Dennis Merritt

# Building Expert Systems in Prolog

Springer-Verlag
World Publishing Corp

Dennis Merritt

# Building Expert Systems in Prolog

With 27 Illustrations

Springer-Verlag

World Publishing Corp

Dennis Merritt, Newton, MA 02158, USA

*Editors*
Steven S. Muchnick, SUN Microsystems, [illegible] Mountain View, CA 94043, USA
Peter Schnupp, InterFace Computer Gmb[illegible] 5[illegible]000 Mnchen 81, West Germany

Library of Congress Cataloging in Publication Data
Merritt, Dennis.
Building expert systems in Prolog / Dennis Merritt.
p. cm. -- (Springer compass international)
Bibliography: p.
Includes indexes.
ISBN 0-387-97016-9 (alk. paper): $ 34.00 (est.)
1. Expert systems (Computer science) 2. Prolog (Computer program language) I. Title. II. Series.
QA76.76E95M47 1989
006.3'3--dc20 89-6193



Reprinted by World Publishing Corporation, Beijing, 1992
for distribution and sale in The People's Republic of China only
ISBN 7-5062-1291-9

ISBN 0-387-97016-9 Springer-Verlag New York Berlin Heidelberg
ISBN 3-540-97016-9 Springer-Verlag Berlin Heidelberg New York

# Preface

When I compare the books on expert systems in my library with the production expert systems I know of, I note that there are few good books on building expert systems in Prolog. Of course, the set of actual production systems is a little small for a valid statistical sample, at least at the time and place of this writing – here in Germany, and in the first days of 1989. But there are at least some systems I have seen running in real life commercial and industrial environments, and not only at trade shows.

I can observe the most impressive one in my immediate neighborhood. It is installed in the Telephone Shop of the German Federal PTT near the Munich National Theater, and helps configure telephone systems and small PBXs for mostly private customers. It has a neat, graphical interface, and constructs and prices an individual telephone installation interactively before the very eyes of the customer.

The hidden features of the system are even more impressive. It is part of an expert system network with a distributed knowledge base that will grow to about 150 installations in every Telephone Shop throughout Germany. Each of them can be updated individually overnight via Teletex to present special offers or to adapt the selection process to the hardware supplies currently available at the local warehouses.

Another of these industrial systems supervises and controls in "soft" real time the excavators currently used in Tokyo for subway construction. It was developed on a Unix workstation and downloaded to a single board computer using a real time operating system. The production computer runs exactly the same Prolog implementation that was used for programming, too.

And there are two or three other systems that are perhaps not as showy, but do useful work for real applications, such as oil drilling in the North Sea, or estimating the risks of life insurance for one of the largest insurance companies in the world. What all these systems have in common is their implementation language: Prolog, and they run on "real life" computers like Unix workstations or minis like VAXs. Certainly this is one reason for the preference of Prolog in commercial applications.

But there is one other, probably even more important advantage: Prolog is a programmer's and software engineer's dream. It is compact, highly readable, and arguably the "most structured" language of them all. Not only has it done away with virtually all control flow statements, but even explicit variable assignment, too!

These virtues are certainly reason enough to base not only systems, but textbooks, on this language. Dennis Merritt has done this in an admirable manner. He explains the basic principles, as well as the specialized knowledge representation and processing techniques that are indispensable for the implementation of industrial software such as those mentioned above. This is important because the foremost

reason for the relative neglect of Prolog in expert system literature is probably the prejudice that "it can be used only for backward chaining rules." Nothing is farther from the truth. Its relational data base model and its underlying unification mechanism adapt easily and naturally to virtually any programming paradigm one cares to use. Merritt shows how this works using a copious variety of examples. His book will certainly be of particular value for the professional developer of industrial knowledge-based applications, as well as for the student or programmer interested in learning about or building expert systems. I am, therefore, happy to have served as his editor.

Peter H. Schnupp
Munich, January 1989

# Acknowledgements

A number of people have helped make this book possible. They include Dave Litwack and Bill Linn of Cullinet who provided the opportunity and encouragement to explore these ideas. Further thanks goes to Park Gerald and the Boston Computer Society, sounding boards for many of the programs in the book. Without the excellent Prolog products from Cogent, AAIS, Arity, and Logic Programming Associates none of the code would have been developed. A special thanks goes to Peter Gable and Paul Weiss of Arity for their early help and Allan Littleford, provider of both Cogent Prolog and feedback on the book. Jim Humphreys of Suffolk University gave the most careful reading of the book, and advice based on years of experience. As have many other Mac converts, I feel compelled to mention my Macintosh SE, Microsoft Word and Cricket Draw for creating an enjoyable environment for writing books. And finally without both the technical and emotional support of Mary Kroening the book would not have been started or finished.

# Contents

# 1
# Introduction

Over the past several years there have been many implementations of expert systems using various tools and various hardware platforms, from powerful LISP machine workstations to smaller personal computers.

The technology has left the confines of the academic world and has spread through many commercial institutions. People wanting to explore the technology and experiment with it have a bewildering selection of tools from which to choose. There continues to be a debate as to whether or not it is best to write expert systems using a high-level shell, an AI language such as LISP or Prolog, or a conventional language such as C.

This book is designed to teach you how to build expert systems from the inside out. It presents the various features used in expert systems, shows how to implement them in Prolog, and how to use them to solve problems.

The code presented in this book is a foundation from which many types of expert systems can be built. It can be modified and tuned for particular applications. It can be used for rapid prototyping. It can be used as an educational laboratory for experimenting with expert system concepts.

## 1.1 Expert Systems

Expert systems are computer applications which embody some non-algorithmic expertise for solving certain types of problems. For example, expert systems are used in diagnostic applications servicing both people and machinery. They also play chess, make financial planning decisions, configure computers, monitor real time systems, underwrite insurance policies, and perform many other services which previously required human expertise.

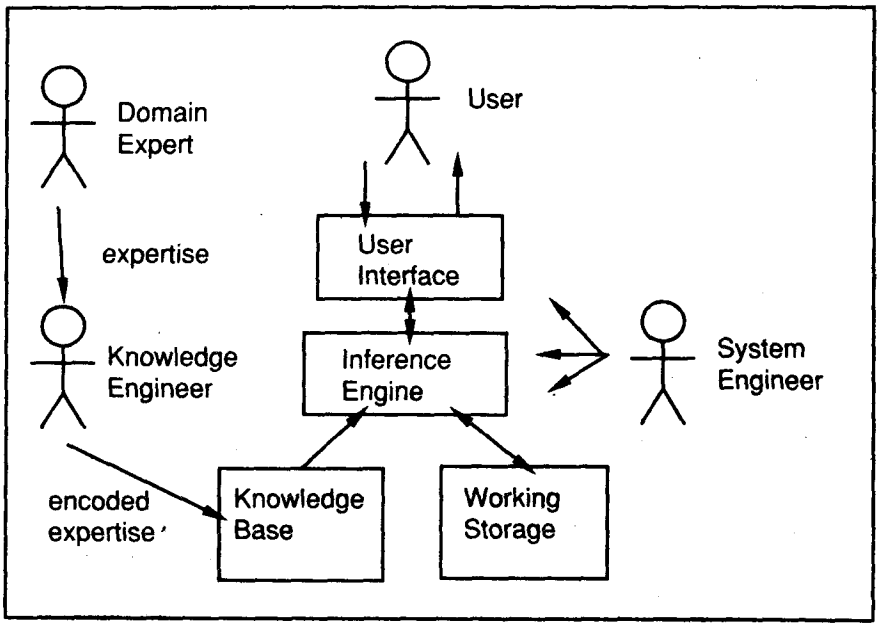


Figure 1.1 Expert system components and human interfaces

Expert systems have a number of major system components and interface with individuals in various roles. These are illustrated in figure 1.1. The major components are:

- Knowledge base - a declarative representation of the expertise, often in IF THEN rules;
- Working storage - the data which is specific to a problem being solved;
- Inference engine - the code at the core of the system which derives recommendations from the knowledge base and problem-specific data in working storage;
- User interface - the code that controls the dialog between the user and the system.

To understand expert system design, it is also necessary to understand the major roles of individuals who interact with the system. These are:

- Domain expert - the individual or individuals who currently are experts solving the problems the system is intended to solve;

- Knowledge engineer - the individual who encodes the expert's knowledge in a declarative form that can be used by the expert system;

- User - the individual who will be consulting with the system to get advice which would have been provided by the expert.

Many expert systems are built with products called expert system shells. The shell is a piece of software which contains the user interface, a format for declarative knowledge in the knowledge base, and an inference engine. The knowledge engineer uses the shell to build a system for a particular problem domain.

Expert systems are also built with shells that are custom developed for particular applications. In this case there is another key individual:

- System engineer - the individual who builds the user interface, designs the declarative format of the knowledge base, and implements the inference engine.

Depending on the size of the project, the knowledge engineer and the system engineer might be the same person. For a custom built system, the design of the format of the knowledge base, and the coding of the domain knowledge are closely related. The format has a significant effect on the coding of the knowledge.

One of the major bottlenecks in building expert systems is the knowledge engineering process. The coding of the expertise into the declarative rule format can be a difficult and tedious task. One major advantage of a customized shell is that the format of the knowledge base can be designed to facilitate the knowledge engineering process.

The objective of this design process is to reduce the semantic gap. Semantic gap refers to the difference between the natural representation of some knowledge and the programmatic representation of that knowledge. For example, compare the semantic gap between a mathematical formula and its representation in both assembler and FORTRAN. FORTRAN code (for formulas) has a smaller semantic gap and is therefor easier to work with.

Since the major bottleneck in expert system development is the building of the knowledge base, it stands to reason that the semantic gap between the expert's representation of the knowledge and the

representation in the knowledge base should be minimized. With a customized system, the system engineer can implement a knowledge base whose structures are as close as possible to those used by the domain expert.

This book concentrates primarily on the techniques used by the system engineer and knowledge engineer to design customized systems. It explains the various types of inference engines and knowledge bases that can be designed, and how to build and use them. It tells how they can be mixed together for some problems, and customized to meet the needs of a given application.

## 1.2 Expert System Features

There are a number of features which are commonly used in expert systems. Some shells provide most of these features, and others just a few. Customized shells provide the features which are best suited for the particular problem. The major features covered in this book are:

- Goal driven reasoning or backward chaining - an inference technique which uses IF THEN rules to repetitively break a goal into smaller sub-goals which are easier to prove;
- Coping with uncertainty - the ability of the system to reason with rules and data which are not precisely known;
- Data driven reasoning or forward chaining - an inference technique which uses IF THEN rules to deduce a problem solution from initial data;
- Data representation - the way in which the problem specific data in the system is stored and accessed;
- User interface - that portion of the code which creates an easy to use system;
- Explanations - the ability of the system to explain the reasoning process that it used to reach a recommendation.

## Goal-Driven Reasoning

Goal-driven reasoning, or backward chaining, is an efficient way to solve problems that can be modelled as "structured selection" problems. That is, the aim of the system is to pick the best choice from many enumerated possibilities. For example an identification problem falls in this category. Diagnostic systems also fit this model, since the aim of the system is to pick the correct diagnosis.

The knowledge is structured in rules which describe how each of the possibilities might be selected. The rule breaks the problem into sub-problems. For example, the following top level rules are in a system which identifies birds.

**IF**
**family is albatross and**
**color is white**
**THEN**
**bird is laysan albatross.**

**IF**
**family is albatross and**
**color is dark**
**THEN**
**bird is black footed albatross.**

The system would try all of the rules which gave information satisfying the goal of identifying the bird. Each would trigger sub-goals. In the case of these two rules, the sub-goals of determining the family and the color would be pursued. The following rule is one that satisfies the family sub-goal:

**IF**
**order is tubenose and**
**size large and**
**wings long narrow**
**THEN**
**family is albatross.**

The sub-goals of determining color, size, and wings would be satisfied by asking the user. By having the lowest level sub-goal satisfied or denied by the user, the system effectively carries on a dialog with the user. The user sees the system asking questions and responding to answers as it attempts to find the rule which correctly identifies the bird.

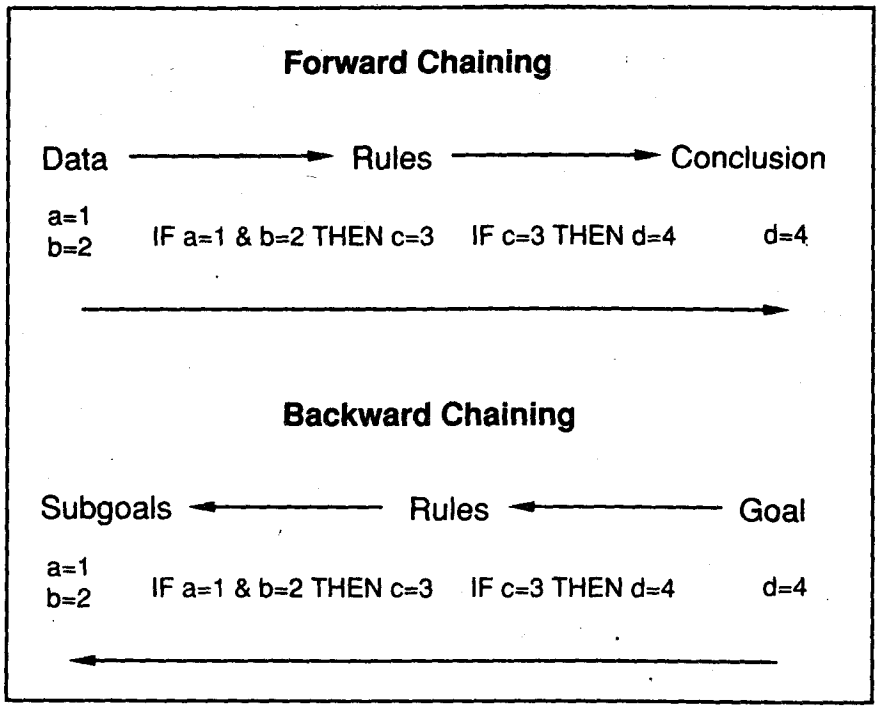


Figure 1.2. Difference between forward and backward chaining

## Uncertainty

Often times in structured selection problems the final answer is not known with complete certainty. The expert's rules might be vague, and the user might be unsure of answers to questions. This can be easily seen in medical diagnostic systems where the expert is not able to be definite about the relationship between symptoms and diseases. In fact, the doctor might offer multiple possible diagnoses.

For expert systems to work in the real world they must also be able to deal with uncertainty. One of the simplest schemes is to associate a numeric value with each piece of information in the system. The numeric value represents the certainty with which the information is known. There are numerous ways in which these numbers can be defined, and how they are combined during the inference process.