AI Magazine Volume 18 Number 3 (1997) (© AAAI)

Logic and Databases

Past, Present, and Future

Jack Minker

■ At a workshop held in Toulouse, France, in 1977, Gallaire, Minker, and Nicolas stated that logic and databases was a field in its own right. This was the first time that this designation was made. The impetus for it started approximately 20 years ago in 1976 when I visited Gallaire and Nicolas in Toulouse, France. In this article, I provide an assessment about what has been achieved in the 20 years since the field started as a distinct discipline. I review developments in the field, assess contributions, consider the status of implementations of deductive databases, and discuss future work needed in deductive databases.

The use of logic and deduction in databases, as noted in Minker (1988b), started in the late 1960s. Prominent among developments was work by Levien and Maron (1965) and Kuhns (1967), and by Green and Raphael (1968a), who were the first to realize the importance of the Robinson (1965) resolution principle for databases. For early uses of logic in databases, see Minker (1988b), and for detailed descriptions of many accomplishments made in the 1960s, see Minker and Sable (1970).

A major influence on the use of logic in databases was the development of the field of logic programming: Kowalski (1974) promulgated the concept of logic as a programming language, and Colmerauer and his students developed the first Prolog interpreter (Colmerauer et al. 1973). I refer to logic programs that are function free as deductive databases (DDBs), or as datalog. I do so because databases are finite structures. Most of the results discussed can be extended to include logic programming.

The impetus for the use of logic in databases came about through meetings in 1976 in Toulouse, France, when I visited Herve Gallaire and Jean-Marie Nicolas while on sabbatical. The idea of a workshop on logic and databases was also conceived at this time. It is clear that a number of other individuals also had the idea of using logic as a mechanism to handle databases and deduction, and they were invited to participate in the workshop. The book *Logic and Data Bases* (1978), edited by Gallaire and Minker, was highly influential in the development of the field, as were the two volumes of *Advances in Database Theory* (Gallaire, Minker, and Nocholas 1984a, 1981) that were the result of two subsequent workshops held in Toulouse. Another influential development was the article by Gallaire, Minker, and Nicolas (1984b), which surveyed work in the field to that point.

The use of logic in databases was received by the database community with a great deal of skepticism: Was deductive databases (DDBs) a field? Did DDBs contribute to database theory or practice (Harel 1980)? The accomplishments I cite in this article are testaments to the fact that logic has contributed significantly both to the theory and the practice of databases. It is clear that logic has everything to do with the theory of databases, and many of those who were then critical of the field have changed their position. In the remainder of this article, I describe what I believe to be the major intellectual developments in the field, the status of commercial implementations, and future trends. As we see, the field of logic and databases has been prolific.

Intellectual Contributions of Deductive Databases

In 1970, Codd (1970) formalized databases in terms of the relational calculus and the relational algebra. He provided a logic language and the relational calculus and described how to compute answers to questions in the relational algebra and the relational calculus. Both

Formalizing databases through logic has played a significant role in our understanding of what constitutes a database, what is meant by a query, what is meant by an answer to a query, and how databases can be generalized for knowledge bases.

the relational calculus and the relational algebra provide declarative formalisms to specify queries. This was a significant advance over network and hierarchic systems (Ullman 1989, 1988), which only provided procedural languages for databases. The relational algebra and the relational calculus permitted individuals who were not computer specialists to write declarative queries and have the computer answer the queries. The development of syntactic optimization techniques (Ullman 1989, 1988) permitted relational database systems to retrieve answers to queries efficiently and compete with network and hierarchic implementations. Relational systems have been enhanced to include views. A *view*, as used in relational databases, is essentially a nonrecursive procedure. There are numerous commercial implementations of relational database systems for large database manipulation and for personal computers. Relational databases are a forerunner of logic in databases.

Although relational databases used the language of logic in the relational calculus, it was not formalized in terms of logic. The formalization of relational databases in terms of logic and the extensions that have been developed are the focus of this article. Indeed, formalizing databases through logic has played a significant role in our understanding of what constitutes a database, what is meant by a query, what is meant by an answer to a query, and how databases can be generalized for knowledge bases. It has also provided tools and answers to problems that would have been extremely difficult without the use of logic.

In the remainder of the article, I focus on some of the more significant aspects contributed by logic in databases: (1) a formalization of what constitutes a database, a query, and an answer to a query; (2) a realization that logic programming extends relational databases; (3) a clear understanding of the semantics of large classes of databases that include alternative forms of negation as well as disjunction; (4) an understanding of relationships between model theory, fixpoint theory, and proof procedures; (5) an understanding of the properties that alternative semantics can have and their complexity; (6) an understanding of what is meant by integrity constraints and how they can be used to perform updates, semantic query optimization (SQO), cooperative answering, and database merging; (7) a formalization and solutions to the update and view-update problems; (8) an understanding of bounded recursion and recursion and how they can be implemented in a practical manner; (9) an understanding of the relationship between logic-based systems and knowledge-based systems; (10) a formalization of how to handle incomplete information in knowledge bases; and (11) a correspondence that relates alternative formalisms of nonmonotonic reasoning to databases and knowledge bases.

I address the area of implementations of DDBs in Implementation Status of Deductive Databases, where commercial developments have not progressed as rapidly as intellectual developments. I then discuss some trends and future directions in Emerging Areas and Trends.

Formalizing Database Theory

Reiter (1984) was the first to formalize databases in terms of logic and noted that underlying relational databases were a number of assumptions that were not made explicit. One assumption deals with negation, that facts not known to be true in a relational database are assumed to be false. This assumption is the well-known *closed-world assumption* (CWA), expounded earlier by Reiter (1978). The *unique-name assumption* states that any item in a database has a unique name and that individuals with different names are different. The *domain-closure assumption* states that there are no other individuals than those in the database. Reiter then formalized relational databases as a set of ground assertions over a language $\mathfrak{L}$ together with a set of axioms. The language $\mathfrak{L}$ does not contain function symbols. These assertions and axioms are as follows:

Assertions: $R(a_1, \cdots, a_n)$, where R is an n-ary relational symbol in $\mathfrak{L}$, and $a_1 \cdots, a_n$ are constant symbols in $\mathfrak{L}$.

Unique-name axiom: If $a_1, \cdots, a_p$ are all the constant symbols of $\mathfrak{L}$, then

$$(a_1 \neq a_2), \cdots, (a_1 \neq a_p), (a_2 \neq a_3), \cdots, (a_{p-1} \neq a_p) \ .$$

Domain-closure axiom: If $a_1, \cdots, a_p$ are all the constant symbols of $\mathfrak{L}$, then

$$\forall X((X = a_1) \vee \cdots \vee (X = a_p)) \ .$$

Completion Axioms: For each relational symbol R, if $R(a_1^1, \cdots a_n^1), \cdots, R(a_1^m, \cdots, a_n^m)$ denote all facts under R, the completion axiom for R is

$$\forall X_1 \cdots \forall X_n (R(X_1, \cdots, X_n) \rightarrow (X_1 = a_1^1 \wedge \cdots \wedge X_n = a_n^1) \vee \cdots \vee (X_1 = = a_1^m \wedge \cdots \wedge X_n = a_n^m)) \ .$$

Equality Axioms:

$$\forall X(X = X)$$
$$\forall X \forall Y ((X = Y) \rightarrow (Y = X))$$
$$\forall X \forall Y \forall Z ((X = Y) \wedge (Y = Z) \rightarrow (X = Z))$$
$$\forall X_1 \cdots \forall X_n (P(X_1, \cdots, X_n) \wedge (X_1 = Y_1) \wedge \cdots \wedge (X_n = Y_n) \rightarrow P(Y_1, \cdots Y_n)) \ .$$

Example 1 illustrates the translation of a small database to logic. It is clear that handling such databases through conventional techniques will lead to a faster implementation. However, it serves to formalize previously unformalized databases.

The *completion axiom* was proposed by Clark (1978) as the basis for his negation-as-failure rule: It states that the only tuples that a relation can have are those that are specified in the relational table. This statement is implicit in every relational database. The completion axiom makes this explicit. Another contribution of logic programs and databases is that the formalization of relational databases in terms of logic permits the definition of a query and an answer to a query to be defined precisely. A *query* is a statement in the first-order logic language $\mathfrak{L}$. $Q(a)$ is an answer to a query, $Q(X)$, over a database *DB* if $Q(a)$ is a logical consequence of *DB*.

Deductive Databases

Relational databases are a special case of DDBs. A DDB can be considered as a theory, *DDB*, in which the database consists of a set of ground assertions, referred to as the *extensional database* (EDB), and a set of axioms, referred to as the intensional database (IDB), of the form

$$P \leftarrow Q_1, \ldots, Q_n \ , \qquad (1)$$

where P, Q_1, …, Q_n are atomic formulas in the language $\mathfrak{L}$. Databases of this form are termed datalog databases (Ullman 1989, 1988). A *datalog database* is a particular instance of a more general Horn logic program that permits function symbols in clauses given by formula 1. The recognition that logic programs are significant for databases was understood by a number of individuals in 1976 (see Gallaire and Minker [1978] for references). The generalization permits views to be defined that are recursive.

The recognition that logic programming and databases are fundamentally related has led to more expressive and powerful databases than is possible with relational databases defined in terms of the relational algebra.

That logic programming and DDBs are fundamentally related is a consequence of the fact that databases are function-free logic programs. As shown in many papers and, in particular, Gottlob (1994), the expressive power of logic programming extends that of relational databases.

In addition to defining a database in terms of an EDB and an IDB, it is necessary to formalize what is meant by an integrity constraint. Kowalski (1978) suggests that an *integrity constraint* is a formula that is consistent with the DDB, but for Reiter (1984) and Lloyd and Topor (1985), an integrity constraint is a theorem of the DDB. For alternative definitions of integrity constraints, see Reiter (1990, 1988) and Demolombe and Jones (1996).

Consider the family database to consist of the *Father* relation with schema *Father*(*father, child*) and the *Mother* relation with schema *Mother*(*mother, child*). Let the database be

FATHER	*father*	*child*
	j	*m*
	j	*s*

MOTHER	*mother*	*child*
	r	*m*
	r	*s*

The database translated to logic is given as follows; we do not include the equality axioms because they are obvious.

Assertions: *Father*(*j, m*), *Father*(*j, s*), *Mother*(*r, m*), *Mother*(*r, s*), where *Father* and *Mother* are predicates, and *j, m, s,* and *r* are constants.

Unique-Name Axiom:

$((j \neq m), (j \neq s), (j \neq r), (r \neq m), (r \neq s), (m \neq s))$.

Domain-Closure Axiom:

$(\forall X)((X = j) \vee (X = m) \vee (X = s) \vee (X = r))$.

Completion Axioms:

$(\forall X_1 \forall X_2)(Father(X_1, X_2) \leftarrow ((X_1 = j) \wedge (X_2 = m)) \vee ((X_1 = \mathrm{j}) \wedge (X_2 = s)))$.

$(\forall X_1 \forall X_2)(Mother(X_1, X_2) \leftarrow ((X_1 = r) \wedge (X_2 = m)) \vee ((X_1 = r) \wedge (X_2 = s)))$.

Example 1. Translation of a Small Database to Logic.

In DDBs, the semantic aspects of a database's design can be captured by integrity constraints. Information about *functional dependencies*—that a relation's key functionally determines the rest of the relation's attribute—can be written via integrity constraints. For example, assume the predicate *flight* for an airline database and that the attributes *Airline* and *No.* are a composite key for the relation. One of the functional dependencies—that the departure time is functionally determined by airline and flight number—is represented by

Dtime[1] = *Dtime*[2] ⇐
flight(*Airline, No., Dtime*[1],-, …,-),
flight(*Airline, No., Dtime*[2],-, …,-) ,

where ⇐ is used to distinguish a rule from an integrity constraint.

Likewise, *inclusion dependencies,* which are

also common semantic information from a database's design, are easily represented. For example, say the predicate *airport* records various information about airports known to the database. We want to ensure that any airport that serves as a departure or an arrival of any flight known to the database is also in the airport relation. The first of these—that the departure airport is known—could be represented as follows:

airport(-, ..., -, *Fieldcode*) ⇐ *flight*(-, ..., -, *Fieldcode*) .

The major use made of integrity constraints has been in updating, to assure that the database is consistent. Nicolas (1979) used techniques from DDBs to speed database update. Blaustein (1981) has also made contributions to this problem. Reiter (1978a) showed that one can query a Horn database with or without integrity constraints, and the answer to the query is the same. However, integrity constraints can be used to advantage in the query process. Although integrity constraints do not affect the result of a query, they can affect the efficiency with which answers can be computed. Integrity constraints provide semantic information about data in the database. If a query requests a join for which there will never be an answer because of system constraints, then an unnecessary join on two potentially large relational tables in a relational database system or performing a long deduction in a DDB is not needed when integrity constraints imply the answer is empty. The process of using integrity constraints to constrain a search is called semantic query optimization (SQO) (Chakravarthy, Grant, and Minker 1990). McSkimin and Minker (1977) were the first to use integrity constraints for SQO in DDBs. Hammer and Zdonik (1980) and King (1981) were the first to apply SQO to relational databases. Chakravarthy, Grant, and Minker (1990) formalized SQO and developed the partial subsumption algorithm and method of residues. The partial subsumption algorithm and method of residues provides a general technique applicable to any relational database or DDB that is able to perform SQO. The general approach to SQO described in Chakravarthy, Grant, and Minker (1990) has been extended to perform bottom-up evaluation (Godfrey, Gryz, and Minker 1996); to include databases with negation in the body of clauses (Gaasterland and Lobo 1993); and to handle recursive IDB rules (Levy and Sagiv 1995).

A cooperative answering system provides information to users about why a particular query succeeded or failed.

A topic related to SQO is that of cooperative answering systems. A *cooperative answering system* provides information to users about why a particular query succeeded or failed. When a query fails, a user, in general, cannot tell why the failure occurred. There can be several reasons: The database currently does not contain information to respond to the user, or there will never be an answer to the query. The distinction could be important to the user. Another aspect related to integrity constraints is that of user constraints. A *user constraint* is a formula that models a user's preferences. It can constrain providing answers to queries in which the user might have no interest (for example, stating that in developing a route of travel, the user does not want to pass through a particular city) or provide other constraints that might restrict the search. As shown by Gaasterland, Godfrey, and Minker (1992b), user constraints, which are identical in form to integrity constraints, can be used for this purpose. Although integrity constraints provide the semantics of the entire database, user constraints provide the semantics of the user. User constraints can be inconsistent with the database; hence, these two semantics are maintained separately. To maintain the consistency of the database, only integrity constraints are relevant. A query can be thought of as the conjunction of the query itself and the user constraints. A query can be optimized semantically based on both integrity constraints and user constraints.

As noted previously, integrity constraints are versatile; they do more than just represent dependencies. General semantic information can be captured as well. Assume that at the national airport in Washington, D.C. (DCA), that no flights are allowed (departures or arrivals) after 10:00 PM or before 8:00 AM because the airport is downtown, and night flights would disturb city residents. This information can be captured as an integrity constraint.

Such knowledge, captured and recorded as integrity constraints, can be used to answer queries to the database more intelligently and more informatively. If someone asks for flights out of DCA to, say, Los Angeles International Airport leaving between 10:30 PM and 12:00 AM, the database could simply return the empty answer set. (There will be no such flights if the database is consistent with its constraints.) It would be better, however, for the database system to inform the querier that there can be no such flights because of the Washington, D.C., flight regulations.

Using user constraints and integrity constraints, one can develop a system that informs users why a query succeeds or fails (Gaasterland et al. 1992). Other features can be incorporated, such as the ability to relax a query, termed *query relaxation*, given that it fails, so that an answer to a related request can be found. See

Gaasterland, Godfrey, and Minker (1992a) for a survey of cooperative answering systems.

SQO, user constraints, and cooperative answering systems are important contributions both for relational database and DDB systems. They will eventually be incorporated into commercial relational database and DDB systems.

Indeed, I cannot imagine a DDB developed for commercial systems to be successful if it does not contain both SQO and cooperative answering capabilities. How can one expect users to understand why deductions succeed or fail if such information is not provided? How can queries doomed to fail because they violate user constraints or integrity constraints be allowed to take up a significant amount of search time if the query cannot possibly succeed? I also believe that these techniques must be incorporated into relational technology. As discussed in Implementation Status of Deductive Databases, systems are beginning to incorporate SQO techniques. Practical considerations of performing in a bottom-up approach have been addressed by Godfrey, Gryz, and Minker (1996).

Extended Deductive Database Semantics

The first generalization of relational databases was to permit function-free recursive Horn rules in a database, that is, rules in which the head of a rule is an atom, and the body of a rule is a conjunction of atoms. These databases are DDBs, or datalog databases. Subsequently, other DDBs that might contain negated atoms in the body of rules were permitted. These alternative extensions and their significance are described in the following subsections.

Horn Semantics and Datalog One of the early developments was by van Emden and Kowalski (1976), who wrote a seminal paper on the semantics of Horn theories. Van Emden and Kowalski made a significant contribution to logic and databases by recognizing that the semantics of Horn databases can be characterized in three distinct ways: (1) model theory, (2) fixpoint theory, and (3) proof theory. These three characterizations lead to the same semantics.

Model theory deals with a collection of models that capture the intended meaning of the database. *Fixpoint theory* deals with a fixpoint operator that constructs the collection of all atoms that can be inferred to be true from the database. *Proof theory* deals with a procedure that finds answers to queries with respect to the database. van Emden and Kowalski (1976) showed that the intersection of all Herbrand models of a Horn DDB is a unique minimal model. The unique minimal model is the same as all the atoms in the fixpoint and are the only atoms provable from the theory.

To find if the negation of a ground atom is true, one can subtract, from the *Herbrand base* (the set of all atoms that can be constructed from the constants and the predicates in the database), the minimal Herbrand model. If the atom is contained in this set, then it is assumed to be false, and its negation is true. Alternatively, answering queries that consist of negated atoms that are ground can be achieved using negation-as-finite failure as described by Reiter (1978b) and Clark (1978).

The first approaches to answering queries in DDBs did not handle recursion and were primarily top-down (or backward reasoning) (Gallaire and Minker 1978). Answering queries in relational database systems was a bottom-up (or forward-reasoning) approach because all answers are usually required, and it is more efficient to do so in a bottom-up approach. The major approaches to handling recursion are based on the renaming of the Alexander (Rohmer, Lescoeur, and Kerisit 1986) and magic set (Bancilhon et al. 1986) methods, which make use of constants that appear in a query and perform search by bottom-up reasoning. Bry (1990) reconciles the bottom-up and top-down methods to compute recursive queries. He shows that the Alexander and magic set methods based on rewriting and the methods based on resolution implement the same top-down evaluation of the original database rules by means of auxiliary rules processed bottom-up. For pioneering work on recursion and alternative methods, see Minker (1996). Minker and Nicolas (1982) were the first to show that there are forms of rules that lead to *bounded recursion,* in which the deduction process must terminate in a finite number of steps. This work has been extended by Naughton and Sagiv (1987). Example 2 illustrates a rule that terminates finitely regardless of the state of the database.

The efficient handling of recursion and the recognition that some recursive cases might inherently be bounded contributes to the practical implementation of DDBs. An understanding of the relationship between resolution-based (top-down) and fixpoint–based (bottom-up) techniques and how the search space of the latter can be made identical to top-down resolution with program transformation is another contribution of DDBs.

Extended Deductive Databases and Knowledge Bases The ability to develop a semantics for theories in which there are rules with a *literal* (that is, an atomic formula or the

I cannot imagine a DDB developed for commercial systems to be successful if it does not contain both SQO and cooperative answering capabilities.

If a rule satisfies the condition that it is singular, then it is bound to terminate in a finite number of steps independent of the state of the database. A recursive rule is singular if it is of the form

$R \leftarrow F \wedge R_1 \wedge \ \ldots \wedge \ R_n$,

where F is a conjunction of possibly empty base (that is, EDB) relations and R, R_1, R_2, …, R_n are atoms that have the same relation name iff (1) each variable that occurs in an atom R_i and does not occur in R only occurs in R_i and (2) each variable in R occurs in the same argument position in any atom R_i where it appears, except perhaps in at most one atom R_1 that contains all the variables of R.

Thus, the rule

$R(X, Y, Z) \leftarrow R(X, Y', Z), R(X, Y, Z')$

is singular because (1) Y' and Z' appear, respectively, in the first and second atoms in the head of the rule (condition 1) and (2) the variables X, Y, and Z always appear in the same argument position (condition 2).

Example 2. Bounded Recursion.

The rules

r_1: $p \leftarrow q$, *not* r
r_2: $q \leftarrow p$
r_3: $q \leftarrow s$
r_4: s
r_5: $r \leftarrow t$

make up a stratified theory. Rule r_5 is in the lowest stratum, but the other rules are in a higher stratum. The predicate p is in a higher stratum than the stratum for r because it depends negatively on r. q is in the same stratum as p because it depends on p. s is also in the same stratum as q. The meaning of the stratified program is that s, q, and p are true, but t and r are false. t is false because there is no defining rule for t. Because t is false, r is false. s is given as true; hence, q is true. Because q is true, and r is false, from rule r_1, p is true.

Example 3. Stratified Program.

negation of an atomic formula) in the head and literals with possibly negated-by-default literals in the body of a clause has significantly expanded the ability to write and understand the semantics of complex applications. Such clauses, referred to as *extended clauses,* are given by

$L \leftarrow M_1, \cdots, M_n, \textit{not } M_{n+1}, \cdots \textit{not } M_{n+k}$, (2)

where L and the M_j, $j = 1, \cdots, (n + k)$ are literals. Such databases combine both classical negation and default negation (represented by *not* immediately preceding a literal) and are referred to as extended DDBs. The combining of classical and default negation provides users with greater expressive power.

Logic programs where default negation can appear in the body of a clause first appeared in the Workshop on Foundations of Deductive Databases and Logic Programming in August 1986. Selected papers from the workshop were published in Minker (1988a). The concept of stratification was introduced to logic programs by Apt, Blair, and Walker (1988) and Van Gelder (1988), who considered stratified theories in which L and the M_j in formula 2 are atomic formulas, and there is no recursion through negation. Apt, Blair, and Walker, and Van Gelder, show that there is a unique preferred minimal model, computed from strata to strata. Przymusinski (1988) termed this minimal model the *perfect model*. When one has a theory that is stratified, one can place clauses in different strata, where predicates in the head of a rule are in a higher stratum than predicates that are negated in the body of the clause, as explained in example 3. Thus, one can compute the positive predicates in a lower stratum, and the negated predicate's complement is true in the body of the clause if the positive atom has not been computed in the lower stratum.

The theory of stratified databases was followed by permitting recursion through negation in formula 2, where the L and M_j are atomic formulas. Example 4 illustrates a database that cannot be stratified. In the context of DDBs, they are called *normal DDBs*. Many papers have been devoted to defining the semantics of these databases. A summary of these semantics is given in Minker and Ruiz (1994). The most prominent of this work for the Horn case are the well-founded semantics (WFS) of Van Gelder, Ross, and Schlipf (1988) and the stable semantics of Gelfond and Lifschitz (1988). The WFS leads to a unique three-valued model. Stable semantics can lead to a collection of minimal models. For some DDBs, this collection can be empty. Fitting (1985) also defined a three-valued model to capture the semantics of normal logic programs. For additional work, see Minker (1996).

There have been several implementations of the WFS. Chen and Warren (1993) developed a top-down approach to answer queries in this semantics, while Leone and Rullo (1992) developed a bottom-up method for datalog databases. Several methods have been developed for computing answers to queries in stable model semantics. Fernández et al. (1993) developed a bottom-up approach based on the concept of model trees. Every branch of a model tree is a model of the database, where a node in a tree is an atom that is shared by each branch below the node. (See example 6 for an illustration of a model tree.) Bell et al. (1993) developed a method based on linear programming. See Minker (1996) for additional methods to compute the well-founded, the stable model, and other related semantics.

A further extension of normal DDBs, proposed by Gelfond and Lifschitz (1990) and Pearce and Wagner (1989), permits clauses in formula 2, where L and M_j are literals, and, therefore, combines classical and default negation in one database. The semantics for normal DDBs was described by Minker and Ruiz (1994).

These notions of default negation have been used as separate ways to interpret and deduce default information. That is, each application has chosen one notion of negation and applied it to every piece of data in the domain of the application. Minker and Ruiz (1996) defined a new class of more expressive DDBs that allow several forms of default negation in the same database. In this way, different pieces of information in the domain can be treated appropriately. They introduced a new semantics called the *well-founded stable semantics* that characterizes the meaning of DDBs that combine the well-founded and the stable semantics. Schlipf (1995) has written a comprehensive survey article on complexity results for DDBs.

The development of the semantics of extended DDBs that permit a combination of classical negation and multiple default negations in the same DDB are important contributions. The study and development of results in the computational complexity of these databases are important contributions to database theory. They permit wider classes of application to be developed.

Knowledge bases are important for AI and expert system developments. A general way to represent knowledge bases is through logic. Work developed for extended DDBs concerning semantics and complexity applies directly to knowledge bases. Baral and Gelfond (1994) describe how extended DDBs can be used to represent knowledge bases. For an example of a knowledge base, see example 5. Extended DDBs, together with integrity constraints, permit a wide range of knowledge bases to be implemented. Many papers devoted to knowledge bases consider them to consist of facts and rules, which is one aspect of a knowledge base, as is the ability to extract proofs. However, integrity constraints supply another aspect of knowledge and differentiate knowledge bases, which can have the same rules but different integrity constraints. One should define a knowledge base as consisting of an extended DDB plus integrity constraints.

Since alternative extended deductive semantics have been implemented, the knowledge base expert can now focus on the problem to be implemented, that is, on writing rules and integrity constraints that characterize the knowledge bases, selecting the particular semantics that meets the needs of the problem, and employing a DDB system that uses the required semantics. The field of DDBs has contributed to providing an understanding of knowledge bases and their implementation.

Consider the database given by

r_1: $p(X) \leftarrow not\ q(X)$
r_2: $q(X) \leftarrow not\ p(X)$
r_3: $r(a) \leftarrow p(a)$
r_4: $r(a) \leftarrow q(a)$.

Notice that clauses r_1 and r_2 are recursive through negation. Hence, the database is not stratifiable. According to the WFS, $\{p(a), q(a), r(a)\}$ are assigned unknown. However, for the stable model semantics, there are two minimal models: $\{\{p(a), r(a)\}, \{q(a), r(a)\}\}$. Hence, one can conclude that $r(a)$ is true, but the disjunct, $p(a) \vee q(a)$ is true in the stable model semantics.

Example 4. Nonstratifiable Database.

Extended Disjunctive Deductive Database Semantics

In the databases discussed previously, information is definite. However, there are many situations where our knowledge of the world is incomplete. For example, when a null value appears as an argument of an attribute of a relation, the value of the attribute is unknown. Uncertainty in databases can be represented by probabilistic information (Ng and Subrahmanian 1993). Another area of incompleteness arises when it is unknown which among several facts are true, but it is known that one or more

Consider the following database, where the predicate $p(X,Y)$ denotes that X is a professor in department Y, $a(X,Y)$ denotes that individual X has an account on machine Y, and $ab(W,Z)$ denotes that it is abnormal in rule W to be individual Z.

We want to represent the following information, where *mike* and *john* are professors in the Computer Science Department:

First, as a rule, professors in the Computer Science Department have Vax accounts. This rule is not applicable to *mike*. He might or might not have an account on this machine.

Second, every computer science professor has a Vax or an IBM account but not both. These rules can be captured in the following DDB:

$p(mike, cs) \leftarrow$
$p(john, cs) \leftarrow$
$\neg p(X, Y) \leftarrow not\ p(X, Y)$
$a(X, vax) \leftarrow p(X, cs), not\ ab(r4, X), not\ \neg a(X, vax)$
$ab(r4, mike) \leftarrow$
$a(X, vax) \vee a(X, ibm) \leftarrow p(X, cs), ab(r4, X)$
$\neg a(X, ibm) \leftarrow p(X, cs), a(X, vax)$
$\neg a(X, vax) \leftarrow p(X, cs), a(X, ibm)$
$a(X, ibm) \leftarrow a(X, vax), p(X, cs)$.

The third rule states that if by default negation, predicate $p(X, Y)$ fails, then $p(X,Y)$ is logically false. The other rules encode the statements listed previously.

From this formalization, one can deduce that *john* has a Vax account, but *mike* has either a Vax or an IBM account but not both.

Example 5. Knowledge Base (Baral and Gelfond 1994).

is true. Therefore, it is necessary to be able to represent and understand the semantics of theories that include incomplete data. A natural way to extend databases to include incomplete data is to permit disjunctive statements as part of the language, where clauses can have disjunctions in their heads. These clauses are represented as

$$L_1 \vee L_2 \vee \ldots \vee L_m \leftarrow M_1, \ldots, M_n, not\ M_{n+1}, \ldots not\ M_{n+k} \quad (3)$$

and are referred to as *extended disjunctive clauses*. Such databases are referred to as *extended disjunctive deduct ive databases* (EDDDBs). *Foundations of Disjunctive Logic Programming* by Lobo, Minker, and Rajasekar (1992) describes the theory of disjunctive logic programs and includes several chapters on *disjunctive deductive databases* (DDDBs). Example 5 illustrates the use of such a theory of databases.

I first discuss the semantics of DDDBs, where clauses are given by formula 3, where the literals are restricted to atoms, and there is no default negation in the body of a clause. Next, I discuss the semantics of EDDDBs, where there are no restrictions on clauses in formula 3.

Disjunctive Deductive Databases

As noted in Minker (1989), work in disjunctive theories was pursued seriously after a workshop organized in 1986 (Minker 1986). The field of DDDBs started approximately in 1982 with the appearance of a paper I wrote (Minker 1982), in which I described how one can answer both positive and negated queries in such databases. For a historical perspective of disjunctive logic programming and DDDBs, see Minker (1989). There is a major difference between the semantics of DDBs and those for DDDBs. Whereas DDBs typically have a unique minimal model that describes the meaning of the database, DDDBs generally have multiple minimal models.

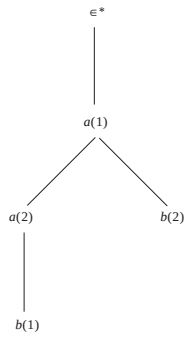
As shown in Minker (1982), it is sufficient to answer positive queries over DDDBs by showing that the query is satisfied in every minimal model of the database. Thus, in the DDDB $a \vee b$, there are two minimal models: (1) $\{a\}$ and (2) $\{b\}$. The query, a?, is not satisfied in the model b; hence, it cannot be concluded that a is true. However, the query $(a \vee b)$ is satisfied in both minimal models; hence, the answer to the query $a \vee b$ is yes. To answer negated queries, it is not sufficient to use Reiter's (1978) CWA because as he noted, from the theory $DB = a \vee b$, it is not possible to prove a, and it is not possible to prove b. Hence, by the CWA, *not a* and *not b* follow. However, $\{a \vee b, not\ a, not\ b\}$ is not consistent. The generalized closed-world assumption (GCWA) (Minker 1982) resolves this problem by specifying that a negated atom be considered true if the atom does not appear in any minimal model of the database. The GCWA provides a model-theoretic definition of negation. An equivalent proof-theoretic definition, also presented in Minker (1982), is that an atom a can be considered false if whenever $a \vee C$ can be proven from the database, then C can also be proven from the database, where C is an arbitrary positive clause. For related work on negation in disjunctive theories, see Minker (1996). For surveys on negation in DDBs and DDDBs, see Shepherdson (1987), Apt and Bol (1994), and Minker (1993).

In DDBs, it is natural for the fixpoint operator to map atoms to atoms. However, for DDDBs, it is natural to map positive disjunctions to positive disjunctions. A set of positive disjunctions is referred to as a *state*. A *model state* is a state whose minimal models all satisfy

the DDDB. The concept of a state was defined by Minker and Rajasekar (1990) as the domain of a fixpoint operator T_p whose least fixpoint characterizes the semantics of a disjunctive logic program P. The operator is shown to be monotonic and continuous; hence, it converges in a countably infinite number (ω) of iterations. The fixpoint computation operates bottom-up and yields a minimal model state that is logically equivalent to the set of minimal models of the program. The Minker-Rajasekar fixpoint operator is an extension of the van Emden–Kowalski fixpoint operator. If one considers all model states of a DDDB and intersects them, then the resultant is a model state, and among all model states, it is minimal. Hence, one obtains a unique minimal model in a Horn database, but one obtains a unique model state in a DDDB. See Decker (1991) for a related fixpoint operator for DDDBs.

Answering queries in DDDBs has been studied by a number of individuals, as described in Minker (1996). I focus on the work of Fernández and Minker (1991), who developed the concept of a model tree. They show how one can incrementally compute sound and complete answers to queries in hierarchical DDDBs. A model tree is shown in example 6. A DDDB is hierarchical if it contains no recursion. Fernández et al. (1993) show how one can develop a fixpoint operator over trees to capture the meaning of a DDDB that includes recursion. The tree representation of the fixpoint is equivalent to the Minker-Rajasekar fixpoint (Minker and Rajasekar 1990). Fernández and Minker compute the model tree of the extensional DDDB once. To answer queries, intensional database rules can be invoked. However, the models of the extensional disjunctive part of the database do not have to be generated for each query. Their approach to computing answers generalizes both to stratified and normal DDDBs.

Loveland and his students (Loveland, Reed, and Wilson 1993) have developed a top-down approach when the database is near Horn. They have developed a case-based reasoner that uses Prolog to perform the reasoning. This effort is one of the few that have implemented DDDBs. Loveland, Reed, and Wilson (1993) introduced a relevancy-detection algorithm to be used with SATCHMO, developed by Manthey and Bry (1988), for automated theorem proving. Their system, termed SATCHMORE (SATCHMO with RElevancy), improves on SATCHMO by limiting uncontrolled use of forward chaining. Seipel (1995) has developed a system, DISLOG, that incorporates many different disjunctive theories and strategies. The system is available on the World Wide Web.



Consider the following example given by the database: {$a(1)$; $a(2) \vee b(2)$; $b(1) \vee b(2)$}. There are two minimal models for this database {{$a(1)$, $a(2)$, $b(1)$}, {$a(1)$, b(2)}}. These models can be written as a tree (see figure above).

Example 6. Model Tree.

See Minker (1996) for references to work on the complexity of answering queries in disjunctive logic programs and Eiter and Gottlob (1995) for complexity results for propositional logic programs.

The development of model-theoretic, fixpoint, and proof procedures placed the semantics of DDDBs on a firm foundation. Methods to handle DDDBs are being developed and should eventually enhance implementations. The GCWA and alternative theories of negation have enhanced our understanding of default negation in DDDBs. Complexity results provide an understanding of the difficulties in finding answers to queries in such systems.

Extended Disjunctive Deductive Databases

Fernández and Minker (1995) present a new fixpoint characterization of the minimal models of disjunctive and stratified DDDBs. They prove that by applying the operator iteratively, in the limit, it constructs the perfect model semantics (Przymusinski 1988) of stratified DDDBs. Given the equivalence between the perfect model semantics of stratified programs and prioritized circumscription (Przymusinski 1988), their fixpoint characterization captures the meaning of the corresponding circumscribed theory. Based on these results, they present a bottom-up evaluation algorithm for

stratified DDDBs. This algorithm uses the model-tree data structure to represent the information contained in the database and to compute answers to queries. Fernández and Minker (1992) develop the theory of DDDBs using the concept of model trees. Work on updates in DDDBs is described in Fernández, Grant, and Minker (1996).

Four alternative semantics were developed for nonstratifiable normal DDDBs at approximately the same time: (1) Ross (1989), the strong WFS; (2) Baral, Lobo, and Minker (1990), the generalized disjunctive WFS (GDWFS); and (3, 4) two semantics by Przymusinski, an extension of the stable model semantics (Przymusinski 1990) for normal disjunctive databases and the stationary semantics (Przymusinski 1990). A number of other important semantics have been developed. Przymusinski (1995) describes a new semantic framework for disjunctive logic programs and introduces the static expansions of disjunctive programs. The class of static expansions extends both the classes of stable, well-founded, and stationary models of normal programs and the class of minimal models of disjunctive programs. Any static expansion of a program P provides the corresponding semantics for P consisting of the set of all sentences logically implied by the expansion. The stable model semantics has also been extended to disjunctive programs (Gelfond and Lifschitz 1991; Przymusinski 1991). The disjunctive WFS (DWFS) of Brass and Dix (1995) is also of considerable interest because it permits a general approach to bottom-up computation in disjunctive programs.

As noted previously, a large number of different semantics exist for both EDDBs and EDDDBs. A user who wants to use such a system is faced with the problem of selecting the appropriate semantics for his/her needs. No guidelines have been developed. However, many complexity results have been obtained for these semantics. Schlipf (1995) and Eiter and Gottlob (1995) have written comprehensive survey articles that summarize the complexity results that are known for alternative semantics.

In addition to the results for extended disjunctive theories, there is work in investigating tractable cases of disjunctive theories. Ben-Eliyahu and Dechter (1994) introduced the concept of a head-cycle–free (HCF) clause. Let a clause consist of a disjunction of literals. A dependency graph G_p is associated with each program P as follows:

First, each clause of the form, formula 2, and each predicate in P is a node.

Second, there is a positive (negative) arc from a predicate node p to a rule node ζ iff p appears positive (negative) in the body of ζ and an arc from ζ to p (resp., and also an arc from p to ζ) if p appears in the head of ζ.

The *positive dependency graph* of P is a subgraph of G_p containing only positive arcs. A directed cycle in G_p is called *negative* if it contains at least one negative arc. A DDB P is HCF if for every two predicate names p and q, if p and q are on a positive directed cycle in the dependency graph G_p, then there is no rule in P in which both p and q appear in the head. It is shown in Ben-Elyahu, Palopoli, and Zemlyanker (1996) that answers to queries expressed in this language can be computed in polynomial time. Furthermore, the language is sufficiently powerful to express all polynomial time queries. It is further shown in Ben-Elyahu and Palopoli (1994) that there is an algorithm that performs, in polynomial time, minimal model finding and minimal model checking if the theory is HCF. An efficient algorithm for solving the (co–NP-hard decision) problem of checking if a model is stable in function-free disjunctive logic programs is developed in Leone, Rullo, and Scarcello (1996). They show that the algorithm runs in polynomial time on the class of HCF programs, and in the case of general disjunctive logic programs, it limits the inefficient part of the computation only to the components of the program that are not HCF.

In addition to work on tractable databases, consideration has been given to approximate reasoning. In such reasoning, one can give up soundness or completeness of answers. Efforts have been developed both for DDBs and DDDBs by Kautz and Selman (1992) and Selman and Kautz (1996), who developed lower and upper bounds for Horn (datalog) databases and compilation methods; Cadoli (1993), who developed computational and semantic approximations; and del Val (1995), who developed techniques for approximating and compiling databases. See also Cadoli (1996) for additional references concerning compilation, approximation, and tractability of knowledge bases.

A second way to determine the semantics to be used is through their properties. Dix (1992) proposed a large number of criteria that are useful in determining the appropriate semantics to be used. Properties deemed useful are (1) *elimination of tautologies*, where one wants the semantics to remain the same if a tautology is eliminated; (2) *generalized principle of partial evaluation*, where if a rule is replaced by a one-step deduction, the semantics is unchanged; (3) positive-negative reduction; (4) *elimination of nonminimal rules*, where a subsumed rule is

eliminated, the semantics remains the same; (5) *consistency*, where the semantics is not empty for all disjunctive databases; and (6) *independence*, where if a literal *l* is true in a program *P*, and *P'* is a program whose language is independent of the language of *P*, then *l* remains true in the program consisting of the union of the two languages.

A semantics can have all the properties that one might desire and be computationally tractable and yet not provide answers that a user expected. If, for example, the user expected an answer *r*(*a*) in response to a query *r*(*X*), and the semantics were, for example, the WFS, the user would receive the answer that *r*(*a*) is unknown. However, if the stable model semantics had been used, the answer returned would be *r*(*a*). Perhaps, the best that can be expected is to provide users with complexity results and criteria by which they can decide which semantics meets the needs of their problems.

Understanding the semantics of disjunctive theories is related to nonmonotonic reasoning. The field of nonmonotonic reasoning has resulted in several alternative approaches to the performance of default reasoning (Moore 1985; McCarthy 1980; McDermott and Doyle 1980; Reiter 1980). The survey article by Minker (1993) and papers by Eiter and Gottlob (1995) and Cadoli and Schaerf (1993) cite results where alternative theories of nonmonotonic reasoning can be mapped into extended disjunctive logic programs and databases. Hence, DDDBs can be used to compute answers to queries in such theories. See Cadoli and Lenzerini (1994) for complexity results concerning circumscription and closed-world reasoning. See also Yuan and You (1993) for a description of the relationships between autoepistemic circumscription and logic programming. They use two different belief constraints to define two semantics: (1) the stable circumscriptive semantics and (2) the well-founded circumscriptive semantics for autoepistemic theories. The work in Yuan and You (1993) and that on static semantics developed by Przymusinski (1995) appear to be related.

Another area to which DDDBs have contributed is the null-value problem. If an attribute of a relation can have a null value, where this value is part of a known set, then one can represent this information as a disjunction of relations, where in each disjunction, a different value is given to the argument. For papers on the null-value problem both in relational and deductive databases, see Minker (1996).

There are several significant contributions of DDDBs:

First, greater expressive power is provided to the user to develop knowledge base systems.

Second, alternative concepts of negation have been developed as evidenced by the different semantics for logic programs (for example, WFS and stable semantics for extended logic programs and alternative semantics for disjunctive logic programs).

Third, complexity results have been found for alternative semantics of DDBs, including alternative theories of negation.

Fourth, methods have been developed to permit prototype systems to be implemented.

Fifth, DDBs can be used as the computational vehicle for a wide class of nonmonotonic-reasoning theories.

In this section, I showed how relational databases can be formalized in terms of logic, permitting databases to be extended beyond what is possible with relational databases. Various extensions were discussed, such as DDBs. EDDBs, DDDBs, and EDDDBs. Alternative theories of negation were discussed, and the semantics of the alternative databases, including negation, were described. These extensions were shown to be useful for developing complex knowledge bases. The role of integrity constraints and other constraints for such systems was described.

Implementation Status of Deductive Databases

The field of DDBs has made significant intellectual contributions over the past 20 years. However, these contributions have not been matched by implementations that are available in the commercial market. In the early 1970s, when Codd (1970) introduced the relational model, there were numerous debates in the database community about the efficacy of such systems relative to network and hierarchical systems (Date 1995). These debates ended when an effective relational system was implemented and shown to be comparable to these systems. Now, some of those individuals who are prominent in relational databases claim that DDBs are not effective and are not needed. Although I believe otherwise, these comments can be addressed better either when a full commercial implementation of a DDB is available or when many of the techniques introduced in DDBs find their way into relational databases. I believe that both of these are beginning to happen.

In the following subsection, I discuss the stages through which implementations of DDBs have progressed and some contributions made during each stage. Following this, I discuss the reasons why I believe that no current

The field of DDBs has made significant intellectual contributions over the past 20 years. However, these contributions have not been matched by implementations that are available in the commercial market.

systems are commercially marketed and speculate on how this situation might change.

Deductive Database Systems

There have been three stages of implementations of DDBs: (1) pre-1970, (2) 1970 to 1980, and (3) 1980 to the present. Each stage has contributed toward understanding the problems inherent in developing DDB systems.

First Stage: Pre-1970s Two efforts stand out during this period: the first by Levien, and Maron (1965) and Kuhns (1967), who developed a prototype system that demonstrated the feasibility of performing deduction in databases and the second by Green and Raphael (1968a, 1968b), who recognized that the resolution method of Robinson (1965) was a uniform procedure based on a single rule of inference that could be used for DDBs. This was the first general approach to DDBs. The work by Levien and Maron (1965) and Kuhns (1967) on Relational Data File (RDF) started in 1963. A procedural language, INFEREX, executed inference routines. Plausible and formal inferencing were both possible in RDF, as was temporal reasoning. The system was implemented on a file consisting of some 55,000 statements. The work by Green and Raphael (1968a, 1968b) resulted in a system termed the question-answering system (QA-3.5). It was an outgrowth of Raphael's thesis on semantic information retrieval (SIR) (Raphael 1968) that performed deduction. QA-3.5 included a natural language component. Another deductive system, relational store structure (RSS), started in 1966 was developed by Marrill (Computer Corporation 1967). The system had 12 deductive rules built into the program and was able to incorporate other deductive rules. The association store processor (ASP), developed by Savitt, Love, and Troop (1967), also performed deduction over binary relational data. The inference rules, specified as relational statements, were handled by breadth-first, followed by depth-first, search. These efforts, as well as those cited in Minker and Sable (1970), were important precursors to DDBs. In table 1, adapted from Minker and Sable (1970), I list some capabilities of systems developed during this stage.

Second Stage: 1970 to 1980 Whereas the first stage could be characterized as using ad hoc techniques for deduction (except for the work by Green and Raphael), the second-stage systems were based on the Robinson resolution principle, as first recognized by Green and Raphael. The SYNTEX system built by Nicolas and Syre (1974) used logic as the basis for deduction. The work by Chang (1978) on the DEDUCE 2 system, Kellogg, Klahr, and Travis (1978) on the Deductively Augmented Data Management (DADM) system, and Minker (1978) on the Maryland Refutation Proof Procedure System (MRPPS 3.0) represent work during the second stage of development of DDBs. These papers appear in Gallaire and Minker (1978). Table 2 provides a brief summary of some of the features of these systems.

DADM precomputed unifications among premises so they did not have to be recomputed during deduction. Variables were typed. Inference plans and database-access strategies were created from a premise file without requiring access to database values.

MRPPS 3.0 performed top-down searches for large databases. It permitted arguments of predicates to contain function symbols and had a knowledge base index to access the data. The deductive system used a typed unification algorithm and a semantic network. The SQO method described in McSkimin and Minker (1977) was incorporated into the system. Answer extraction, natural language processing, and voice output were part of the system.

The DEDUCE 2 system performed deduction over databases. Nonrecursive Horn rules were used and were compiled in terms of base relations. Integrity constraints were used to perform SQO on queries. Problems with respect to recursive rules and termination were also discussed (Chang 1981).

Third Stage: 1980 to Present A large number of prototype DDBs were developed, and most are described in Ramakrishnan and Ullman (1995). I briefly discuss several major efforts: work at the European Computer Research Consortium (ECRC) led by Nicolas, at the University of Wisconsin led by Ramakrishnan, and at Stanford led by Ullman. They attempted to develop operational and possibly commercial DDBs. They contributed significantly to both the theory and the implementation of DDBs. Detailed descriptions of contributions made by these systems and others can be found in Ramakrishnan and Ullman (1995). In table 3, I list some of the capabilities of systems developed in this stage, adapted from Ramakrishnan and Ullman (1995).

Implementation efforts at ECRC on DDBs started in 1984 and led to the study of algorithms and prototypes: deductive query-evaluation methods (QSQ-SLD and others) (Vieille 1986); integrity checking (SOUNDCHECK) (Decker 1986); the DDB EKS(-V1) by Vieille and his team (1990); hypothetical reasoning and checking (Vieille, Bayer, and Kuechenhoff 1996); and aggregation through recursion (Lefebvre 1994). The EKS system used a top-down evaluation method and was released to

Name	**QA-3.5** (Green and Raphael 1968a, 1968b) (question-answering system)	**ASP** (Savitt, Love, and Troop 1967) (association storing processor)	**RDF** (Levien and Maron 1965) (relational data file)	**RSS** (Computer Corporation 1967) (relational structures system)
Organization	Stanford Research Institute	Hughes Aircraft Corp.	RAND Corp.	Computer Corporation of America
Designers	Raphael, Green, and Coles	Savitt, Love, and Troop	Levien and Maron	Marill
Computer	PDP 10	IBM 360/65	IBM 7044 360/65	IBM 360/75
Programming language	Lisp 1.5	Assembly language	Assembly language	Assembly language
Input language model	Near–natural language model based on simple transformations and context-free grammar	Stylized input form and a procedural language	Stylized input forms analyzed by FOREMAN language	Near–natural language model based on matching sentence templates
Syntactic analysis technique	Earley algorithm for context-free grammar	N/A	N/A	Match of sentence against stored templates
Semantic analysis technique	Semantics stack built during syntax analysis phase	N/A	N/A	Pattern $\Rightarrow$ action operation invoked as a result of template match
Intermediate language	First-order predicate calculus	Binary relations	Binary relations	n-ary relations ($n \leq 7$ as implemented)
Data structures	Lisp-chained list structures	Relational statement elements randomized (coded) and replicated statements stored under each element	Files quadruplicated and ordered by statement number and three elements	Statement elements are hash coded and open statements linked to corresponding closed statements
Inference procedures	Formal theorem proving by Robinson resolution procedure	Inference rules specified as relational statements handled by breadth first followed by depth	Plausible inference rules specified in a procedural language called INFEREX	Twelve general rules of deductive logic used
Output language	Near–natural language generated in a synthesis phase	Relational statements	Relational statements	Near–natural language generated from n-ary relational statements

Table 1. First-Stage Deductive Database Implementations (adapted from Minker and Sable [1970]).

ECRC shareholder companies in 1990.

Implementation efforts at MCC Corporation on a DDB started in 1984 and emphasized bottom-up evaluation methods (Tsur and Zaniolo 1986) and query evaluation using such methods as seminaive evaluation, magic sets and counting (Beeri and Ramakrishnan 1991; Bancilhon et al. 1986; Sacca and Zaniolo 1986), semantics for stratified negation and set grouping (Beeri et al. 1991), investigation of safety, the finiteness of answer sets, and join-order optimization. The LDL system was implemented in 1988 and released in the period 1989 to 1991. It was among the first widely available DDBs and was distributed to universities and shareholder companies of MCC.

Name	**MRPPS 3.0** (Minker 1978) (Maryland refutation proof procedure system)	**DADM** (Kellogg, Klahr, and Travis 1978) (Deductively augmented data management)	**DEDUCE 2** (Chang 1978)
Organization	University of Maryland	System Development Corp.	IBM San Jose
Designers	Minker, McSkimin, Wilson, and Aronson	Kellogg, Klahr, and Travis	Chang
Computer	UNIVAC 1108	N/A	N/A
Programming language	SIMPL	N/A	N/A
Input language model	Multisorted, well-formed formulas	Primitive conditional statements and natural language	DEDUCE (Chang 1976) (based on symbolic logic)
Intermediate language	Clausal form	Primitive conditional statements	DEDUCE
Data structures	Semantic networks, knowledge base index	Predicate array, premise array, semantic network, predicate connection graph (Sickel 1976; Kowalski 1975)	Connection graph
Inference procedures	SL resolution (Kowalski and Kuehner 1971) and Lush resolution (Hill 1974)	Connection graph	Connection graph
Output language	Natural language voice and English (Powell 1977)	Primitive condition statements	DEDUCE
Features	Semantic query optimization, multisorted variables, no recursion, non-Horn clauses, clauses not necessarily function free, relations not in first normal form	Semantic query optimization, multisorted variables, no recursion	Semantic query optimization

Table 2. Second-Stage Deductive Database Implementations.

SL = Linear resolution with Selection function.
LUSH = Linear resolution with Unrestricted Selection function for Horn clauses.

Implementation efforts at the University of Wisconsin on the CORAL DDBs started in the 1980s. Bottom-up and magic set methods were implemented. The system, written in C and C++, is extensible and provides aggregation for modularly stratified databases. CORAL supports a declarative language and an interface to C++ that allows for a combination of declarative and imperative programming. The declarative query language supports general Horn clauses augmented with complex terms, set grouping, aggregation, negation, and relations with tuples that contain universally quantified variables. CORAL supports many evaluation strategies and automatically chooses an efficient evaluation strategy. Users can guide query optimization by selecting from among alternative control choices. CORAL provides imperative constructs such as update, insert, and delete rules. Disk-resident data are supported using the EXODUS storage manager, which also provides transaction management in a client-server environment.

Implementation at Stanford started in 1985 on NAIL! (Not Another Implementation of Logic!). The effort led to the first paper on recursion using the magic set method (Bancilhon et al. 1986). Other contributions were aggregation in logical rules and theoretical contributions to negation: stratified negation by Van Gelder (1988); well-founded negation by Van Gelder, Ross, and Schlipf (1991); and

Name	Developed	Recursion	Negation	Aggregation	Update	Integrity constraints	Optimization	Storage	Interfaces
ADITI (Vaghani et al. 1991)	U. Melbourne	General	Stratified	Stratified	No	No	Magic sets, seminaive	Extensional database, intensional database	Prolog
COL (Abiteboul and Grumback 1991)	INRIA	?	Stratified	Stratified	No	No	None	Main memory	Machine learning
CONCEPT BASE (Jeusfeld and Staudt 1993)	U. Aachen	General	Locally Stratified	No	Yes	No	Magic sets, seminaive	EDB only	C, Prolog
CORAL (Ramakrishnan, Srivastava, and Sudarshan 1992)	U. Wisconsin	General	Modular Stratified	Modular Stratified	No	No	Magic sets, seminaive, context factoring, projection pushing	Extensional database, intensional database	C, C++, extensible
EKS-V1 (Vieille et al. 1992)	ECRC	General	Stratified	General	Yes	Yes	Query-subquery, left-right linear	Extensional database, intensional database	Persistent Prolog
DECLARE (Kiesslinger and Schmidt 1994)	MAD Intelligent Systems	General	Locally Stratified	General	No	No	Magic sets, seminaive, projection pushing	EDB only	C, Common Lisp
LDL, LDL++, SALAD (Chimenti et al. 1990)	MCC	General	Stratified	Stratified	Yes	No	Magic sets, seminaive, left-right linear, projection pushing		
Extensional database only	C, C++, SQL								
LOGRES (Cacace et al. 1990)	Polytech. of Milan	Linear	Inflationary semantics	Stratified	Yes	Yes	Seminaive, algebraic X forms	Extensional database, intensional database	INFORMIX
NAIL-GLUE (Morishita, Derr, and Phipps 1993)	Stanford U.	General	Well founded	Glue only	Glue only	No	Magic sets, seminaive, right linear	EDB only	None
STARBURST (Mumick et al. 1990)	IBM Almaden	General	Stratified	Stratified	No	No	Magic sets, seminaive (variant)	Extensional database, intensional database	Extensible

Table 3. Existing Implementations of Deductive Databases (adapted from Ramakrishnan and Ullman [1995]*).*

modularly stratified negation by Ross (1990). A language called GLUE (Morishita, Derr, and Phipps 1993), developed for logical rules, has the power of SQL statements as well as a conventional language for the construction of loops, procedures, and modules.

Implementations of DDBs in the first, second, and third stages of their development have demonstrated the feasibility and practicality of the technology. Tools and techniques have been developed to produce efficient DDBs.

Prospects for Commercial Implementation of Disjunctive Databases[1]

... we should not abandon all research on theories of negation and alternative semantics, but we must take stock of what we have accomplished and make it more accessible for users.

One might address why after 20 years of theoretical research in DDBs, no commercial systems exist. To place this statement in perspective, it is well to recall that approximately 12 years passed before relational systems were available commercially. As Ullman has stated on a number of occasions, DDB theory is more subtle than relational database theory. However, many prototypes have been developed starting from the 1960s, as described in the previous subsection. However, none of the systems in Ramakrishnan and Ullman (1995) are likely to become commercial products, with, possibly, two exceptions: (1) ADITI (Ramamohanarao 1993) and (2) VALIDITY (Friesen et al. 1995; Ling, Mendelzon, and Vieille 1995), developed at the Bull Corporation. According to a personal communication with Ramamohanarao, leader of the ADITI effort, that I had in May 1996, they are perhaps one year from having a commercial product. In a communication that I received from him on 20 February 1997, he stated: "We have now completed most of the difficult parts of the low-level implementation of ADITI. I am very hopeful of getting the beta release of the system by December 1997. The task was much harder and time consuming than I have ever anticipated."

Whether it becomes a product remains to be seen. I believe it will depend on moving the system from a university setting to industry. Implementers and application specialists, rather than university researchers, are required.

At the Bull Corporation, Nicolas and Vieille have headed an effort to develop the VALIDITY DDB system that integrates object-oriented features. In Minker (1996), I reported that the VALIDITY DDB effort had been ongoing for about four years. It appeared to be entering the marketplace and was being moved from the Bull Corporation to a new company that will be responsible for its maintenance, marketing, and improvements. In a personal communication with Nicolas on 7 March 1997, he stated:

> VALIDITY is now being further developed and marketed by Next Century Media, Inc., a California corporation in which Groupe Bull has some equity interests. Its principal office [is] in the San Francisco area.
>
> The VALIDITY DOOD software platform is currently mainly used to develop NCM's products in electronic media for interactive media applications. Two of these products enable marketers to target their advertising messages to household clusters, to individual households, and to specific consumers, based on the user's expressed and implied interests and preferences, and to convert the data coming from the user into a database of ongoing and useful information about these customers. A third product enables marketers to measure the effectiveness of their media plan and expenditures in a timely manner, based on a full census of the entire audience, rather than on samples which are fraught with inherent biases and errors.

No commercial systems exist for several reasons. First, most prototypes were developed at universities. Without commercial backing for the venture, universities are not positioned to either develop or support maintenance required for large system developments. Systems developed in research organizations controlled by consortia (ECRC and MCC) have not had full backing of consortia members. Second, implementation efforts to develop a commercial product were vastly underestimated. A large investment must be made to develop a DDB that both competes and extends relational technology. According to industry standards, an investment on the order of $30 to $50 million is required to develop and place a database system in the market, no matter what technology it relies on. Furthermore, researchers tend to change their interests rather than consolidate their work and invest in technology transfer toward industry. Third, until recently, no convincing demonstration has been made of a large commercial problem that requires a DDB, which might be why the MCC and ECRC developments were terminated. However, now, a large number of applications could take advantage of this technology, as evidenced by the book by Ramakrishnan (1995) and the applications being performed by the VALIDITY deductive object-oriented database (DOOD) system. In addition, Levy et al. (1995) studied the problem of computing answers to queries using materialized views and note that this work is related to applications such as

global information systems, mobile computing, view adaptation, and the maintenance of physical data independence. Levy et al. (1996) describe how DDBs can be used to provide uniform access to a heterogeneous collection of more than 100 information sources on the World Wide Web. Fourth, apparently no university researchers have tried to obtain venture capital to build a product outside the university. Efforts by some from MCC to obtain venture capital did not succeed.

Does lack of a commercial system at this date forebode the end of logic and databases? I believe that such a view is naive. First, there still is a strong prospect, as noted previously, of commercial DDBs. Second, considering that it took 12 years before relational database technology entered the marketplace, there is no need for alarm. Third, as the following developments portend, relational databases are starting to incorporate techniques stemming from research in DDBs.

Indeed, many of the techniques introduced within DDBs are finding their way into relational technology. The new SQL standards for relational databases are beginning to adopt many of the powerful features of DDBs. In the SQL-2 standards (also known as SQL-92) (Melton and Simon 1993), a general class of integrity constraints called *asserts* allow for arbitrary relationships between tables and views to be declared. These constraints exist as separate statements in the database and are not attached to a particular table or view. This extension is powerful enough to express the types of integrity constraint generally associated with DDBs. However, only the full SQL-2 standard includes assert specifications. The intermediate SQL-2 standard, the basis for most current commercial implementations, does not include asserts. The relational language for the next-generation SQL, SQL3, currently provides an operation called *recursive union* that supports recursive processing of tables (Melton 1996). As noted in Melton (1996): "The use of the recursive union operator allows both linear (single-parent, or tree) recursion and nonlinear (multiparent, or general directed graph) recursion. This solution will allow easy solutions to bill-of-material and similar applications."

Linear recursion is currently a part of the client server of IBM's DB2 system. IBM is using the magic set method to perform linear recursion. Also, indications are that the ORACLE database system will support some form of recursion.

A further development is that SQO is beginning to be incorporated into relational databases. In DB2, cases are recognized when only one answer is to be found, and the search is terminated. In other systems, equalities and other arithmetic constraints are being added to optimize search. I believe it will not be long before join elimination is introduced into relational technology. One can now estimate when it will be useful to eliminate a join (Godfrey, Gryz, and Minker 1996). The tools and techniques already exist, and it is merely a matter of time before users and system implementers have them as part of their database systems.

Another technology available for commercial use is cooperative databases. The tools and techniques exist, as evidenced by COBASE (Chu, Chen, and Merzbacher 1994) and CARMIN (Gaasterland et al. 1992). With the introduction of recursion and SQO techniques into relational database technology, it will be necessary to provide users with cooperative responses so they understand why certain queries fail or succeed. It will also permit queries to be relaxed when the original query fails, permitting reasonable, if not logically correct, answers to be provided to users. Because user constraints can be handled in the same way that integrity constraints are handled, we will see relational systems that incorporate the needs of individual users into a query, as represented by their constraints.

Two significant developments have taken place in the implementation of commercial DDBs. First is the incorporation of techniques developed in DDDBs into relational technology. Recursive views that use the magic set technique for implementation are being permitted, and methods developed for SQO are being applied. Second is the development of a DOOD, VALIDITY, that is in commercial use as well as the development of the ADITI DDB that is scheduled to undergo beta testing in December 1997. It remains to be seen how long one can make patches to relational technology to simulate the capabilities of DDB systems.

Emerging Areas and Trends

In the previous sections, we discussed many theories and semantics for negation in both extended DDBs and DDBs. We understand a great deal about negation, except for how and when to use a given theory, which will be an area of confusion when users want to apply the work. Much more work is needed if the areas of implementation and application are to catch up with the intellectual developments achieved over the past 20 years. The field is saturated with alternative theories of semantics, and work is needed on more fertile topics. Unless we do so, funding for logic and databases will wane, as I believe it has in the United

States. However, we should not abandon all research on theories of negation and alternative semantics, but we must take stock of what we have accomplished and make it more accessible for users.

The role of logic will be of increasing importance because of the need to handle highly complex data (partly as a result of the advances in networking technology and the reduction in the cost of both processing time and primary, secondary, and tertiary memory). These data will require more complex models of data access and representation. Advances will require formal models of logic rather than ad hoc solutions. Below, I briefly mention some fertile areas for further exploration. This listing of important areas to investigate is not intended to be exhaustive.

The role of logic will be of increasing importance because of the need to handle highly complex data.

Temporal databases, which deal with time, are important for historical, real-time databases and other aspects of databases. Work in this area has been done by Snodgrass (1987), Chomicki (1995), and Sistla and Wolfson (1995). A paper on applying transition rules to such databases for integrity constraint checking appears in Martin and Sistac (1996).

Transactions and updates need further attention. Semantic models of updates exist (Fernández, Grant, and Minker 1996) that assure views and data are updated correctly. Transactions that require sequences of updates, long-duration transaction models, and work-flow management are areas that require work. In emerging applications of database systems, *transactions* are viewed as sequences of nested, and most probably interactive, subtransactions that can sparsely occur over long periods of time. In this scenario, new complex transaction systems must be designed. Logic-based transaction systems will be essential to assure that an appropriate and correct transaction is achieved. See Minker (1996) for references.

Active databases consist of data that protect their own integrity and describe the database semantics. They are represented by the formalism event-condition-action (ECA) (Xerox Technologies 1989) and denote that whenever an event *E* occurs, if condition *C* holds, then trigger action *A*. It has a behavior of its own beyond passively accepting statements from users or applications. On recognition of certain events, it invokes commands and monitors and manages itself. It can invoke external actions that interact with systems outside the database and can activate a potentially infinite set of triggers. Although declarative constraints are provided, the ECA formalism is essentially procedural in nature. Zaniolo (1995) noted the need for declarative semantics of triggers. He has developed a unified semantics for active DDBs and has shown how active database rules relate to transaction-conscious stable model semantics. Baral and Lobo (1996) proposed a first step toward characterizing active databases. A clear semantics, sound implementations, and a better understanding of complexity issues are required in active databases. Work in the situation calculus and datalog extensions apply here.

Data mining and inductive inference deal with finding generalizations extracted from a database or a logic program. Generalizations can be integrity constraints that must be true with respect to the database or generalizations that might be true of the current state but might change if there are updates. Database administrators will need to determine which generalizations are integrity constraints and which apply only to the current state. SQO can handle either case and inform the user which constraint might be applicable to a query. As demonstrated in Muggleton and De Raedt (1994), logic programming can be used to form inductive inferences, and *Knowledge Discovery in Databases* (Piatetsky-Shapiro and Frawley 1991) covers work on knowledge data mining. Laurent and Vrain (1996) discuss how to couple DDBs and inductive logic programming to learn query rules for optimizing databases with update rules.

Integrating production systems with DDBs is needed to develop a formal approach to integrate and develop the semantics of rule-based systems. See Minker (1996) for references.

Logical foundations of DOODs is needed. No formal definition exists that covers all aspects of object-oriented databases. Efforts have been undertaken by Kifer and his coworkers (Kifer, Lausen, and Wu 1995) to develop a formal foundation for object-oriented databases. Work is required to develop techniques, a formal theory of updating, and all tools and techniques for DDBs.

Description logics restrict knowledge representation so that deduction is tractable but sufficiently powerful to represent knowledge naturally. See Minker (1996) for references to systems that incorporate description logics. In DDBs, representational power is also limited to allow for more tractable deduction. Some of these limits are a restriction to Horn clauses, no logical terms, no existential quantification, and so forth. Research in DDBs has sought to extend the representational power yet preserve tractability to the greatest extent possible: For example, DDDBs allow for general clauses, and the addition of null values allows for a type of

existential quantification. DDBs and description logics have remained distinct, but their goals are similar.

Heterogeneous databases integrate multiple databases into one system that do not necessarily share the same data models. There is the need for a common logic-based language for mediation and a formal semantics of such databases. Work on HERMES by Subrahmanian et al. (1994), on TSIMMIS by Chawathe et al. (1994), and by Ramakrishnan (Miller, Ioannidis, and Ramakrishnan 1994) and his colleagues illustrate the efforts in this area. Heterogeneous databases are also needed to handle textual data. Kero and Tsur (1996) describe the $\mathfrak{IQ}$ system that uses a DDB $\mathfrak{LDL}$++ to reason about textual information. Language extensions for the semantic integration of DDBs is proposed by Asirelli, Renso, and Turini (1996). The language allows mediators to be constructed, using a set of operators for composing programs and message-passing features.

Multimedia databases (Subrahmanian and Jajodia 1995) is an emerging area for which new data models are needed. These databases have special problems, such as the manipulation of geographic databases; picture retrieval where a concept orthogonal to time can appear in the database: space; and video databases, where space and time are combined. Temporal and spatial reasoning are needed. Logic will play a major role in the development of query languages for these new data models and will permit deductive reasoning, and a formal semantics will provide a firm theoretical basis for them.

Combining databases relates both to heterogeneous and multimedia systems. Here, one is trying to combine databases that share the same integrity constraints and schema. Such databases arise in distributed system work and the combining of knowledge bases. In addition to handling problems that arise because the combined databases might be inconsistent, one has to handle priorities that can exist among individual facts. A formal treatment and references appear in Pradhan and Minker (1995).

Integrity constraints, SQO, and constraint logic programming (CLP) are related topics. SQO uses constraints in the form of integrity constraints to prune the search space. These integrity constraints introduce equalities, inequalities, and relations into a query to help optimize search (Chakravarthy, Grant, and Minker 1990). CLP introduces domain constraints. These constraints might be equalities or inequalities and might even be relations (Jaffar and Maher 1994). Constraint databases and constraint-intensive queries are required in many advanced applications. Constraints can capture spatial and temporal behavior that is not possible in existing databases. Relationships between these areas need to be explored further and applied to DDBs. Spatial databases defined in terms of polynomial inequalities are investigated by Kuipers et al. (1996), who consider termination properties of datalog programs.

Abductive reasoning is the process of finding explanations for observations in a given theory. Given a set of sentences T (a theory) and a sentence G (an observation), the abductive task can be characterized as the problem of finding a set of sentences (abductive explanation for G) such that $T \cup \Delta \models G$, $T \cup \Delta$ is consistent, and Δ is minimal with respect to set inclusion (Kakas, Kowalski, and Toni 1993).

A comprehensive survey and critical overview of the extension of logic programming to the performance of abductive reasoning (abductive logic programming) is given in Kakas, Kowalski, and Toni (1993). They outline the framework of abduction and its applications to default reasoning and introduce an augmentation-theoretic approach to the use of abduction as an interpretation for negation as failure. They show that abduction has strong links to extended disjunctive logic programming. Abduction is shown to generalize negation as failure to include not only negative but also positive hypotheses and to include general integrity constraints. They show that abductive logic programming is related to the justification-based truth maintenance system of Doyle (1979). Inoue and Sakama (1996) developed a fixpoint semantics for abductive logic programs in which the belief models are characterized as the fixpoint of a disjunctive program obtained by a suitable program transformation. For a summary of complexity results on abductive reasoning and nonmonotonic reasoning, see Cadoli and Schaerf (1993).

Abductive reasoning is the process of finding explanations for observations in a given theory.

High-level robotics is an area of active research in which logic plays a significant role. Knowledge bases are used to solve problems in cognition required to plan actions for robots and deal with multiple agents in complicated environments. Work in deductive and disjunctive databases relates to this problem. In some instances, a robot can have several options that can be represented by disjunctions. Additional information derived from alternative information sources such as sensors can serve to disambiguate the possibilities. Universities engaged in this research are the University of Toronto (Lesperance et al. 1994), the University of Texas at El Paso (Baral, Gelfond, and

Provetti 1996), the University of Texas at Austin (Gelfond and Lifschitz 1993), and the University of Linkoping (Sandewall 1994). Kowalski and Sadri (1996) discuss a unified agent architecture that combines rationality with reactivity that relates to this topic.

Applications of DDB techniques will be important. Greater emphasis is required to apply DDB technology to realistic problems. DDBs have been shown to be important for both relational and deductive systems on such topics as SQO (Chakravarthy, Grant, and Minker 1990), cooperative answering (Gaasterland, Godfrey, and Minker 1992), and global information systems and mobile computing (Levy et al. 1995).

Commercial implementations of DDBs and DOODs are needed. The deductive model of databases is beginning to take hold, evidenced by the textbook by Abiteboul, Hull, and Vianu (1995). The prospect that the ADITI (Ramamohanarao 1993) system might be available in a year and the applications for which the DDB VALIDITY are being used indicate that progress is being made in developing a commercial DDB. The merger of object-oriented and deductive formalisms is taking place, as illustrated by the proceedings of the DOOD conference series (Ling, Mendelzon, and Vieille 1995; Ceri, Tanaka, and Tsur 1993; Delobel, Kifer, and Masunaga 1991; Kim, Nicolas, and Nishio 1989). That the VALIDITY (Friesen, Lefebvre, and Vielle 1996) system is currently in use by customers indicates that the object-oriented and deductive formalisms are soon to be available commercially. Additional features will be required for commercial systems such as cooperative answering (Gaasterland, Godfrey, and Minker 1992a) and arithmetic and aggregate operators (Dobbie and Topor 1996).

It is clear from this discussion that logic and databases can contribute significantly to a large number of exciting new topics. Hence, the field of logic and databases will continue to be a productive area of research and implementation.

Logic and databases have helped the field of databases be a scientific endeavor rather than an ad hoc collection of techniques.

We understand what constitutes a database, a query, and an answer to a query and where knowledge has its place.

Summary

I discussed major accomplishments that have taken place in logic and databases during the 20 years since 1976. Among these accomplishments are the extension of relational databases, the development of the semantics and complexity of these alternative databases, the ability to permit knowledge base systems to be represented and developed, and the use of logic programming and DDBs to implement nonmonotonic reasoning systems. I discussed many new areas that will be important in the near- and long-term future. It is clear that the field of logic and databases had a significant prehistory before 1970 and a well-defined area of research, complete with past achievements and continued future areas of work.

In the past 20 years, we have seen logic and databases progress from a fledgling field to a fully developed, mature field. The new areas that I cited that need further investigation show that we have not nearly exhausted the work in this field. I envision that many more workshops will be held on this topic. Logic and databases have helped the field of databases be a scientific endeavor rather than an ad hoc collection of techniques. We understand what constitutes a database, a query, and an answer to a query and where knowledge has its place. I look forward to the next 20 years of this field. I hope that I will have an opportunity, then, to look back and see a field that has accomplished much and is still vibrant. To remain vibrant, we will have to take on some of the new challenges rather than be mired in the semantics of more exotic databases. We will have to address implementation issues, and we will have to be able to provide guidance to practitioners who will need to use the significant developments in logic and databases.

Acknowledgments

This article is a condensation, and in some cases an expansion, of an invited keynote address presented at the Workshop on Logic in Databases in San Miniato, Italy, in 1996. Those interested in the longer version of the article should refer to Minker (1996). A number of my colleagues contributed their thoughts on what they considered to be the significant developments in the field, including Robert Demolombe, Hervé Gallaire, Georg Gottlob, John Grant, Larry Henschen, Bob Kowalski, Jean-Marie Nicolas, Raghu Ramakrishnan, Kotagiri Ramamohanarao, Ray Reiter, and Carlo Zaniolo. Many of my former and current students also contributed thoughts, including Sergio Alvarez, Chitta Baral, Jose Alberto Fernández, Terry Gaasterland, Parke Godfrey, Jarek Gryz, Jorge Lobo, Sean Luke, and Carolina Ruiz. Although many of the views reflected in the article might be shared by those who made suggestions, I take full responsibility for them. The full paper is dedicated in honor of Hervé Gallaire and Jean-Marie Nicolas with whom I have worked as co-author, co-editor, colleague, and friend and who have helped to make the field of deductive databases a reality. Support for this article was received from the National Science Foundation under grant IRI 9300691.

Note

1. This subsection reflects comments made at the Workshop on Logic in Databases in San Miniato, Italy, 1 to 2 July, 1996, in the panel session, Deductive Databases: Challenges, Opportunities, and Future Directions, by Arno Siebes, Shalom Tsur, Jeff Ullman, Laurent Vieille, and Carlo Zaniolo, and in a personal communication by Jean-Marie Nicolas. I am wholly responsible for the views expressed in this subsection.

References

Abiteboul, S., and Grumback, S. 1991. A Rule-Based Language with Functions and Sets. *ACM Transactions on Database Systems* 16(1): 1–30.

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases.* Reading, Mass.: Addison-Wesley.

Apt, K., and Bol, R. 1994. Logic Programming and Negation: A Survey. *Journal of Logic Programming* 19–20: 9–71.

Apt, K.; Blair, H.; and Walker, A. 1988. Towards a Theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming,* ed. J. Minker, 89–148. San Francisco, Calif.: Morgan Kaufmann.

Asirelli, P.; Renso, C.; and Turini, F. 1996. Language Extensions for Semantic Integration of Deductive Databases. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 415–434. New York: Springer-Verlag.

Bancilhon, F.; Maier, D.; Sagiv, Y.; and Ullman, J. 1986. Magic Sets and Other Strange Ways to Implement Logic Programs. In Proceedings of the ACM Symposium on Principles of Database Systems, 1–15. New York: Association of Computing Machinery.

Baral, C., and Gelfond, M. 1994. Logic Programming and Knowledge Representation. *Journal of Logic Programming* 19–20: 73–148.

Baral, C., and Lobo, J. 1996. Formal Characterization of Active Databases. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 195–215. New York: Springer-Verlag.

Baral, C.; Gelfond, M.; and Provetti, A. 1997. Representing Actions: Laws, Observations, and Hypothesis. *Journal of Logic Programming* 31(1–3): 201–243.

Baral, C.; Lobo, J.; and Minker, J. 1990a. Generalized Disjunctive Well-Founded Semantics for Logic Programs: Declarative Semantics. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems,* 465–473. New York: North Holland.

Baral, C.; Lobo, J.; and Minker, J. 1990b. Generalized Disjunctive Well-Founded Semantics for Logic Programs: Procedural Semantics. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent Systems,* 456–464. New York: North Holland.

Beeri, C., and Ramakrishnan, R. 1991. On the Power of Magic. *Journal of Logic Programming* 10(3–4): 255–300.

Beeri, C.; Naqvi, S.; Shmueli, O.; and Tsur, S. 1991. Set Constructors in a Logic Database Language. *Journal of Logic Programming* 10(3–4): 181–253.

Bell, C.; Nerode, A.; Ng, R.; and Subrahmanian, V. 1993. Implementing Stable Model Semantics by Linear Programming. In Proceedings of the 1993 International Workshop on Logic Programming and Non-Monotonic Reasoning, June, Lisbon, Portugal.

Ben-Eliyahu, R., and Dichter, R. 1994. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence* 12:53–87.

Ben-Eliyahu, R., and Palopoli, L. 1994. Reasoning with Minimal Models: Efficient Algorithms and Applications. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning,* 39–50. San Francisco, Calif.: Morgan Kaufmann.

Ben-Eliyahu, R.; Palopoli, L.; and Zemlyanker, V. 1996. The Expressive Power of Tractable Disjunction. Paper presented at the Twelfth European Conference on Artificial Intelligence (ECAI96), 12–16 August, Budapest, Hungary.

Blaustein, B. 1981. Enforcing Database Assertions: Techniques and Applications. Ph.D. thesis, Computer Science Department, Harvard University.

Brass, S., and Dix, J. 1995. A General Approach to Bottom-Up Computation of Disjunctive Semantics. In *Nonmonotonic Extensions of Logic Programming,* eds. J. Dix, L. Pereira, and T. Przymusinski, 127–155. Lecture Notes in Computer Science 927. New York: Springer-Verlag.

Bry, F. 1990. Query Evaluation in Recursive Databases: Bottom-Up and Top-Down Reconciled. *Data and Knowledge Engineering* 5:289–312.

Cacace, F.; Ceri, S.; Crespi-Reghizzi, S.; and Zicari, R. 1990. Integrating Object-Oriented Data Modeling with a Rule-Based Programming Paradigm. In Proceedings of the ACM SIGMOD Conference on Management of Data, 225–236. New York: Association of Computing Machinery.

Cadoli, M. 1996. Panel on "Knowledge Compilation and Approximation": Terminology, Questions, References. In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/Math-96), 183–186. Available from Martin Columbic, Bar-Ilan University, Department of Computer Science, Tel Aviv, Israel.

Cadoli, M. 1993. Semantical and Computational Aspects of Horn Approximations. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 39–44. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Cadoli, M., and Lenzerini, M. 1994. The Complexity of Closed-World Reasoning and Circumscription. *Journal of Computer and Systems Science* 43:165–211.

Cadoli, M., and Schaerf, M. 1993. A Survey of Complexity Results for Non-Monotonic Logics. *Journal of Logic Programming* 13:127–160.

Ceri, S.; Tanaka, K.; and Tsur, S., eds. 1993. *Proceedings of the Third International Conference on Deductive and Object-Oriented Databases—DOOD'93.* Heidelberg: Springer-Verlag.

Chakravarthy, U. S.; Grant, J.; and Minker, J. 1990. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems* 15(2): 162–207.

Chang, C. 1981. On Evaluation of Queries Containing Derived Relations. In *Advances in Database Theory, Volume 1*, ed. H. G. J. M. J.-M. Nicolas, 235–260. New York: Plenum.

Chang, C. 1978. DEDUCE 2: Further Investigations of Deduction in Relational Databases. In *Logic and Databases*, ed. H. G. J. Minker, 201–236. New York: Plenum.

Chang, C. 1976. DEDUCE—A Deductive Query Language for Relational Databases. In *Pattern Recognition and Artificial Intelligence*, ed. C. Chen, 108–134. San Diego, Calif.: Academic.

Chawathe, S.; Garcia-Molina, H.; Hammer, J.; Ireland, K.; Papakonstantinou, Y.; Ullman, J.; and Widom, J. 1994. The TSIMMIS Project: Integration of Heterogeneous Information Sources. Paper presented at IPSJ Conference, October, Tokyo, Japan.

Chen, W., and Warren, D. 1993. A Goal-Oriented Approach to Computing Well-Founded Semantics. *Journal of Logic Programming* 17(2–4): 279–300.

Chimenti, D.; Gamboa, R.; Krishnamurthy, R.; Naqvi, S.; Tsur, S.; and Zaniolo, C. 1990. The LDL System Prototype. *IEEE Transactions on Knowledge and Data Engineering* 2(1): 76–90.

Chomicki, J. 1995. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems* 20(2): 111–148.

Chu, W. W.; Chen, Q.; and Merzbacher, M. A. 1994. COBASE: A Cooperative Database System. In *Nonstandard Queries and Nonstandard Answers, Studies in Logic and Computation 3*, eds. R. Demolombe and T. Imielinski, 41–73. Oxford, U.K.: Clarendon.

Clark, K. L. 1978. Negation as Failure. In *Logic and Data Bases*, eds. H. Gallaire and J. Minker, 293–322. New York: Plenum.

Codd, E. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(6): 377–387.

Colmerauer, A.; Kanoui, H.; Pasero, R.; and Roussel, P. 1973. Un Systeme de Communication Homme-Machine en Francais (A System of Man-Machine Communication in French). Technical Report, Groupe de Intelligence Artificielle Universitae de Aix-Marseille II, Marseille.

Computer Corporation. 1967. Relational Structures Research. Technical Report, Computer Corporation of America, Cambridge, Massachusetts.

Date, C. 1995. A*n Introduction to Database Systems*, 6th ed. Reading, Mass.: Addison-Wesley.

Decker, H. 1991. On the Declarative, Operational, and Procedural Semantics of Disjunctive Computational Theories. In Proceedings of the Second International Workshop on the Deductive Approach to Information Systems and Databases, 9–11 September, Aiguablava, Spain.

Decker, H. 1986. Integrity Enforcement on Deductive Databases. In *Proceedings of the First International Conference on Expert Database Systems*, 381–395. Menlo Park, Calif.: Benjamin Cummings.

Delobel, C.; Kifer, M.; and Masunaga, Y., eds. 1991. *Proceedings of the Second International Conference on Deductive and Object-Oriented Databases (DOOD'91).* Heidelberg: Springer-Verlag.

del Val, A. 1995. An Analysis of Approximate Knowledge Compilation. In Proceedings of the Fourteenth International Conference on Artificial Intelligence. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Demolombe, R., and Jones, A. J. I. 1996. Integrity Constraints Revisited. *Journal of the IGPL (Interest Group in Pure and Applied Logics): An Electronic Journal on Pure and Applied Logic* 4(3): 369–383.

Dix, J. 1992. A Framework for Representing and Characterizing Semantics of Logic Programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, eds. B. Nebel, C. Rich, and W. Swartout, 591–602. San Francisco, Calif.: Morgan Kaufmann.

Dobbie, G., and Topor, R. 1996. Arithmetic and Aggregate Operators in Deductive Object-Oriented Databases. In Logic in Databases (LID'96), eds. D. Pedreschi and C. Zaniolo, 399–407. New York: Springer-Verlag.

Doyle, J. 1979. Truth Maintenance System. *Artificial Intelligence* 12(3): 231–272.

Eiter, T., and Gottlob, G. 1995. On the Computation Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence* 15(3–4): 289–323.

Fernández, J., and Minker, J. 1995. Bottom-Up Computation of Perfect Models for Disjunctive Theories. *Journal of Logic Programming* 25(1): 33–51.

Fernández, J., and Minker, J. 1992. Semantics of Disjunctive Deductive Databases. In *Proceedings of the International Conference on Database Theory*, 332–356. Berlin: Springer-Verlag.

Fernández, J. A., and Minker, J. 1991. Bottom-Up Evaluation of Hierarchical Disjunctive Deductive Databases. In *Logic Programming Proceedings of the Eighth International Conference*, ed. K. Furukawa, 660–675. Cambridge, Mass.: MIT Press.

Fernández, J.; Grant, J.; and Minker, J. 1996. Model-Theoretic Approach to View Updates in Deductive Databases. *Journal of Automated Reasoning* 17(2): 171–197.

Fernández, J. A.; Lobo, J.; Minker, J.; and Subrahmanian, V. 1993. Disjunctive LP + Integrity Constraints = Stable Model Semantics. *Annals of Mathematics and Artificial Intelligence* 8(3–4): 449–474.

Fitting, M. 1985. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming* 2:295–312.

Friesen, O.; Lefebvre, A.; and Vieille, L. 1996. VALIDITY: Applications of a DOOD System. In *Proceedings of the Fifth International Conference on Extending Database Technology—EDBT'96*, 131–134. Lecture Notes in Computer Science 1057. New York: Springer-Verlag.

Friesen, O.; Gauthier-Villars, G.; Lefebvre, A.; and Vieille, L. 1995. Applications of Deductive Object-Oriented Databases Using DEL. In *Applications of Logic Databases,* ed. R. Ramakrishnan, 1–22. New York: Kluwer Academic.

Gaasterland, T., and Lobo, J. 1993. Processing Negation and Disjunction in Logic Programs through Integrity Constraints. *Journal of Intelligent Information Systems* 2(3): 225–243.

Gaasterland, T.; Godfrey, P.; and Minker, J. 1992a. An Overview of Cooperative Answering. *Journal of Intelligent Information Systems* 1(2): 123–157.

Gaasterland, T.; Godfrey, P.; and Minker, J. 1992b. Relaxation as a Platform for Cooperative Answering. *Journal of Intelligent Information Systems* 1:293–321.

Gaasterland, T.; Godfrey, P.; Minker, J.; and Novik, L. 1992. A Cooperative Answering System. In *Proceedings of the Logic Programming and Automated Reasoning Conference,* ed. A. Voronkov, 478–480. Lecture Notes in Artificial Intelligence 624. New York: Springer-Verlag.

Gaasterland, T.; Godfrey, P.; Minker, J.; and Novik, L. 1992. A Cooperative Answering System. In *Proceedings of the Logic Programming and Automated Reasoning Conference,* ed. A. Voronkov, 478–480. Lecture Notes in Artificial Intelligence 624. New York: Springer-Verlag.

Gallaire, H., and Minker, J., eds. 1978. *Logic and Databases.* New York: Plenum.

Gallaire, H.; Minker, J.; and Nicolas, J.-M., eds. 1984a. *Advances in Database Theory, Volume 2.* New York: Plenum.

Gallaire, H.; Minker, J.; and Nicolas, J.-M. 1984b. Logic and Databases: A Deductive Approach. *ACM Computing Surveys* 16(2): 153–185.

Gallaire, H.; Minker, J.; and Nicolas, J.-M., eds. 1981. *Advances in Database Theory, Volume 1.* New York: Plenum.

Gelfond, M., and Lifschitz, V. 1993. Representing Actions and Change by Logic Programs. *Journal of Logic Programming* 17(2–4): 301–323.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

Gelfond, M., and Lifschitz, V. 1990. Logic Programs with Classical Negation. In *Proceedings of the Seventh International Conference on Logic Programming,* eds. D. Warren and P. Szeredi, 579–597. Cambridge, Mass.: MIT Press.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming,* eds. R. Kowalski and K. Bowen, 1070–1080. Cambridge, Mass.: MIT Press.

Godfrey, P.; Gryz, J.; and Minker, J. 1996. Semantic Query Evaluation for Bottom-Up Evaluation. In Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems, June, Zakopane, Poland.

Gottlob, G. 1994. Complexity and Expressive Power of Disjunctive Logic Programming (Research Overview). In *International Logic Programming Symposium ILPS'94,* ed. M. Bruynooghe, 23–42. Cambridge, Mass.: MIT Press.

Green, C., and Raphael, B. 1968a. Research in Intelligent Question-Answering Systems. In Proceedings of the ACM Twenty-Third National Conference, 169–181. New York: Association of Computing Machinery.

Green, C., and Raphael, B. 1968b. The Use of Theorem-Proving Techniques in Question-Answering Systems. In Proceedings of the Twenty-Third ACM National Conference. New York: Association of Computing Machinery.

Hammer, M., and Zdonik, S. 1980. Knowledge-Based Query Processing. In Proceedings of the Sixth International Conference on Very Large Data Bases, 137–147. Washington, D.C.: IEEE Computer Society.

Harel, D. 1980. Review Number 36,671 of Logic and Data Bases by H. Gallaire and J. Minker. *Computing Reviews* 21(8): 367–369.

Hill, R. 1974. Lush Resolution and Its Completeness. Technical Report, DCL Memo 78, Department of Artificial Intelligence, University of Edinburgh.

Inoue, K., and Sakama, C. 1996. A Fixpoint Characterization of Abductive Logic Programs. *Journal of Logic Programming* 27(2): 107–136.

Jaffar, J., and Maher, M. 1994. Constraint Logic Programming: A Survey. *Journal of Logic Programming* 19–20: 503–581.

Jeusfeld, M., and Staudt, M. 1993. Query Optimization in Deductive Object Bases. In *Query Processing for Advanced Database Applications,* eds. G. Vossen, J. C. Feytag, and D. Maier. San Francisco, Calif.: Morgan Kaufmann.

Kakas, A. C.; Kowalski, R. A.; and Toni, F. 1993. Abductive Logic Programming. *Journal of Logic and Computation* 6(2): 719–770.

Kautz, H., and Selman, B. 1992. Forming Concepts for Fast Inference. In Proceedings of the Tenth National Conference on Artificial Intelligence, 786–793. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kellogg, C.; Klahr, P.; and Travis, L. 1978. Deductive Planning and Pathfinding for Relational Data Bases. In *Logic and Data Bases,* eds. H. Gallaire and J. Minker, 179–200. New York: Plenum.

Kero, B., and Tsur, S. 1996. The CALLIGRAPHIC IQ System: A Deductive Database Information Lens for Reasoning about Textual Information. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 377–395. New York: Springer-Verlag.

Kiessling, W., and Schmidt, H. 1994. DECLARE and SDS: Early Efforts to Commercialize Deductive Database Technology. *VLDB Journal* 3(2): 211–243.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM* 42(4): 741–843.

Kim, W.; Nicolas, J.-M.; and Nishio, S., eds. 1990. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89).* Amsterdam: North-Holland.

King, J. 1981. QUIST: A System for Semantic Query Optimization in Relational Databases. In Proceedings of the Seventh International Conference on Very Large Data Bases, 510–517. Washington, D.C.: IEEE Computer Society.

Kowalski, R. 1978. Logic for Data Description. In *Logic and Data Bases,* ed. H. G. J. Minker, 77–102. New York: Plenum.

Kowalski, R. 1974. Predicate Logic as a Programming Language. In Proceedings of the International Federation for Information Processing (World Computer Congress) 4, 569–574. Amsterdam: North-Holland.

Kowalski, R., and Sadri, F. 1996. Toward a Unified Agent Architecture That Combines Rationality with Reactivity. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 131–150. New York: Springer-Verlag.

Kowalski, R. A. 1975. A Proof Procedure Using Connection Graphs. *Journal of the ACM* 22(4): 572–595.

Kowalski, R. A., and Kuehner, D. 1971. Linear Resolution with Selection Function. *Artificial Intelligence* 2:227–260.

Kuhns, J. 1967. Answering Questions by Computer: A Logical Study. Technical Report, The Rand Corporation, Santa Monica, California.

Kuipers, B.; Paredaens, J.; Smits, M.; and den Bussche, J. V. 1996. Termination Properties of Spatial Datalog Programs. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 95–109. New York: Springer-Verlag.

Laurent, D., and Vrain, C. 1996. Learning Query Rules for Optimizing Databases with Update Rules. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 173–192. New York: Springer-Verlag.

Lefebvre, L. 1992. Towards an Efficient Evaluation of Recursive Aggregates in Deductive Databases. Paper presented at the Fourth International Conference on Fifth-Generation Computer Systems (FGCS), 1–5 June, Tokyo, Japan.

Leone, N., and Rullo, P. 1992. The Safe Computation of the Well-Founded Semantics for Logic Programming. *Information Systems* 17(1): 17–31.

Leone, N.; Rullo, P.; and Scarcello, F. 1996. Stable Model Checking for Disjunctive Programs. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 281–294. New York: Springer-Verlag.

Lesperance, Y.; Levesque, H.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. 1994. A Logical Approach to High-Level Robot Programming—A Progress Report. Paper presented at the 1994 AAAI Fall Symposium on Control of the Physical World by Intelligent Systems, November, New Orleans.

Levien, R., and Maron, M. 1965. Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval. The Rand Corporation, Santa Monica, California.

Levy, A., and Sagiv, Y. 1995. Semantic Query Optimization in Datalog Programs. In *Principles of Database Systems 1995 (PODS95),* 163–173. New York: Association of Computing Machinery.

Levy, A.; Rajaraman, A.; and Ordille, J. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the Twenty-Second Very Large Data Base Conference,* 251–262. San Francisco, Calif.: Morgan Kaufmann.

Levy, A.; Mendelzon, A.; Sagiv, Y.; and Srivastava, D. 1995. Answering Queries Using Views. In Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-95), 95–104. New York: Association of Computing Machinery.

Ling, T.-W.; Mendelzon, A.; and Vieille, L., eds. 1995. *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD'95).* LNCS 1013. Heidelberg: Springer-Verlag.

Lloyd, J. W., and Topor, R. W. 1985. A Basis for Deductive Database Systems. *Journal of Logic Programming* 2(2): 93–109.

Lobo, J.; Minker, J.; and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming.* Cambridge, Mass.: MIT Press.

Loveland, D.; Reed, D.; and Wilson, D. 1993. SATCHMORE: SATCHMO with Relevancy. Technical report, CS-1993-06, Department of Computer Science, Duke University.

McCarthy, J. 1980. Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence* 13:27–39.

McDermott, D., and Doyle, J. 1980. Nonmonotonic Logic I. *Artificial Intelligence* 13:41–72.

McSkimin, J., and Minker, J. 1977. The Use of a Semantic Network in Deductive Question-Answering Systems. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 50–58. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Manthey, R., and Bry, F. 1988. SATCHMO: A Theorem Prover Implemented in PROLOG. In *Proceedings of the Ninth International Conference on Automated Deduction (CADE),* 415–434. Lecture Notes in Computer Science 310. New York: Springer-Verlag.

Martin, C., and Sistac, J. 1996. Applying Transition Rules to Bitemporal Deductive Databases for Integrity Constraint Checking. In *Logic in Databases (LID'96),* eds. D. Pedreschi and C. Zaniolo, 111–128. New York: Springer-Verlag.

Melton, J. 1996. An SQL3 Snapshot. In Proceedings of the Twelfth International Conference on Data Engineering, 666–672. Washington, D.C.: IEEE Computer Society.

Melton, J., and Simon, A. R. 1993. *Understanding the New SQL: A Complete Guide.* San Francisco, Calif.: Morgan Kaufmann.

Miller, R.; Ioannidis, Y.; and Ramakrishnan, R. 1994. Translation and Integration of Heterogeneous Schemas: Bridging the Gap between Theory and Practice. *Information Systems* 19(1): 3–31.

Minker, J. 1996. Logic and Databases: A 20-Year Retrospective. In *Logic in Databases,* eds. D. Pedreschi and C. Zaniolo, 3–57. New York: Springer-Verlag.

Minker, J. 1993. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming* 17(2–4): 95–126.

Minker, J. 1989. Toward a Foundation of Disjunctive Logic Programming. In *Proceedings of the North American Conference on Logic Programming,* 121–125. Cambridge, Mass.: MIT Press.

Minker, J., ed. 1988a. *Foundations of Deductive Databases and Logic Programming.* San Francisco, Calif.: Morgan Kaufmann.

Minker, J. 1988b. Perspectives in Deductive Databases. *Journal of Logic Programming* 5:33–60.

Minker, J. ed. 1986. Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming. College Park, Md.: University of Maryland Institute for Advanced Computer Studies.

Minker, J. 1982. On Indefinite Databases and the Closed-World Assumption. In *Lecture Notes in Computer Science* 138, 292–308. New York: Springer Verlag.

Minker, J. 1978. Search Strategy and Selection Function for an Inferential Relational System. *Transactions on Data Base Systems* 3(1): 1–31.

Minker, J., and Nicolas, J.-M. 1982. On Recursive Axioms in Deductive Databases. *Information Systems* 7(4): 1–15.

Minker, J., and Rajasekar, A. 1990. A Fixpoint Semantics for Disjunctive Logic Programs. *Journal of Logic Programming* 9(1): 45–74.

Minker, J., and Ruiz, C. 1996. Mixing a Default Rule with Stable Negation. In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, 122–125. Available from Martin Columbic, Bar-Ilan University, Department of Computer Science, Tel Aviv, Israel.

Minker, J., and Ruiz, C. 1994. Semantics for Disjunctive Logic Programs with Explicit and Default Negation. *Fundamenta Informaticae* 20(3–4): 145–192.

Minker, J., and Sable, J. 1970. Relational Data System Study. Technical Report RADC-TR-70-180, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York.

Moore, R. 1985. Semantical Considerations on Nonmonotonic Logic. *Artificial Intelligence* 25:75–94.

Morishita, S.; Derr, M.; and Phipps, G. 1993. Design and Implementation of the GLUE-NAIL Database System. In Proceedings of the ACM-SIGMOD'93 Conference, 147–167. New York: Association of Computing Machinery.

Muggleton, S., and DeRaedt, L. 1994. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming* 19–20:629–679.

Mumick, I.; Finkelstein, S.; Pirahesh, H.; and Ramakrishnan, R. 1990. Magic Is Relevant. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 247–258. New York: Association of Computing Machinery.

Naughton, J., and Sagiv, Y. 1987. A Decidable Class of Bounded Recursions. In Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 227–236. New York: Association of Computing Machinery.

Ng, R., and Subrahmanian, V. 1993. Probabilistic Logic Programming. *Information and Computation* 101(2): 150–201.

Nicolas, J.-M. 1979. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica* 18(3): 227–253.

Nicolas, J.-M., and Syre, J. C. 1974. Natural Question-Answering and Automatic Deduction in System Syntex. In *Proceedings of the IFIP Congress 1974,* 595–599. Berlin: Springer-Verlag.

Pearce, P., and Wagner, G. 1989. Logic Programming with Strong Negation. In *Proceedings of the International Workshop on Extensions of Logic Programming,* ed. P. Schroeder-Heister, 311–326. Lecture Notes in Artificial Intelligence. New York: Springer-Verlag.

Piatetsky-Shapiro, G., and Frawley, W. J., eds. 1991. *Knowledge Discovery in Databases*. Menlo Park, Calif.: AAAI Press.

Powell, P. 1977. Answer-Reason Extraction in a Parallel Relational Data Base System. Master's thesis, Department of Computer Science, University of Maryland.

Pradhan, S., and Minker, J. 1995. Combining Datalog Databases Using Priorities. *Journal of Intelligent Cooperative Information Systems* 4(3): 231–260.

Przymusinski, T. 1995. Static Semantics for Normal and Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence* 14:323–357.

Przymusinski, T. 1990. Stationary Semantics for Disjunctive Logic Programs and Deductive Databases. In *Proceedings of the North American Conference on Logic Programming,* eds. S. Debray and M. Hermenegildo, 40–62. Cambridge, Mass.: MIT Press.

Przymusinski, T. C. 1991. Stable Semantics for Disjunctive Programs. *New Generation Computing* 9:401–424.

Przymusinski, T. C. 1990. Extended Stable Semantics for Normal and Disjunctive Programs. In *Proceedings of the Seventh International Logic Programming Conference,* eds. D. Warren and P. Szeredi, 459–477. Cambridge, Mass.: MIT Press.

Przymusinski, T. C. 1988. On the Declarative Semantics of Deductive Databases and Logic Programming. In *Foundations of Deductive Databases and Logic Programming,* eds. J. Minker, 193–216. San Francisco, Calif.: Morgan Kaufmann.

Ramamohanarao, K. 1993. An Implementation Overview of the ADITI Deductive Database System. In *Third International Conference (DOOD'93),* 184–203. Lecture Notes in Computer Science 760. New York: Springer-Verlag.

Ramakrishnan, R. 1995. *Applications of Logic Databases.* New York: Kluwer Academic.

Ramakrishnan, R., and Ullman, J. 1995. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming* 23(2): 125–149.

Ramakrishnan, R.; Srivastava, D.; and Sudarshan, S. 1992. CORAL—Control, Relations, and Logic. In *Proceedings of the Eighteenth International Conference on Very Large Databases,* ed. L.-Y. Yuan, 238–250. San Francisco, Calif.: Morgan Kaufmann.

Raphael, B. 1968. A Computer Program for Semantic Information Retrieval. In *Semantic Information Processing,* ed. M. Minsky, 33–134. Cambridge, Mass.: MIT Press.

Reiter, R. 1990. On Asking What a Database Knows. In *Computational Logic, e*d. J. W. Lloyd. Basic Research Series. New York: Springer-Verlag.

Reiter, R. 1988. On Integrity Constraints. In *Proceedings of the Second Conference on the Theoretical Aspects of Reasoning about Knowledge,* ed. M. Y. Vardi, 97–111. San Francisco, Calif.: Morgan Kaufmann.

Reiter, R. 1984. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling,* eds. M. Brodie, J. Mylopoulos, and J. Schmit, 163–189. New York: Springer-Verlag.

Reiter, R. 1980. A Logic for Default Reasoning. *Artificial Intelligence* 13:81–132.

Reiter, R. 1978a. Deductive Question-Answering on Relational Data Bases. In *Logic and Data Bases,* eds. H. Gallaire and J. Minker, 149–177. New York: Plenum.

Reiter, R. 1978b. On Closed-World Databases. In *Logic and Data Bases,* eds. H. Gallaire and J. Minker, 55–76. New York: Plenum.

Robinson, J. H. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM* 12(1): 23–41.

Rohmer, J.; Lescoeur, R.; and Kerisit, J.-M. 1986. The Alexander Method: A Technique for the Processing of Recursive Axioms in Deductive Databases. *New Generation Computing* 4(3): 273–285.

Ross, K. 1990. Modular Stratification and Magic Sets for DATALOG Programs with Negation. In Proceedings of the ACM Symposium on Principles of Database Systems, 161–171. New York: Association of Computing Machinery.

Ross, K. 1989. Well-Founded Semantics for Disjunctive Logic Programs. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases,* 352–369. New York: North Holland.

Sacca, D., and Zaniolo, C. 1986. On the Implementation of a Simple Class of Logic Queries. In Proceedings of the ACM Symposium on Principles of Database Systems, 16–23. New York: Association of Computing Machinery.

Sandewall, E. 1994. The Range of Applicability of Some Non-Monotonic Logics for Strict Inertia. *Journal of Logic and Computation* 4(5): 581–616.

Savitt, D.; Love, H.; and Troop, R. 1967. ASP: A New Concept in Language and Machine Organization. In 1967 Spring Joint Computer Conference, 87–102.

Schlipf, J. 1995. Complexity and Undecidability Results for Logic Programming. *Annals of Mathematics and Artificial Intelligence* 15(3-4): 257–288.

Seipel, D. 1995. Efficient Reasoning in Disjunctive Deductive Databases. Ph.D. thesis, Department of Computer Science, University of Tubingen.

Selman, B., and Kautz, H. 1996. Knowledge Compilation and Theory Approximation. *Journal of the ACM* 43(2): 193–224.

Shepherdson, J. 1988. Negation in Logic Programming. In *Foundations of Deductive Databases and Logic Programming,* ed. J. Minker, 19–88. San Francisco, Calif.: Morgan Kaufmann.

Sickel, S. 1976. A Search Technique for Clause Interconnectivity Graphs. *IEEE Transactions on Computers* C-25(8): 823–835.

Sistla, P., and Wolfson, O. 1995. Temporal Conditions and Integrity Constraint Checking in Active Database Systems. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, 269–280. New York: Association of Computing Machinery.

Snodgrass, R. 1987. The Temporal Query Language TQUEL. *ACM Transactions on Database Systems* 12(2): 247–298.

Subrahmanian, V., and Jajodia, S. 1995. *Multimedia Database Systems.* New York: Springer Verlag.

Subrahmanian, V.; Adali, S.; Brink, A.; Emery, R.; Lu, J.; Rajput, A.; Rogers, T.; and Ross, R. 1994. HERMES: A Heterogeneous Reasoning and Mediator System. Available from www.cs.umd.edu/projects/hermes/overview/paper.

Tsur, S., and Zaniolo, C. 1986. LDL: A Logic-Based Data Language. Paper presented at the VLDB Conference, 25–28 August, Kyoto, Japan.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems, Volume II: The New Technologies.* Principles of Computer Science Series. Rockville, Md.: Computer Science Press.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems, Volume I.* Principles of Computer Science Series. Rockville, Md.: Computer Science.

Vaghani, J.; Ramamohanarao, K.; Kemp, D. B.; and Stuckey, P. J. 1991. Design Overview of the ADITI Deductive Database System. In Proceedings of the Seventh International Conference on Data Engineering, 240–247. Washington, D.C.: IEEE Computer Society.

van Emden, M., and Kowalski, R. 1976. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM* 23(4): 733–742.

Van Gelder, A. 1988. Negation as Failure Using Tight Derivations for General Logic Programs. In *Foundations of Deductive Databases and Logic Programming,* ed. J. Minker, 149–176. San Francisco, Calif.: Morgan Kaufmann.

Van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the Association for Computing Machinery* 38(3): 620–650.

Van Gelder, A.; Ross, K.; and Schlipf, J. 1988. Unfounded Sets and Well-Founded Semantics for General Logic Programs. In Proceedings of the Seventh Symposium on the Principles of Database Systems, 221–230. New York: Association of Computing Machinery.

Vieille, L. 1986. Recursive Axioms in Deductive Databases: The Query-Subquery Approach. Paper presented at the First International Conference on Expert Database Systems, 1–4 April, Charleston, South Carolina.

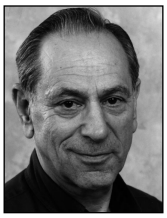
Vielle, L.; Bayer, P.; and Kuechenhoff, V. 1996. Integrity Checking and Materialized View Handling by Update Propagation in the EKS-V1 System. In *Materialized Views*, eds. A. Gupta and I. Mumick. Cambridge, Mass.: MIT Press.

Vieille, L.; Bayer, P.; Kuechenhoff, V.; Lefebvre, A.; and Manthey, R. 1992. The EKS-V1 System. In *Proceedings of Logic Programming and Automated Reasoning*, 504–506. New York: Springer-Verlag.

Xerox. 1989. HIPAC: A Research Project in Active, Time-Constrained Databases. Technical Report 187, Xerox Advanced Information Technologies, Palo Alto, California.

Yuan, L., and You, J.-H. 1993. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning* 10:143–160.

Zaniolo, C. 1995. Active Database Rules with Transaction-Conscious Stable Models Semantics. In *Proceedings of Deductive Object-Oriented Databases 1995*, 55–72. New York: Springer-Verlag.



Jack Minker is a professor of computer science in the Department of Computer Science and the Institute for Advanced Computer Studies at the University of Maryland. His research areas are deductive databases, logic programming, AI, and nonmonotonic reasoning. He was the first chairman of the Department of Computer Science at the University of Maryland from 1974 to 1979 and chairman of the Advisory Committee on Computing at the National Science Foundation from 1979 to 1982. In 1985, Minker received the Association for Computing Machinery (ACM) Outstanding Contribution Award for his work in human rights. He is a fellow of the American Association for the Advancement of Science, a founding fellow of the American Association for Artificial Intelligence, a fellow of the Institute of Electrical and Electronics Engineers, and a founding fellow of the ACM. He received the University of Maryland Presidential Medal for 1996 and is a distinguished scholar-teacher for 1997 to 1998. His e-mail address is minker@cs.umd.edu.Subject: IEA/AIE-98 in Cooperation with AAAI