Chapter 2.19
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(empl, emp2) ← Employee(empl, _, _, _,_,, dept1),
Department(dept1,_, _, emp2)

to stand for the fact that emp2 is the manager of empl'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)