Developing Expert, Database, and Natural Language Systems

Knowledge Systems and Prolog

Second Edition

Developing Expert, Database, and Natural Language Systems

Knowledge Systems and Prolog

Second Edition

Adrian Walker
Michael McCord
John F. Sowa
Walter G. Wilson

Library of Congress Cataloging - In-Publication Data

Knowledge systems and Prolog : developing expert, database, and
natural language systems / Adrian Walker... -- 2nd ed.
p. cm.
Includes bibliographical references.
ISBN 0-201-52424-4
1. Expert systems (Computer science) 2. Prolog (Computer program
language) 3. Natural language processing (Computer science)
I. Walker, Adrian.
QA76.76.E95K58 1990
006.3'3--dc20

89-18658
CIP

Reproduced by Addison-Wesley from camera-ready copy supplied by the authors.

ISBN 0-201-52424-4
ABCDEFGHIJ-MA-943210

Preface

This book is about knowledge systems. It is about how to design them, how to use them, and how to program them in a language called Prolog. A knowledge system is a computer program that solves, or helps to solve, problems that would otherwise have to be handled by a human expert alone. A knowledge system may be

- an expert system,
- a program that understands English (or another natural language),
- or a combination of the two.

In addition it may make good use of a database management system to store parts of its knowledge.

This second edition of the book coincides with the availability of IBM Prolog as a full IBM program product, supporting both IBM syntax and Edinburgh syntax. The IBM Prolog program product has interfaces to other programming languages, in particular to the database language SQL.

In addition to the expert system and natural language processing topics covered in the first edition, the new edition also shows, with detailed examples, how to build and use English-like intelligent front ends to relational databases for traditional data processing topics such as inventory control and report generation. The integration of data processing and artificial intelligence topics is illustrated by means of practical examples of writing English-like, executable specifications of enterprise business models. The examples include flexible modelling of business objects by inheritance hierarchies with overrides, and automatic explanations of certain kinds of inheritance-based object-oriented reasoning.

Knowledge systems are used to give advice and to solve problems in many areas, such as business, science, technology, and law. The knowledge systems

described in this book have been developed at IBM, and have been used for experiments in several subjects, including manufacturing planning, communication network management, evaluation of the ease of use of software packages, and machine translation from one natural language to another.

Prolog is a programming language based on logical reasoning. As such, it draws strength from the principles of mathematical logic—principles that were developed well before the invention of the computer. Each of us can reason, using knowledge in order to produce advice on several subjects, and each of us can communicate in English or another natural language. Prolog is a very useful programming language in which to write down our methods of reasoning and of using English, so that they can be used by a computer. Prolog forms the basis for a major national research effort in Japan, and it is increasingly used worldwide as a very high level programming language. The Prolog programming techniques in this book are drawn from existing knowledge systems, from classes the authors have taught at IBM over a number of years, and from experience with several programming projects including large system performance analysis, analysis of assembler programs, and Programmable Logic Array simulation. The programming examples are in IBM Prolog, which has built-in links to other software, such as relational database management systems. It is available for the VM and MVS operating systems. (At the time of this writing, the program product "IBM PROLOG for 370" runs under the VM operating system. An earlier program offering, "MVS/Programming in Logic" also runs under the MVS operating system.) This version of Prolog was developed at the Paris Scientific Center of IBM and at IBM laboratories in California.

The book has several levels of information. It contains

- a nontechnical introduction to knowledge systems,
- sections about how to set up and use knowledge systems,
- introductory and advanced sections about how to program in Prolog, and
- some summaries of advanced research.

This is an unusual span of topics and levels in one book. It is made possible, and we think useful, by a unifying theme—the ability of people and of computer programs to reason with everyday logic. Prolog is based on logic, and it provides a clear, almost common sense view of the subject.

In keeping with the span of topics and levels, this book can be read in several ways, with several purposes in mind.

- Because the book is partly about codifying the kinds of knowledge that all of us have, it should be of interest to readers who would like to get a feel for how their own areas of expertise could be represented in a knowledge system.
- Because the book also goes into depth in how to program in Prolog, it can be read as a university text in Prolog programming, with applications to knowledge systems and with pointers for research. In a practical course, it can be used directly with IBM Prolog. For a course that emphasizes

principles, there is material on the logical basis for the semantics of Prolog programs. Because the book describes many interesting aspects of expert systems, databases, and natural language processing, it can be useful in university courses on artificial intelligence, on databases, and in courses for Linguists and Psychologists who are interested in computational experiments in their fields.

- For readers who already know one or more programming languages, this book can be used as an introduction to Prolog that will take them rapidly beyond the introductory level, and that provides interesting examples of knowledge systems. For those who are already Prolog programmers, the book describes IBM Prolog—its syntax, and its rich set of built-in predicates for direct support of many common programming practices.

Five different chapter sequences are suggested.

- For readers who are interested mainly in the possible *uses of expert database systems*, the suggested sequence is
  - Chapter 1,
  - then Section 4.2,
  - with more detail drawn from Chapters 2 and 3, plus the remainder of Chapter 4 as needed.
  - The only prerequisite for this sequence is a liking for computing and for reasoning logically.

- For readers who are interested mainly in *implementing expert database system shells*, the suggested sequence is
  - Chapter 1,
  - then Sections 3.1, 3.2.5, 4.2, 4.3 and 4.4,
  - with more detail drawn from Chapter 2, plus the remainder of Chapters 3 and 4 as needed.

  For this sequence, some background in logic, in relational databases, in programming, or in discrete mathematics will be a help, although the book is self-contained. It will also be useful to have available a computer with IBM Prolog and the SQL database management system installed.

- For readers who are interested mainly in *natural language processing* the suggested sequence is
  - Chapters 1, 2 and 5,
  - with additional material drawn from Chapter 3 as needed.
  - Again, the only prerequisite for this sequence is a liking for computing and for reasoning logically.

- For readers who are interested mainly in *programming in Prolog* the suggested sequence is
  - Chapters 1 and 2,
  - Chapter 3 as far as page 177,
  - Appendix A on how to use IBM Prolog,
  - the rest of Chapter 3, then Chapters 4 and 5.
  - For this sequence, some background in logic, or in programming, or in discrete mathematics will be a help, although the book is self-contained. It will also be useful to have available a computer with IBM Prolog installed.

- For readers who are interested first in the *logical fundamentals* of our approach to knowledge systems and Prolog, the suggested sequence is
  - Chapters 1 and 2,
  - Appendix B on the logical basis for Prolog and Syllog,
  - then material from Chapters 3, 4, and 5 as needed.
  - For this sequence, some background in logic, or in discrete mathematics will be a help, although again the book is self-contained.

There are exercises at the ends of Chapters 2, 3, 4 and 5. The exercises are discussed—and complete solutions are given for some of them—in the text.

Chapter 1, *Knowledge Systems: Principles and Practice*, discusses the importance of knowledge systems. It gives a view of what a knowledge system is and what it should be able to do. We look at the evolution of knowledge systems and their supporting software technology, and we describe the role of logic and of the Prolog language in this evolution. Examples are given to show that logic provides a good common notation for knowledge representation, and we describe some of the trends that will make it easier to transfer knowledge.

Chapter 2, *A Prolog to Prolog*, describes Prolog as a programming language based on symbolic logic. It shows how to write simple Prolog programs and run them on a terminal, and gives many examples of Prolog programs. We look at the declarative and procedural styles and interpretations of Prolog programs. We introduce Prolog data structures and Prolog's all-important generalized pattern match mechanism, which is called *unification.*

Chapter 3, *Programming Techniques in Prolog*, describes some more advanced programming methods. We show the importance of declarative style in writing Prolog programs. We look at this style of writing in Prolog as a distillation of good software engineering practices. We describe some advanced data structures and the relation of data structures to the control and use of recursion. Metalevel programming in Prolog is introduced, and we describe some techniques for data processing and report generation.

Chapter 4, *Expert Database Systems in Prolog,* looks at a specific expert database system shell; how it is used, how knowledge can be written down for it, and how its underlying inference mechanism and database linkage is programmed. We look at ways of using Prolog to reason with diverse representations of knowledge. We describe exact and judgmental reasoning for expert systems, the importance of explanations, and methods of producing helpful explanations.

Chapter 5, *Natural Language Processing in Prolog,* looks at the techniques that can be used to support English dialogue with a knowledge system. We look at the process of translating an English sentence into a special Logical Form Language, so that we can reason directly about the meaning of the sentence. The translation process uses a dictionary, a grammar of English, and a way of transforming the structure of a sentence into a logical form. A technique for representing large lexicons is described. A grammar is used to show the process of extracting an underlying structure from an English sentence. Then we show how to find meaning in the structure by translating it into a logical form.

Chapter 6 contains our *Conclusions*—the book discusses significant aspects of artificial intelligence, databases, logic, and programming. The methods used in the book are based on computational logic, which acts as a bridge between the empirical aspects of knowledge systems and the formal foundations of reasoning in logic. Logic programming, in the form of Prolog, makes it possible to cover an unusually wide range of topics, and to do so in the practical sense that we show how to program much of what we discuss.

There are two appendices. Appendix A, *How to Use IBM Prolog* describes the use of Prolog at a terminal, including techniques for metainterpretation, for holding different programs in one Prolog workspace, and for input and output using files. Appendix B, *Logical Basis for Prolog and Syllog*, outlines a basis in mathematical logic for the meaning of Prolog programs, including a special treatment of negation in Prolog.

It is a pleasure to acknowledge that many conversations with colleagues have helped to shape the material in this book. We would like to thank Andre Algrain, Clemens Beckstein, Arendse Bernth, Shelene Chang, Francisco Corella, Joan Dunkin, Norman Foo, Se June Hong, Ann Gruhn, Dinesh Katiyar, Doug Lorch, Peter Marusek, Alexa McCray, Gustaf Neumann, Richard O'Keefe, Fernando Pereira, Anand Rao, J. Alan Robinson, Bill Santos, Ehud Shapiro, Peter Sheridan, Oded Shmueli, Andrew Taylor, Doug Teeple, Daphne Tzoar, Maarten van Emden, Jean Voldman, Barbara Walker, and Wlodzimierz Zadrozny. We would particularly like to thank Professors Jacques Cohen (Brandeis University), Edward L. Fisher (North Carolina State University), Frank Kriwaczek (Imperial College, London) and Michael Lebowitz (Columbia University) for their incisive reviews of an early version of this book. Special thanks go to John Prager for meticulous reading of a final draft of the first edition, and to Reed Hyde for material on IBM Prolog using the MVS operating system. Chapter 1 and Section 4.2.2 are based on material that has appeared in the IBM Journal of Research and Development and we would like to thank the journal for permission to use that material here. We are most grateful to IBM

for the support we have received for our research in knowledge systems and Prolog, and for the resources that IBM has kindly provided for the preparation of this book. Finally, we thank the people at Addison-Wesley for being so well organized and for their good advice.

Michael McCord, Adrian Walker, Walter G. Wilson
IBM Research Division
T. J. Watson Research Center

Yorktown Heights, N.Y.

John F. Sowa
IBM Systems Research Institute

Thornwood, N.Y.

Contents

1

Knowledge Systems: Principles and Practice

Adrian Walker

> "The object of reasoning is to find out, from the consideration of what we already know, something else which we do not know."
>
> Charles Sanders Peirce, *Fixation of Belief.*

As noted in the Preface, a knowledge system is an expert system, or a program that understands a natural language such as English; it may also be some combination of the two, and it may make good use of a database management system.

The rate of publication of papers about knowledge systems is now higher than ever before. There are survey articles in computer science journals, in popular computing and scientific journals, and in the general press. The volume of research publication is also unusually high. Expert systems have so far proved their worth in structure elucidation in chemistry, in helping to find mineral deposits, in helping technicians in hospitals, in suggesting maintenance procedures for locomotives, and in many specialties for which we do not have enough human experts. The commercial potential of the subject is being recognized.

In a period of such intensive activity, it can be healthy to step back from the day to day excitement of new uses of knowledge systems, new research results, product announcements, and the formation of new companies in the area of knowledge systems. This chapter describes some of the interplay between principles and practice in knowledge systems; we argue that it is very fruitful to combine principles and practice closely, and that logic programming (in

particular Prolog) is a strong candidate for bridging the traditional gap between the two. We set out a particular view of where work on expert systems and natural language processing has come from, what is being done now, and of some trends for the future. Because of the scope of the subject, our view necessarily focuses on just some of the trends and achievements.

What is a knowledge system, an expert system, a natural language understanding program? The name expert system has been applied to many diverse programs. So in the next section we describe the properties that we think such a system should have. Section 1.2 sketches some central issues in expert systems from a historical point of view. In Section 1.3 we outline the role of Prolog and logic programming in expert systems. In Section 1.4 we note that an expert system is only as good as the knowledge it contains, and we describe some methods of knowledge representation. Section 1.5 outlines some of the methods used in getting a computer to understand a natural language such as English. Then in Section 1.6 we describe some trends in knowledge acquisition. Section 1.7 is a summary.

1.1 WHAT IS A KNOWLEDGE SYSTEM?

Every program contains knowledge about some problem. A payroll program, for example, has knowledge about pay rates, deductions, and tax schedules. It also includes "common sense" knowledge about business practices and the number of hours in a week or days in a month. What makes knowledge systems different from conventional programs is that they represent the knowledge in a higher-level form. Instead of encoding knowledge in low-level statements, they store it in a *knowledge base* of rules and facts that stay close to the way people think about a problem. This book presents the two major kinds of knowledge systems:

- Expert systems: problem-solving systems that reach expert or at least highly competent levels of performance.
- Natural language systems: systems that converse with people in their native languages at a level that approaches the generality and flexibility of ordinary discourse.

Besides introducing these subjects, the book goes on to present detailed methods for designing and implementing such systems in Prolog, and for using them when they are completed.

What distinguishes an expert system from a conventional program is not just its expertise, but the way that the expertise is stored and processed. A payroll program, for example, certainly has more expertise about tax rates and deductions than most people, but it applies the expertise in a rigid, inflexible way. Furthermore, it cannot explain its knowledge or answer questions about its use: an employee who believes that the wrong tax rate was applied cannot ask the payroll program why it made a certain deduction. An expert system behaves more like an intelligent assistant. It can apply its knowledge in flexible ways to

novel kinds of problems. Whenever it reaches a conclusion, the user can ask how that conclusion was reached and what rules were used to deduce it.

Since knowledge about any subject is constantly growing and changing, an expert system should be flexible in integrating new knowledge incrementally into the knowledge base. Indeed, the expert system should help the designer to translate knowledge into rules and facts. We would also like it to display its knowledge in a form that is easy for us to read. If we are to take actions that can have serious consequences in the real world, based on advice given by an expert system, then we would like the system to provide explanations of its advice (Michie 1982). Because the expert knowledge that people have is often incomplete and only partly understood, we would like an expert system to be able to reason with judgmental or inexact knowledge. This knowledge may be *declarative* (about the nature of a task), *procedural* (about ways of doing the task efficiently), or both.

Although all expert systems encode their knowledge in a more accessible form than conventional programs, many of them have complex notations that can only be used by a trained computer scientist. There are two complementary methods of making it easier to put in knowledge and to get advice:

- make the notation closer to English (or another natural language)
- make the system able to work effectively when given only *declarative* knowledge about the nature of a task.

Generally, we can make a system easier to use by improving its ability to handle English, by making it so that the knowledge we give it can be more declarative, or by doing both. We can choose how far we go along the path to full English. In this book we shall show a limited approach in Chapter 4, and a full English approach in Chapter 5.

In an approach described in Chapter 4, we work with *sentence forms* on the screen of a computer terminal. Sentence forms consist of any words in English (or any other natural language) together with variables that can be filled in by other words. We give meaning to the sentence forms by using them to write rules containing our knowledge about a subject. The system remembers the sentences that we type in. When we want to ask a question, the sentences the system knows about are presented as a grouped menu, with the sentences that are most important shown in the first group. We pick one sentence; then we optionally modify some of the variables to make the sentence correspond to our question. (If we want to be able to ask the same question using two different sentences, then we must tell the system that the two sentences mean the same thing.) This approach has the disadvantage that it is somewhat restricted. It has the advantage that it works with any natural language or jargon, and that there is no need to construct a dictionary or a grammar beforehand—we simply type in the knowledge and use it. Chapter 4 describes a knowledge base about manufacturing planning that is built in this way.

In Chapter 5 we show how to develop a parser and a semantic interpreter that come to grips with the full richness of natural language. Natural languages

allow people to express information in different forms to emphasize different points or just to allow some stylistic variation. All of the following requests, for example, ask for the same information:

```
What is Sam's age?
How old is Sam?
Please print Sam's age.
I would like to know how old Sam is.
```

A general language handler can parse each sentence to determine its syntax and then build up an internal logical form that describes the meaning. Appropriate rules of inference can relate the different logical forms. They would show, for example, that age is a measure of the degree of oldness. Chapter 5 presents a general method for mapping natural language into a logical form and handling problems of this sort. The methods of Chapter 5 are of value not only for creating natural language interfaces to expert systems, but also for other natural language applications, such as machine translation (McCord 1986).

We mentioned that if a system is to be easy to use, it should cope with sentences that are easy to read, or that are in full English. There is another dimension to ease of use. We also want a system to reason with *declarative* knowledge, so that rather than telling it *how* to do a task, we can just tell it *what* the task is. This means that, internally, the system must be able to derive enough *procedural* knowledge to execute the declarative knowledge both correctly and efficiently. We can choose how far we go along the path to direct use of declarative knowledge. This is discussed in Chapter 4 and in Appendix B. It's worth noting that knowledge systems can be viewed according to the extent and manner in which they separate declarative and procedural concerns. For example, in Kowalski et al. (1988) some research projects in

- legal reasoning,
- capturing common sense general knowledge together with a broad library of specific knowledge to analogize to,
- structuring a large procedural knowledge base, and
- adding highly efficient support for rules inside a relational database system.

are compared from the point of view of use of declarative and procedural knowledge. It appears that this point of view is important in the management of knowledge in large software projects.

In summary, a full-scale knowledge system should be able to do the following tasks:

- Solve or help to solve important problems that would otherwise require the services of a human expert.
- Integrate new knowledge incrementally into the knowledge base.
- Help the designer to elicit, organize, and transfer knowledge.
- Display knowledge in a form that is easy for people to read.