arity (a). For this generalization the sets of variables appearing in each $T_1, ..,T_n$ are pairwise disjoint; relationships between these variables are established through the conditions.

The representation hierarchy is obtained by considering tables with syntactically restricted conditions. Let $l_1, l_2, ..., l_m$ be literals. A condition is said to be *Horn*, if it is of the form $(l_1$ *or* l_2 *or* ... *or* $l_m)$, $m \geq 1$, such that at most one of the literals is not a negated equality. A table T is said to be a *Horn-table* (h-table) if the global condition is a set of Horn conditions, and the local conditions are conjunctions of equalities.

The representation hierarchy in [2] includes the following classes of tables: *Conditional tables* (c-tables, all conditions are conjunctions of literals), *global tables* (g-tables, the global condition is a set of conjunctions of literals and the local conditions are *true*), *equality tables* (e-tables, the global condition is a set of conjunctions of equalities, the local conditions are *true*), *inequality tables* (i-tables, the global condition is a set of conjunctions of negated equalities and the local conditions are *true*), and *plain tables* (p-tables), where all conditions are *true*.

The (augmented) representation hierarchy is shown in Figure 3. We note that h-tables are on the same level as c-tables, but we shall see that the h-tables preserve the computational bounds for g-tables.

The u-tables were introduced in [10], generalizing a construct in [12]. The g-tables are equivalent (modulo isomorphism) to the logical databases of [20]. Constructs similar to p-tables appears in [4, 12], and the e-tables appear in [12].

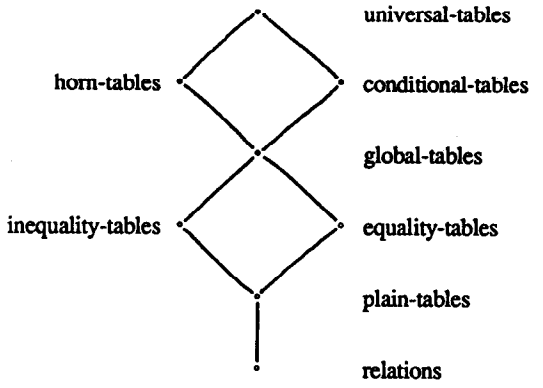

FIGURE 3. The representation hierarchy.

3. THE COMPLEXITY OF QUERY EVALUATION

A *query* q of *arity* $(a_1, ..., a_n) \rightarrow (b_1, ..., b_m)$ is a mapping from instances to instances of the appropriate arities. Let I be an instance of arity $(a_1, ..., a_n)$. Then the *answer* to a query q is q(I), i.e. the mapping applied to the instance. The answer q(I) is of arity $(b_1, ..., b_m)$.

In an incomplete information database we have two types of answers. Let $\mathfrak{T}$ be a set of instances of appropriate arities. Then the *possible answer* to a query q is

$$q(\mathfrak{T}) = \{q(I) : I \text{ is in } \mathfrak{T}\}.$$

The *certain answer* is

$$\{J : J \text{ is in } q(I), \text{ for all } I \text{ in } \mathfrak{T}\}.$$

We shall consider three classes of queries (cf. Figure 4):

(1) *Positive existential queries*. They can be expressed using relational expressions with operators *project, cartesian product, union, positive restrict* and *positive select.*

(2) *First order queries*, i.e. queries that can be expressed using *genral restrict, general select* and *difference* in addition to the operators in the expressions for (1) (the complete relational algebra or calculus).

(3) *Datalog queries* (recursive queries). These queries can be expressed as least fixpoints of positive existential queries.

FIGURE 4. Part of the query hierarchy [3].

In complete databases the answer to all queries in the classes (1) to (3) can be evaluated time polynomial in the size of the database [3] (data complexity, i.e. in the number of tuples in the database. The relation width and query size are fixed.) The data complexity of query evaluation for incomplete information databases was studied in [2] (see also references therein). The data complexity is a function of the *size* of a table. By the size we mean the number of tuples in the table plus the length of the conditions. The arities of the tables are assumed to be fixed. The size generalizes in the natural way to vectors of tables.

For the possible answer the following decision problem was used in [2]:

Let q be a query, k a positive integer and x one of (u, h, c, g, e, i, p). Then the *possibility problem* is, given a set P of at most k facts and an x-table representing a set $\mathfrak{T}$ of instances, to decide if there is an instance J in $q(\mathfrak{T})$, such that all facts of P are facts of J. The problem is denoted POSS(k, q, x).

The following result is known:

THEOREM 1 [2].

(i) POSS(k, q, c) is in NP.
(ii) If q is a positive existential query, then POSS(k, q, c) is in PTIME.
(iii) There is a first order query q, such that POSS(1, q, p) is NP-complete.
(iv) There is a datalog query q, such that POSS(1, q, p) is NP-complete.

To this we can now add

THEOREM 2.

(i) POSS(k, q, u) is in NP.
(ii) If q is a positive existential query, then POSS(k, q, h) is in PTIME.
(iii) Let q be the identity query. Then POSS(1, q, u) is NP-complete.

PROOF (*sketch*). (i) is a variation of Theorem 1 (i). The third claim is prooved by a reduction from the 3CNF satisfiablitiy problem [8]. The formula is encoded in the global condition. For (ii) we note that the query q can be incorporated in the table using the technique of [12]. The resulting table is still a Horn table, and its size is polynomial in the size of the original table. Then we look for a bounded pattern.∎

The theorem means that we can efficiently evaluate the possible answer to any positive existential query in Horn tables. The class of Horn tables is closed under these queries, meaning that the result is still a Horn table (and of size a polynomial of the original table). Thus the structure of the table is regular enough, so that a user who asks "is this small set of facts possible in the result" can have his answer in polynomial time. The results of Theorem 1 and 2 are summarized in Figure 5.

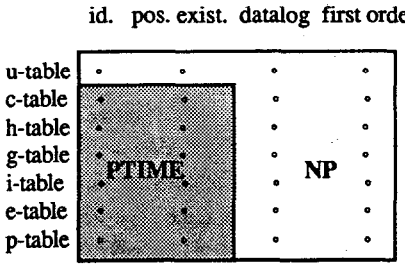

FIGURE 5. The complexity of the possibility problem.

For characterizing the complexity of evaluating the certain answer the following decision problem was used in [2]:

Let q be a query, k a positive integer and x one of (u, h, c, g, e, i, p). Then the *certainty problem* is, given a set P of at most k facts and a x-table representing a set $\mathcal{T}$ of instances, to decide if all facts of P are facts of q(I), for all I in $\mathcal{T}$.

Since the certainty problem is polynomially equivalent to the same problem where k is no longer a parameter [2], we shall denote the problem CERT(q, x). Now we can cite the following result:

THEOREM 3 [2].

(i) CERT(q, c) is in coNP.
(ii) For any datalog query q, CERT(q, g) is in PTIME.
(iii) Let q be the identity. Then CERT(q, c) is coNP-complete.
(iv) There is a first order query q, such that CERT(q, p) is coNP-complete.∎

The following theorem completes the picture.

THEOREM 4.

(i) CERT(q, u) is in coNP.
(ii) For any datalog query q, CERT(q, h) is in PTIME.

PROOF (*sketch*). For (ii) we replace x by δ in the input table T, for all (x, δ), such that the global condition Φ(T) logically implies x=δ. This step can be done in time polynomial in the size of T. Then we forget about the conditions and evalute the query as in complete databases, i.e. we treat the variables as constants, pairwise different, and different from all "real" constants. The evaluation in complete databases can be done in time polynomial in the size of the database [3].∎

The results of Theorem 3 and 4 are summarized in Figure 6.

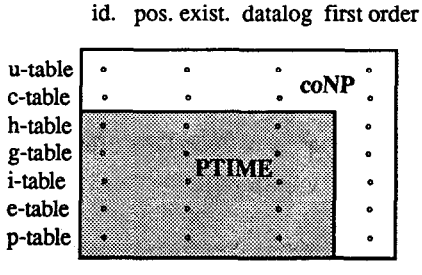

FIGURE 6. The complexity of the certainty problem.

Thus Horn tables strengthens the result for global tables (logical databases). In comparison with the possible answer, Horn tables allow efficient evaluation of the certain answer to positive existential and datalog queries, wheras only positive existential queries can be allowed in efficiently evaluating the possible answer.

The real importance of the class of Horn tables will become clear in the next section, where we consider the problem of dependency enforcement. It turns out that Horn tables are algebraically closed w.r. t. total dependencies, and that the Horn tables is the only restricted class with this property.

4. DEPENDENCIES : REPRESENTATION AND COMPLEXITY

The data dependencies form a language for specifying the semantics of an application [7], by restricting the set of allowed instances. Another way to look at the dependencies is as a set of *deductive rules*. This view holds especially for the so called tuple generating dependencies. In incomplete information databases the equality generating dependencies also play the role of deductive rules, since they will enforce restrictions on the null values.

Let P be a predicate of arity n. Then $P(x_1, ..., x_n)$ is an *atomic formula*. A *total dependency* of *arity* n is a first order sentence (*for all* $(x_1, ..., x_k)(A_1$ *and* ... *and* $A_p)$ *implies* B) [5]. Here each A_i is an atomic formula, and B is an atomic formula or an equality $x_i=x_j$, where x_i and x_j appears in the A_i:s. There are no free variables. If B is an atomic formula, then the dependency is a *tuple generating* one (abbr. TGD), and if B is an equaility, then the sentence represents an *equality generating dependency* (abbr. EGD). The well known *join dependency* (abbr JD) is a special case of a TGD, and the also well known *functional dependency* (abbr. FD) is a special case of an EGD. From hereafter we will by a dependency mean a total one. Dependencies will be denoted σ, and finite sets of dependencies of the same arities are denoted Σ.

An instance I of arity (n) *satisfies* a dependency σ of arity (n) if I is a model (in the sence of mathematical logic) of the sentence. If I satisfies every dependency σ in a set Σ we say that I satisfies Σ. In the sequel, when we speak of instances and sets of dependencies, we implicitly assume that they are of the same arity. We shall also consider the following restricted classes of dependencies (cf. Figure 7).

(1) *Sets of FD´s*. This is the most basic and practical class.

(2) *Sets consisting of EGD´s and one JD*. In [6] it is argued that every real world application can be sufficiently constrained by one join dependency and a set of functional dependencies. Such a class is properly contained in this class.

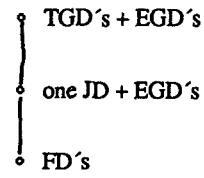

FIGURE 7. Classes of dependencies.

Let I be an instance and Σ a set of dependencies. Then the *completion* of I w.r.t. Σ, denoted $\Sigma(I)$, is the smallest instance J, such that J contains I and J satisfies Σ (cf. [13]). Note that the completion does not necessary always exist, but if it does, it is unique and contains all the information implied by the dependencies (deductive rules). The requirement for minimality is the interpretation of the closed world assumption.

Then let $\mathfrak{I}$ be a set of instance and Σ a set of dependencies. Then the *completion* of $\mathfrak{I}$ w.r.t. Σ is

$$\Sigma(\mathfrak{I}) = \{\Sigma(I) : I \text{ is in } \mathfrak{I}\}.$$

Since $\mathfrak{I}$ is represented by a table in some class we are faced with the problem of representing $\Sigma(\mathfrak{I})$ in some way. It turns out that apart from the u-tables, the class of Horn tables is the only class that is able to represent $\Sigma(\mathfrak{I})$ for all Σ. Formally we have

THEOREM 5. (i) For any h-table T and set of dependencies Σ, there is a h-table U, such that U represents the set $\Sigma(\mathfrak{I})$, where $\mathfrak{I}$ is the set of instances represented by T.
(ii) There is a p-table T and a functional dependency σ, such that no c-table represents $\sigma(\mathfrak{I})$, where $\mathfrak{I}$ is the set of instances represented by T.

PROOF. For (i) one uses the algebraic counterparts of the dependencies [1, 21] in the generalized chase of [10]. ∎

The process of computing the completion, given a table T and a set Σ, will be called *dependency enforcement*. The data complexity of dependency enforcement is the subject for the rest of this paper. For complete databases $\Sigma(I)$ can be computed in time polynomial in the number of facts in I [19].

THEOREM 6. Let T be h-table and $\mathfrak{I}$ the set of instances it represents, and let Σ be a set of equality generating dependencies and one join dependency. Then it is possible to compute a h-table U that represents $\Sigma(\mathfrak{I})$, in time polynomial in the size of T.

PROOF. The proof is based on the fact that a join dependency requires only one application to converge [16]. This property carries over to the generalized chase of tables.•

The theorem means that the information contained in the dependencies can be efficiently incorporated into the database, yielding a h-table of size polynomial in the size of the original table. Since the resulting table is a Horn table, it will allow efficient subsequent query evaluation in the cases mentioned in Section 3. Furthermore, the allowed class of dependencies is of practical interest.

In order to exhaust the computational complexity of dependency enforcement we shall consider a variant of the possibility problem.

Let Σ be a set of dependencies, k a positive integer and x one of (u, h, c, g, e, i, p). Then the *possibility problem* is, given a set P of at most k facts and a x-table representing a set $\mathbf{T}$ of instances, to decide if there is an instance J in $\Sigma(\mathbf{T})$, such that all facts of P are facts of J. The problem is denoted POSS(k, Σ, x).

THEOREM 7.

(i) POSS(k, Σ, u) is in NP.

(ii) If Σ is a set of equality generating dependencies and one join dependency, then POSS(k, Σ, h) is in PTIME.

(iii) There is a functional dependency σ, such that POSS(1, σ, c) is NP-complete.

(iv) There is a set of dependencies Σ, such that POSS(1, Σ, p) is NP-complete.

PROOF. For (i) we guess a valuation h, and check if P is contained in Σ(h(T)). It follows from [19] that Σ(h(T)) can be computed in time polynomial in h(T). (ii) follows from Theorem 6 and Theorem 2. For (iii) and (iv) we use reductions from the 3CNF satisfiablity problem [8].•

The results of Theorem 7 are summarized in Figure 8.

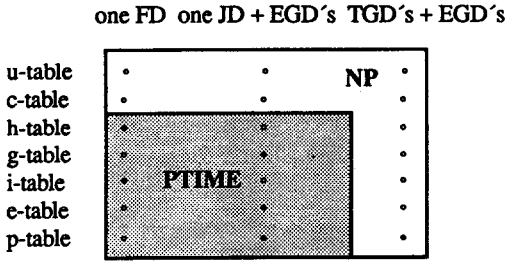

FIGURE 8. The possiblity problem for dependencies.

If the user is only concerned with the subsequent evaluation of the certain answer to datalog queries we can allow a larger class of dependencies. In this case we compute a table that approximates the required result, in such a way that when queries are evaluated no corruption of the information is visible.

Let T and U be tables, and $\mathbf{T}$ resp. $\mathbf{J}$ the set of instances they represent. Let Σ be a set of dependencies. Then we say that U *sufficiently* represents $\Sigma(\mathbf{T})$, if for any datalog query, the certain answers when evaluated on $\mathbf{J}$ and on $\Sigma(\mathbf{T})$ are the same.

TEOREM 8. (i) Let T be a h-table , and $\mathbf{T}$ the set of instances it represents. Let Σ be a set of total dependencies. Then there is an e-table U that sufficiently represents $\Sigma(\mathbf{T})$.
(ii) There is a c-table T, and a set of dependencies Σ, such that no e-table sufficiently represents $\Sigma(\mathbf{T})$, where $\mathbf{T}$ is the set of instances represented by T.

PROOF. For (i) we use a variation of the generalized chase process [10].•

Theorem 8 (i) strengthens a result in [11], where a similar property was proved for e-tables under the open world assumption.

THEOREM 9. Let T, Σ and U be as in Theorem 8 (i). Then U can be computed in time polynomial in the size of T.•

Finally we shall consider the complexity of computing a sufficient representation for all tables in the hierarchy. For this we use a variation of the certainty problem.

Let Σ be a set of dependencies, and let x one of (u, h, c, g, e, i, p). Then the *certainty problem* is, given a set P of facts and a x-table representing a set $\mathbf{T}$ of instances, to decide if all facts of P are facts of Σ(I), for all I in $\mathbf{T}$.

The certainty problem for dependencies is dencted CERT(Σ, x). We now have

THEOREM 10.

(i) For any set Σ of dependecies, CERT(Σ, u) is in coNP.

(ii) For any set Σ of dependencies, CERT(Σ, h) is in PTIME.

(iii) CERT($\emptyset$, c) is coNP-complete.

PROOF. (i) The complement of the problem is to verify whether there exists a valuation h, such that P is not included in $\Sigma(h(T))$. The latter problem is in NP. (ii) follows from Theorem 9 and Theorem 4, and (iii) follows directly from Theorem 3 (iii). •

These results are summarized in Figure 9.

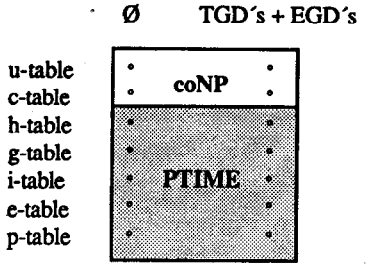

FIGURE 9. The certainty problem for dependencies.

The complexity result of this section are related to those of [9] and [19]. The first paper uses a weaker form of completion and considers the combined complexity, i.e. a measure of the size on the database and the dependency set. Theorem 10 (ii) can be seen as a strengthening of Theorem 4.2 in [19], where the same bound is proved for a construct equivalent to a p-table where some columns contain only constants and the rest of the columns contain only (distinct) variables. The hardness results of [19] are not directly comparable to ours, since [19] has a different interpretation of the closed world assumption.

5. CONCLUSIONS

We have complemented the representation hierachy with a construct called Horn tables. The class of Horn tables is algebraically closed under any set of total dependencies, and it is the only class in the hierarchy with this property (apart from the top element in the hierachy, which is of theoretical interest only). All sets of dependencies consisting of equality generating dependencies and one join dependency can be efficiently incorporated into a Horn table. The certain answer to any datalog query can be evaluated in polynomial time in the size of the Horn table. The same is true for the possible answer to any positive existential query. If one is concerned only with certain answers to datalog queries, any set of dependencies can be efficiently incorporated into a Horn table so that no corruption of the information occurs w.r.t. the answers.

In a forthcoming work we shall deal with the problem of updating Horn tables.

ACKNOWLEDGEMENT

This work was financially supported by the Academy of Finland.

REFERENCES

1. Abiteboul, S., Algebraic Analogues to fundamental notions of query and dependency theory. Rapports de Recherche 201. Institut National de Recherche en Informatique et en Automatique. Roquencourt, Avril 1983.

2. Abiteboul, S., Kanellakis, P., Grahne, G., On the representation and querying of sets of possible worlds. *Proc. ACM SIGMOD Internat. Conf. on Management of Data.* San Francisco, May 27-29, 1987, pp. 34-48.

3. Chandra, A. K., Harel, D., Structure and complexity of relational queries. *J. Comput. System. Sci. 25,* 1 (Aug. 1982), 99-128.

4. Codd, E.F., Extending the database relational model to capture more meaning. *ACM Trans Database Syst. 4,* 4 (Dec. 1979), 379-434.

5. Fagin, R., Horn clauses and database dependencies. *J. Assoc. Comput. Mach. 29,* 4 (Oct. 1982), 952-985.

6. Fagin, R., Mendelzon, A. O., Ullman, J. D., A simplified universal relation assumption and its properties. *ACM Trans. Database Syst. 7,* 3 (Sept. 1982), 343-360.

7. Fagin, R., Vardi, M. Y., The theory of data dependencies - a survey. In *Mathematics of Information Processing* (M. Anshel and W. Gewritz, eds). American Mathematical Society, Providence, 1986, pp. 17-91.

8. Garey, M. R., Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP -Completeness.* W. H. Freeman and Company, San Francisco, 1979.

9. Graham, M., Mendelzon, A. O., Vardi, M. Y., Notions of dependency satisfaction. *J. Assoc. Comput. Mach. 33,* 1 (Jan. 1986), 105-129.

10. Grahne, G., Dependency satisfaction in databases with incomplete information. *Proc. 10th Internat. Conf. on Very Large Data Bases.* Singapore, Aug. 27-31, 1984, pp. 37-45.

11. Imielinski, T., Lipski, W., Incomplete information and dependencies in relational databases. *Proc. ACM SIGMOD Internat. Conf. on Management of Data.* San Jose, May 23-26, 1983, pp. 178-184.

12. Imielinski, T., Lipski, W., Incomplete information in relational databases. *J. Assoc. Comput. Mach. 31,* 4 (Oct. 1984), 761-791.

13. Maier, D., *The Theory of Relational Databases.* Computer Science Press, Rockville, MD, 1983.

14 Reiter, R., On closed world databases. In: Gallaire, H., Minker, J. (eds), *Logic and Databases.* Plenum Press, New York, 1978., pp. 56-76.

15. Reiter, R., Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling* (M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, eds) Springer-Verlag, New York, 1984, pp. 191-233.

16. Sagiv, Y., On computing restricted projections of the representaive instance. *Proc. Fourth ACM SIGACT-SIGMOD Symp. on Principles of Database Systems.* Portland, March 25-27, 1985, pp. 171-180.

17. Ullman, J. D. *Principles of Database Systems, Second Edition.* Computer Science Press, Potomac, MD, 1982.

18. Vardi, M. Y., The complexity of relational query languages. *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, May 1982, pp. 137-146.

19. Vardi, M. Y., On the integrity of databases with incomplete information. *Proc. 5th ACM Symp. on Principles of Database Systems*, Boston, March 1986, pp. 252-266.

20. Vardi, M. Y., Querying logical databases. *J. Comput. System. Sci. 33,* 2 (Oct. 1986), 142-160.

21. Yannakakis, M., Papadimitriou, C. H., Algebraic dependencies. *J. Comput. System. Sci. 25,* 1 (Aug. 1982), 2-41.