KNOWLEDGE ENGINEERING

Hojjat Adeli

Vol.II, Applications

KNOWLEDGE ENGINEERING

Volume II APPLICATIONS

Hojjat Adeli, Editor
Department of Civil Engineering
The Ohio State University

McGraw-Hill Publishing Company

New York St. Louis San Francisco Auckland Bogotá Caracas
Hamburg Lisbon London Madrid Mexico Milan Montreal New Delhi
Oklahoma City Paris San Juan São Paulo Singapore Sydney Tokyo Toronto

This book was set in Times Roman by the College Composition Unit
in cooperation with Ruttle Shaw & Wetherill, Inc.
The editors were B. J. Clark and John M. Morriss;
the production supervisor was Leroy A. Young.
The cover was designed by Carla Bauer.
Project supervision was done by The Total Book.
R. R. Donnelley & Sons Company was printer and binder.

KNOWLEDGE ENGINEERING

Vol. II, Applications

2 3 4 5 6 7 8 9 0 DOC DOC 9 5 4 3 2 1 0

ISBN 0-07-000357-2

Library of Congress Cataloging-in-Publication Data

Knowledge engineering / [edited by] Hojjat Adeli.
p. cm.
Includes index.
Contents: v. 1. Fundamentals—v. 2. Applications.
ISBN 0-07-000355-6 (v. 1)—ISBN 0-07-000357-2 (v.2)
1. Expert systems (Computer science) I. Adeli, Hojjat, (date).
QA76.76.E95K577 1990
006.3'3—dc20 89-8338

EDITOR'S BIOGRAPHY

Currently professor of civil engineering at The Ohio State University, Hojjat Adeli received his Ph.D. from Stanford University in 1976. He is the editor-in-chief of the international journal, *Microcomputers in Civil Engineering,* and the author or editor of nearly 200 research publications including several books in the fields of knowledge engineering and expert systems, computer-aided design, parallel processing, mathematical optimization and simulation, applied mechanics, and structural engineering. He is also the editor-in-chief of the forthcoming Marcel Dekker book series *New Generation Computing.* The first two volumes of series, *Supercomputing in Engineering Analysis* and *Parallel Processing in Computational Mechanics* are scheduled for publication in late 1990. He is listed in twelve *Who's Who*'s and biographical listings including *Who's Who in the World, Men of Achievement,* and *International Directory of Distinguished Leadership.*

CONTRIBUTORS

Ronald C. Arkin is an assistant professor in the school of information and computer science at the Georgia Institute of Technology. He received his Ph.D. from the University of Massachusetts in 1987. Between 1977 and 1985 he taught at Hawthorne College in Antrim, New Hampshire, where he was recognized as the faculty member of the year in 1983. Dr. Arkin's research interests are centered upon intelligent action and perception, particularly as evidenced by mobile robots and computer vision. He is listed in *Who's Who in the World, Who's Who in the East,* and *Who's Who of Emerging Leaders in America.*

Thomas L. Dean received his Ph.D. from Yale University in 1986. He is currently an assistant professor in the department of computer science at Brown University. His research interests include logic programming, deductive retrieval methods, and robot problem solving.

John Fox is head of the biomedical computing unit at the Imperial Cancer Research Fund, London, United Kingdom. He is editor of *The Knowledge Engineering Review,* an international journal. He has published extensively in the area of decision theory and technology.

John Grant received his Ph.D. from New York University in 1970. He is currently professor of computer and information science at Towson State University and visiting professor of computer science at the University of Maryland. He has published extensively in the area of databases and logic. He is the author of *Logical Introduction to Databases,* a textbook for advanced undergraduates.

Gail E. Kaiser is an assistant professor of computer science at Columbia University. She received her Ph.D. from Carnegie Mellon University in 1986. She was selected as a U.S. National Science Presidential Young Investigator in 1988. Her research interests include programming environments, application

of AI technology to software development and maintenance, object-oriented languages and databases, and distributed systems.

Steven L. Lytinen is an assistant professor of electrical engineering and computer science at the University of Michigan. He received his Ph.D. from Yale University in 1984, where he wrote a knowledge-based machine translation system. One focus of his research has been on the interaction of different levels of linguistic knowledge in natural language analysis, and recently he has begun extending this work to speech recognition. Other research interests include language acquisition and acquiring knowledge from instructions.

Jack Minker received his Ph.D. from the University of Pennsylvania in 1959. He is currently a professor in the department of computer science and Institute for Advanced Computer Studies at the University of Maryland. He is on the editorial board of the *Journal of Logic Programming, Information Systems, IEEE Expert, Encyclopedia of Artificial Intelligence, Computing Reviews, Expert Systems: Research and Applications, International Series in Logic Programming,* and the *Journal of Academic Proceedings of Soviet Jewry.* He was the first chair of the department of computer science at the University of Maryland from 1974–1979 and served as chair of the Advisory Committee on computing to the U.S. National Science Foundation from 1980–1982. He has numerous publications in the areas of artificial intelligence, automated theorem proving, deductive databases, and logic programming. He has edited several books on deductive databases and logic programming, the most recent of which is *Foundations of Deductive Databases and Logic Programming.*

Setsuo Ohsuga received his Ph.D. from the University of Tokyo in 1966. He is currently a professor in the Research Center for Advanced Science and Technology at the University of Tokyo. He is now the vice president of the Japanese Society for Artificial Intelligence. Professor Ohsuga is a regional editor of the journal of *Knowledge-Based Systems.* His research interests include artificial intelligence, knowledge processing, databases, and CAD.

Mike G. Rodd received his B.Sc., M.Sc., and Ph.D. degrees from the University of Cape Town before joining the University of the Witwatersrand as professor of electronics in 1978. He became head of the department in 1982. He is currently head of the department of electrical and electronic engineering at the University College of Swansea, Wales. He has published widely in the areas of distributed computer control systems, real-time operating systems, computer architecture, computer vision, and the social impacts of automation. He is editor-in-chief of the international journal, *Engineering Applications of AI.*

Spyros G. Tzafestas received his M.Sc. in automatic control from the University of London (Imperial College) in 1967. He received a Ph.D. in electrical engineering and D.Sc. for his published work in the systems and control field from Southampton University, England, in 1969, and 1978, respectively. From

1973–1984, he was professor of systems and control engineering at Patras University, Greece. He is currently professor of computer science at the National Technical University of Athens, Greece. He has published 12 books and over 200 technical papers. He is editor-in-chief of the *Journal of Intelligent and Robotic Systems*, associate editor of three journals, and editor of the Reidel book series *Microprocessor-Based Systems Engineering*. He is a fellow of IEEE and the Institution of Electrical Engineers (United Kingdom), and vice president of the International Association for Mathematics and Computers in Simulation. His research interests include distributed-parameter and large-scale systems, reliability and maintenance optimization, fault detection, robotics, and knowledge-based intelligent systems.

Larry Wos is a senior mathematician in the Mathematics and Computer Science Division at Argonne National Laboratory. He has published extensively on automated theorem proving and automated reasoning and has authored two books on the subject. He is editor-in-chief of the *Journal of Automated Reasoning* and has been president of the Association for Automated Reasoning since its inception in 1982. He (with a colleague) won the first awarded American Mathematical Society Automated Theorem-Proving prize in 1983.

PREFACE

The first volume of *Knowledge Engineering* presents state-of-the-art reviews and tutorials on fundamental aspects of knowledge engineering. The second volume complements the first by presenting applications of applied artificial intelligence (AI). The field of applied AI and knowledge engineering is very young. Students usually must refer to numerous sources to learn the fundamentals of the subject. The two volumes attempt to present summaries of the various subjects in a single document and are oriented toward practical applications. They are suitable as primary reference books in introductory courses on applied AI and knowledge engineering.

Leading and internationally recognized researchers have contributed to these volumes. We hope this effort becomes a continuing book series with future volumes concentrating on other aspects of knowledge engineering and new applications of AI.

Hojjat Adeli
Editor

CONTENTS

CHAPTER 1

INTEGRITY CONSTRAINTS IN KNOWLEDGE-BASED SYSTEMS

JOHN GRANT
JACK MINKER

1 INTRODUCTION

A database system is a system used for the storage and manipulation of facts. Deductive database and knowledge-based systems can be used for the storage and manipulation of knowledge that consists of facts and rules. We distinguish between a deductive database and a knowledge base by allowing function symbols in the latter but not in the former. An expert system may then be defined as a meta-interpreter over a knowledge base or deductive database that provides facilities such as a natural language interface or an explanation of the reasoning used in obtaining answers.

Both database and knowledge-based systems include integrity constraints, which are statements that must be satisfied by the database or the knowledge base. Much of the literature about knowledge-based systems emphasizes the facts and rules but deals with integrity constraints as a minor matter. In fact, the knowledge-based systems of today consider the facts and rules as the knowledge base. Our point of view is quite different: we consider integrity constraints as the essence of knowledge for a knowledge-based system. The reason is that the inclusion of integrity constraints provides semantic information about the data in the database. This semantic information more precisely defines what may exist in the database.

Consider the following example. In a deductive database, it is possible to write a rule to define a grandparent in terms of parent as follows: X is a grand-

parent of Y if X is a parent of Z and Z is a parent of Y. In a geometric database, a syntactically identical definition can be given for parallel lines in terms of perpendicular lines. That is, X is parallel to Y if X is perpendicular to Z and Z is perpendicular to Y. However, in the first case, there is an integrity constraint on parents: any individual may have no more than two parents. There is no corresponding integrity constraint for perpendicular lines. Thus, semantic considerations may distinguish between syntactically identical facts and rules.

The theme of this chapter is the application of knowledge, in the form of integrity constraints, to problems as diverse as update validation, query optimization, and informative answer generation. Researchers have previously developed special-purpose methods, using integrity constraints, to study these problems individually. Some of these will be reviewed later in the proper context as we describe each topic. Our emphasis will be on the presentation of a unified framework for representing the interaction of integrity constraints with the facts and rules of the knowledge base and on showing how this approach provides solutions to those problems.

Section 2 contains a formal definition of a knowledge-based system. We use the language of first-order logic to describe the contents of a knowledge-based system. First-order logic provides a very useful formalism for dealing with knowledge-based systems, because it is a uniform language for expressing facts, rules, integrity constraints, and queries. The uniform technique used in this paper for the application of integrity constraints in knowledge-based systems is called *partial subsumption*. Section 3 contains a description of this method. In a knowledge-based system, partial subsumption of the integrity constraints can be performed at an initial stage, before the processing of queries and updates. The result of partial subsumption is a set of residues that are then attached to the predicates and are used for various purposes during data manipulation.

The topic of Section 4 is update validation. This was, in fact, the initial and primary reason for the introduction of integrity constraints into database systems. Whenever a database is updated, the integrity constraints must remain true. This concept carries over to knowledge bases. We will show in this section how the generation of residues from the integrity constraints facilitates the validation of updates.

A query optimizer chooses a sequence of operations that translate the original query into an efficient program. Traditional query optimization uses the properties of various relational operators and the physical representation of data such as sizes of tables and indexings but does not use semantic knowledge about the application domain, that is, the integrity constraints. In Section 5 we show how semantic query optimization, that is, the optimization of query translation by the use of integrity constraints, can be applied to knowledge-based systems.

A knowledge-based system should respond to a questioner in an intelligent and cooperative manner. In Section 6 we will show that integrity constraints, via the residues, are highly useful in providing a knowledge-based

system with a cooperative and informative answer. In particular, integrity constraints often provide the reason for the answer to a query. Communicating the reason for the answer to the user may be much more informative than the answer by itself. The last section, Section 7, summarizes the chapter.

2 DEFINITION OF A KNOWLEDGE-BASED SYSTEM

Since knowledge-based systems are commonly formalized in logic, we start this section by providing some terminology and concepts of first-order logic. For general information on the applications of logic to databases see Gallaire et al. (1984). The syntax of first-order logic is based on a class of languages and the formulas that may be written in these languages. Each language contains a set of logic and non-logic symbols. The logic symbols are the same for each language and contain variables, logical connectives and quantifiers, and punctuation symbols; the non-logic symbols are constants, predicates, and functions. The rules for constructing formulas are the same for all first-order languages.

The symbols of a first-order language are as follows.

1. Logic symbols:

   Variables: x, y, z, $x1$, $y1$, $z1$,...(infinitely many)
   Connectives: $\neg$ (not), $\vee$ (or), & (and), $\rightarrow$ (implies)
   Quantifiers: $\forall$ (for all), $\exists$ (there exists)
   Punctuation: parentheses and comma
   Equality: $=$

2. Non-logic symbols:

   Constants: a, b, c, susan, prof,...(as many as needed, possibly 0)
   Predicates: P, R, Teach,...(at least one)
   Functions: f, g, plus,...(as many as needed, possibly 0)

The presence of functions distinguishes knowledge-based systems from deductive database systems. Thus, deductive database systems are a special (but important) case of knowledge-based systems.

The symbols of a language are combined in a standard way to produce terms and formulas. We refer the reader to standard books in logic (Enderton, 1972; Mendelson, 1978) and theorem-proving (Chang and Lee, 1973; Loveland, 1978) for details. As is usual in work on theorem proving, we deal primarily with clauses, where all variables are universally quantified and which have the general form

$$\neg A_1 \vee \ldots \vee \neg A_k \vee A_{k+1} \vee \ldots \vee A_n$$

where $k \leq n$ and each A_i is an atomic formula. We use a Prolog-like notation where in most cases such a clause is written as

$$A_{k+1}, \ldots, A_n \leftarrow A_1, \ldots, A_k$$

with the atoms $A_1, \ldots, A_k$ in the *body* and $A_{k+1}, \ldots, A_n$ in the *head* of the clause. A *ground* clause contains no variables. A *Horn* clause has at most one atom in the head, that is, $k \leq n \leq k + 1$. A *definite* clause has one atom in the head, that is, $n = k + 1$. A clause is *range-restricted* if every variable that appears in the head also appears in the body. Range-restricted clauses are very useful because their truth or falsity does not depend on the set of constants in the language, only on the constants in the clause. A clause is *recursive* if a predicate appears in both the head and the body.

The presence of recursive definitions gives great power to knowledge-based systems over relational database systems. Consider a definition such as the one for Ancestor in terms of Parent:

$$\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y)$$
$$\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, z), \text{Ancestor}(z, y)$$

The second clause is recursive, and the simple query

$$\leftarrow \text{Ancestor}(\text{joe}, x)$$

finds the ancestors of *joe*. However, Ancestor cannot be defined by using the standard relational algebraic operations on Parent.

We mentioned earlier that clauses are usually written in such a way that both the body and the head of a clause contain only positive atoms. However, there are cases where it is useful to define rules using one or more negated atoms in the body. For example,

$$\text{Backordered}(x) \leftarrow \text{Ordered}(x), \neg \text{Instock}(x)$$

defines the predicate Backordered in terms of a negated atom in the head of the clause.

While both recursion and negation in the clause body are important and powerful concepts for knowledge bases, substantial problems may arise when they are combined. Consequently, usually a limitation is placed on the combination of recursion and negation by not allowing recursive definitions via negation. For example,

$$P(x) \leftarrow \neg Q(x)$$
$$Q(x) \leftarrow P(x)$$

would not be allowed. The idea is that in defining a predicate, negation should be applied only to already known predicates. That was the case for the definition of Backordered, as the Instock predicate was assumed to deal only with facts. But in this case, Q does not have a prior definition as far as P is concerned; in fact, Q is defined in terms of P. A knowledge-based (or deductive database) system that applies this restriction is said to be *stratified* [see Apt et

al. (1988), Van Gelder (1988), and Przymusinski (1988) for the complete definition and many results about stratification].

Now we come to the definition of the knowledge base component of a knowledge-based system. Not all of the restrictions that we give are essential for our results, but these restrictions simplify the presentation and proofs of results, and, in any case, most knowledge bases in practice either satisfy the assumptions or can be transformed into such a form. A *knowledge base* $K = \langle \text{TH, IC} \rangle$ consists of two components: a theory TH and a set of integrity constraints IC. In this chapter we consider TH to consist of two distinct sets of clauses and a metarule:

$$\text{TH} = \langle \text{EDB, IDB, NFF} \rangle$$

EDB, the extensional database, is a set of ground atoms; these represent the facts of the knowledge base. The predicates that appear in EDB are called extensional predicates. IDB, the intensional database, is a set of deductive laws (axioms) that are range-restricted Horn clauses that satisfy the property of stratification. We also assume that no predicate is both intensional and extensional. NFF, negation as finite failure, is a metarule used for proving negated clauses as follows: If P is a ground atom, then $\text{TH} \vdash \neg P$ if not[$\text{TH} \vdash P$], that is, if all attempts to prove P terminate in failure. In order to handle equality correctly, we assume that TH also contains $x = x \leftarrow$. Finally, in this chapter we restrict IC to be a set of nonrecursive range-restricted clauses containing only extensional predicates.

A *query* is a conjunction of atoms, Q: $A1$ &...& Ak. We write Q as $Q(x1, \ldots, xn)$ if $x1, \ldots, xn$ are all the variables in Q. An *answer* to $Q(x1, \ldots, xn)$ is a sequence of constants, $\langle a1, \ldots, an \rangle$, such that $\text{TH} \vdash Q(a1, \ldots, an)$, where $Q(a1, \ldots, an)$ stands for the simultaneous substitution of ai for xi in Q. This definition carries over to the case where $n = 0$; such a query is a yes-no question.

We illustrate the notion of a knowledge-based system by providing an example. This example has only a few predicates, no functions, no recursion, and no negated atoms in the body, but it illustrates important concepts and techniques that can be applied in a similar way to large and more complex knowledge bases. First we write the extensional and intensional predicates and provide the attributes for each. The attributes are not used specifically later, but they are helpful in understanding the meaning of the predicates. We do not actually write the facts in the EDB because typically the number of facts is large. We also write each integrity constraint both as a clause and in English.

Example.

Extensional predicates:

Schedule(Teacher Name, Department Name, Course Number)
Registration(Student Name, Department Name, Course Number)
Catalog(Department Name, Course Number, Credits)
Instructor(Teacher Name)

Intensional predicate:

Teacherof(Teacher Name, Student Name)

Extensional database:

Schedule, Registration, Catalog, Instructor

Intensional database:

$$\text{Teacherof}(x, y) \leftarrow \text{Schedule}(x, u, v), \text{Registration}(y, u, v)$$

Intensional constraints:

IC1. Baker teaches only history courses.

$$y = \text{hist} \leftarrow \text{Schedule}(\text{baker}, y, z)$$

IC2. Course numbers in the computer science department are less than 600.

$$y < 600 \leftarrow \text{Catalog}(\text{cosc}, y, z)$$

IC3. No one is a teacher and a student for the same course.

$$\leftarrow \text{Schedule}(x, y, z), \text{Registration}(x, y, z)$$

IC4. Davis is registered only for economics courses.

$$y = \text{econ} \leftarrow \text{Registration}(\text{davis}, y, z)$$

IC5. Every teacher's name in the Schedule relation appears in the Instructor relation.

$$\text{Instructor}(x) \leftarrow \text{Schedule}(x, y, z)$$

3 PARTIAL SUBSUMPTION AND RESIDUES

In order to apply integrity constraints in knowledge-based systems, we use a technique called *partial subsumption*. This method is applied to predicates before data manipulation. Partial subsumption yields fragments of integrity constraints, called *residues*, which are then attached to the predicates and are used later during query processing and updates. The process of partial subsumption, residue generation, and residue attachment to predicates is called *semantic compilation*. This process can be performed at an initial stage and need not be modified after standard data manipulation. Only changes to the IDB or the IC would force a semantic recompilation. In this section we define and illustrate semantic compilation. A formal treatment, including proofs of theorems asserting the correctness of the process, may be found in Chakravarthy et al. (1988).

Partial subsumption is a modification of subsumption, so we start with the latter. Subsumption is a relationship between two clauses.

Definition. A clause C *subsumes* a clause D if there is a substitution σ such that $C\sigma$ is a subclause of D. For example, if

$$C = R(x, b) \leftarrow P(x, y), Q(y, z, b)$$

and

$$D = R(a, b) \leftarrow P(a, z), Q(z, z, b), S(a)$$

then C subsumes D by the substitution $\{a/x, z/y\}$.

To understand partial subsumption, we need to look at the basic subsumption algorithm in Chang and Lee (1973), testing to see if C subsumes D. We explain this algorithm by illustrating its effect on the clauses C and D. First, D is instantiated to a ground clause by using new constants, not present in C or D. We will use $k1, \ldots, kn$ for these constants and call the substitution θ. Here, $\theta = \{k1/z\}$, so that

$$D\theta = R(a, b) \leftarrow P(a, k1), Q(k1, k1, b), S(a)$$

Then $D\theta$ is negated; $\neg D\theta$ is a set of literals. In this case,

$$\neg D\theta = \{ \leftarrow R(a, b), P(a, k1) \leftarrow , Q(k1, k1, b) \leftarrow , S(a) \leftarrow \}$$

Next, the algorithm tries to construct a linear refutation tree with C as the root, using at each step an element of $\neg D\theta$ in the resolution. The result is that C subsumes D if and only if at least one such refutation tree ends with the null clause. Here we obtain the tree shown in Fig. 1-1.

The essence of partial subsumption is the application of the subsumption algorithm to an integrity constraint and the body of an IDB clause (which defines the intensional predicate). As we need to do this process for extensional relations also, we write the trivial axiom $R \leftarrow R$ for each extensional relation, strictly for the purpose of applying the subsumption algorithm in this manner. In general, the subsumption algorithm does not yield the null clause, because the integrity constraint does not subsume the body of the axiom. However, a subclause of the integrity constraint might subsume the body of the axiom. This is

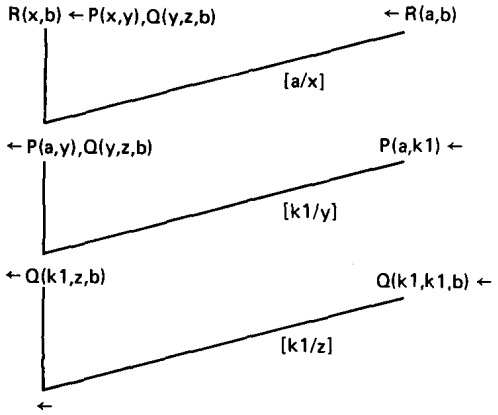

FIGURE 1-1
Linear refutation tree for subsumption.