Logic and Knowledge Bases

J. Grant
Towson University, USA

J. Minker
University of Maryland at College Park, USA

INTRODUCTION

Knowledge bases (KBs) must be able to capture a wide range of situations. One must be able to represent and answer questions regarding indefinite information where it is not clear that there is a unique answer to a question. One must also represent and answer questions about negative information. We discuss a powerful way to represent such information, namely through reasoning about knowledge bases using logic.

In the real world, information known at one time may change. However, in first-order logic, information once known cannot change. This phenomenon is known as *monotonicity*. Since KBs deal with incomplete information, they are not monotonic. We shall discuss a form of *logic programming*, below, which is able to handle nonmonotonic information and situations required by KBs such as definite and indefinite data, and logical and default negation.

The question of how to adapt first-order logic to handle complex situations started in the 1950s. Early systems handled problems in an ad hoc way. Several primitive deductive databases (DDBs), function-free logic programs, were developed in the 1960s. Robinson (1965) developed a general method for automated theorem proving to perform deduction. This method is known as the *Robinson Resolution Principle*; it is a generalization of *modus ponens* to first-order predicate logic. Green and Raphael (1968) were the first to recognize the importance and applicability of the work performed by Robinson and developed a system using this principle.

November 1977 is generally considered to be the start of the modern era of DDBs. A workshop, "Logic and Data Bases," was organized in Toulouse, France, by Gallaire and Nicolas in collaboration with Minker. The workshop included researchers who had performed work in deduction from 1969 to 1977 using the Robinson Resolution Principle. The book *Logic and Data Bases*, edited by Gallaire and Minker (1978), contained these papers. Many significant contributions were described in the book. Nicolas and Gallaire discussed the difference between model theory and proof theory. They demonstrated that the approach taken by the database community was model theoretic—that is, the database represents the truths of the theory, and queries are answered by a bottom-up search. However, in logic programming, answers to a query use a proof theoretic approach, starting from the query, in a top-down search. Reiter discussed the *closed world assumption* (*CWA*), whereby in a theory, if one cannot prove that an *atomic formula* is *true*, then the negation of the atomic formula is assumed to be true. The *CWA* is a *default rule* that permits one to make a decision on negation even if the decision may not be correct.

Reiter's paper elucidated three major issues: the definition of a query, an answer to a query, and how one deals with negation. Clark presented an alternative theory of negation, the concept of if-and-only-if conditions that underlie the meaning of negation, called negation-as-finite-failure. The Reiter and Clark papers are the first to formally define default negation in logic programs and deductive databases. Several implementations of deductive databases were reported. Nicolas and Yazdanian described the importance of integrity constraints in deductive databases. The book provided, for the first time, a comprehensive description of the interaction between logic and databases, and knowledge bases.

References to work on the history of the development of the field of deductive databases and to a description of early systems may be found in Minker (1996).

BACKGROUND

Much of the world's data is stored in relational databases. A relational database consists of tables, each with a fixed number of rows. Each row of a table contains information about a single object. For example, an employee table may contain columns for an employee number, name, address, age, salary, and department name. Each row contains data about one employee. In the same database a department table may contain department name, department number, phone number, and manager's employee number. The connection between the two tables is provided by the common column on department

name. Relational databases also allow for integrity constraints that prevent some types of incorrect updates. For example, the specification of employee number as the key of the employee table means that only one row is allowed for any employee number.

Writing a relational database in logic formalism, we associate a predicate with each table, using the same name for convenience. Then an atomic formula (atom), such as,

Department(sales, 5, 1234567, 11223),

means that there is a row in the Department table with the values listed there. In general, an atom consists of a predicate and a set of arguments which may be constants, variables, or function symbols with arguments. We shall deal only with function-free atoms. Deductive databases extend the concept of relational databases by allowing tables to be defined implicitly by using a formula. In this example, we may define an intensional predicate,

Supervisor(emp1,emp2)←Employee(emp1,_,_,_,_,dept1),
Department(dept1,_, _, emp2)

to stand for the fact that emp2 is the manager of emp1's department. We use underscores to indicate irrelevant attributes.

This type of definition is allowed for relational databases, where it is called a *view*.

However, the following definition:

Superior(emp1, emp2) ← Supervisor(emp1, emp2)

Superior(emp1, emp2) ← Supervisor(emp1, emp3),
Superior(emp3, emp2)

where superior stands for the supervisor, the supervisor's supervisor, and so on, uses recursion and was not allowed in the original relational database framework.

More formally, a deductive database, DDB, is a triple, <*EDB, IDB, IC*>. *EDB,* the extensional database, is a set of facts, namely the rows in tables. *IDB,* the *intensional database*, is a set of rules that implicitly define new tables. *IC* is a set of *integrity constraints*. All three parts of a *DDB* are written as logic formulas. Queries are also written as logic formulas. For example, the query:

← Employee(_, name, address, _, _, deptname),
Department(deptname, 5, _,_)

asks for the names and addresses of employees in department 5, including the name of the department, while the query:

← Supervisor(11223, emp)

asks for the supervisors of employee *11223*.

The *IDB,* in the general case, contains a set of rules of the form:

A1, . . ., An ← B1, . . . , Bm, not Bm+1, . . . , not Bm+k (1)

where all the Ai, $1 \le i \le n$, and Bj , $1 \le j \le m + k$, are literals (atoms or logically negated atoms, e.g., $\neg p$, where p is an atom), and *not* stands for default negation. Whereas logical negation specifies that an atom is definitely not *true*, default negation *not* is an implicit form of negation that permits one to conclude that a defaulted atom is not *true*, even if it is not explicitly known to be not *true*. The left-hand side of the reverse implication is called the *head* and the right hand side is the *body*. The meaning of such a rule is:

A1 or . . . , or An is true if B1 and . . . , and Bm
and not Bm+1 and , . . . , not Bm+k are true.

A *logic program* is a collection of rules of the form (1). Since we deal only with function-free rules, we call such a set of rules a *deductive database*. There are different kinds of deductive databases depending upon the rules used.

The first generalization of relational databases permitted function-free recursive Horn rules in a database—that is, rules in which n = 1 and k = 0. These deductive databases are called *Datalog* databases. Formulas where the head is empty are also allowed: they stand for queries and some types of integrity constraints. When the formula is considered to be a query, *Q(X1, . . . , Xn)*, and hence the head is empty and the free variables are *X1, . . . , Xn*, an answer to the query has the form < *a1, . . . , an* > so that Q*(a1, . . . , an)* follows from the database.

MAIN FOCUS OF THE ARTICLE

Datalog Semantics

Van Emden and Kowalski (1976) formalized the semantics of logic programs that consist of Horn rules, where the rules are not necessarily function-free. They recognized that these programs can be characterized in three distinct ways: by model, fixpoint, or proof theory, leading to the same semantics. When the logic program is function-free, their work provides the semantics for *Datalog* databases. Horn rules may be recursive, that is, a predicate on the left-hand side of a rule may have the same predicate on the right-hand side of the rule. Hence,