PROLOG AND EXPERT SYSTEMS

PROLOG AND EXPERT SYSTEMS

Kenneth A. Bowen

Applied Logic Systems, Inc.
and
Syracuse University

McGraw-Hill, Inc.

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

This book was set in Times Roman by Publication Services.
The editor was Eric M. Munson.
The production supervisor was Friederich W. Schulte.
The cover was designed by Joseph Gillians.
Project supervision was done by Publication Services.
R. R. Donnelley & Sons Company was printer and binder.

PROLOG AND EXPERT SYSTEMS

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

ISBN 0-07-006731-7

Library of Congress Cataloging-in-Publication Data

Bowen, Kenneth A.
Prolog and expert systems / Kenneth A. Bowen—1st ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-07-006731-7
1. Expert systems (Computer science) 2. Prolog (Computer program language)
QA76.76.E95B67 1991
006.3'3—dc20 90-24032

Also Available from McGraw-Hill

Schaum's Outline Series in Computers

Most outlines include basic theory, definitions, and hundreds of solves problems and supplementary problems with answers.

Titles on the Current List Include:

Advanced Structured Cobol
Boolean Algebra
Computer Graphics
Computer Science
Computers and Business
Computers and Programming
Data Processing
Data Structures
Digital Principles, 2d edition
Discrete Mathematics
Essential Computer Mathematics
Linear Algebra, 2d edition
Mathematical Handbook of Formulas & Tables
Matrix Operations
Microprocessor Fundamentals, 2d edition
Programming with Advanced Structured Cobol
Programming with Assembly Language
Programming with Basic, 3d edition
Programming with C
Programming with Fortran
Programming with Pascal
Programming with Structured Cobol

Schaum's Solved Problems Books

Each title in this series is a complete and expert source of solved problems containing thousands of problems with worked out solutions.

Related Titles on the Current List Include:

3000 Solved Problems in Calculus
2500 Solved Problems in Differential Equations
2000 Solved Problems in Discrete Mathematics
3000 Solved Problems in Linear Algebra
2000 Solved Problems in Numerical Analysis

Available at your College Bookstore. A complete listing of Schaum titles may be obtained by writing to: Schaum Division
McGraw-Hill, Inc.
Princeton Road, S-1
Hightstown, NJ 08520

ABOUT THE AUTHOR

Kenneth A. Bowen, founder and President of Applied Logic Systems, Inc., is currently a professor of computer science at Syracuse University, Syracuse, New York, where he was formerly a professor of mathematics. He recieved his B.S.,M.S., and PhD degrees (in mathematics) from the University of Illinois at Urbana, Illinois. Dr. Bowen is a world-wide recognized authority on logic programming and Prolog, and has been an invited speaker at conferences throughout the United States, Europe and Asia. Dr. Bowen was an invited participant in the joint Japanese – U.S. Seminar on the Current State of Logic Programming, sponsored jointly by MITI (Japan) and NSF (USA) at Argonne National Laboratories in October, 1989, and served as the General Chairman for the Joint Meeting of the Fifth International Conference and Symposium on Logic Programming held in Seattle in 1988.

With his extensive experience, he has acted as a consultant to several major computer manufacturers and defense contractors, as well as various agencies of the Federal Government. As a research professor at Syracuse University, Dr. Bowen has directed a large number of government- and corporate-funded projects in the area of logic programming as it relates to knowledge base maintenance. He was one of the principal investigators for the Rome Air Development Command Artificial Intelligence Consortium and was the organizer, and one of the presenters, of a nation-wide live closed-circuit television presentation on Prolog and Expert Systems broadcast in December 1985.

PREFACE

This book is the result of years of indulgence: indulgence in a love of logic and its computational manifestation in Prolog, indulgence in the joy of teaching Prolog to my students, friends, and family (even when they had reservations about what it was I was leading them into), and indulgence in the delight of using odd and anthropomorphic methaphors and pictures to convey the ideas. I've tried, in the static pages of this book, to capture some of the pleasure I've found in teaching these ideas. I can only hope that some of it comes through.

One of the striking and successful areas of application of Prolog is the implementation of expert systems. I have oriented the presentation of Prolog programming toward the presentation of several approaches to the implementation of expert systems. The exercises utilize information from the everyday and scientific worlds as the basis for developing aspects of such systems. More substantial raw information has been included in the Appendix. There are several sequences of exercises that use the various parts of the Appendix. The following table delineates them.

§	A-TV	B-Enzymes	C-Clouds	D-IRS	E-PhotoEquip	F-AutoRepair
1	1.8,1.9	1.12	1.10,1.11	1.13	1.14	
2	2.8,2.9	2.11	2.10	2.12	2.13	
3	3.8,3.9	3.7	3.6	3.10		3.11
4				4.10		
5	5.12	5.2	5.13			
6	6.15	6.16				
7	7.10	7.8		7.6,7.11-12		
8	8.1		8.5	8.3	8.2	8.6
9	9.1		9.4	9.3	9.2	9.5
10						
11			11.1	11.1	11.1	11.1
12					12.1	12.2

The approach of this book grew out of my approach to the teaching of programming languages at the School of Computer and Information Science at Syracuse University. I usually begin a programming language course with a brisk introduction to the language in question, intending to provide an overall view and appreciation of the language's salient features. I generally omit or brush lightly over the more detailed or obscure aspects of the language during this introduction. Basic competence with the language is reinforced in the students through the use of many small exercises. During the second portion of the course, I usually undertake one or more reasonably large programming problems. As we develop the program(s), I refine the student's understanding and use of the language in the context at hand. I have found this to be a particularly successful approach for several reasons: First, if the large programs are well selected, they are of inherent interest to the students, and hence capture their attention more than sequences of small exercises. Consequently, the students appear much more receptive to discussions of the fine points of the programming language, since these points often have significant bearing on the overall large program. Second, undertaking a reasonably large programming task enables me to discuss and illustrate software design issues, which are all too often glossed over in basic programming language courses. Part I of this book thus reflects the introductory portions of my Prolog programming courses, and Part II, on the implementation of expert systems using Prolog, corresponds to the second portion of my courses. Instructors and students should cover Part I at a brisk pace, and in particular, should just skim Chapter 7. Almost all aspects of Prolog are seriously exercised in Part II during the development of various expert systems. As differing aspects of Prolog are encountered in Part II, readers and instructors should return to the relevant portions of Part I (in particular Chapter 7) for review.

It is a customary statement, yet true, that this book would not have existed without the help of many people. First must come all my many students and friends—together with my family—all of whom entered into the learning process with me, and who taught me what worked (and what didn't) to convey the ideas. More immediately, my colleagues at Applied Logic Systems—Johanna Bowen, Kevin Buettner, Keith Hughes, and Andy Turk, have proven invaluable critics and contributors of ideas. The following reviewers were also helpful in reviewing early stages of the manuscript: Charles Frank, Northern Kentucky University; Forouzan Golshani, Arizona State University; Dennis Kibler, University of California, Irvine; and Kathleen Swigger, North Texas State University. And finally, but not least important, my wife and daughters, Johanna, Melissa, and Alexandra, have been among my sharpest critics and most imaginative contributors. To all, I say thank you. They have improved the book inestimably. The remaining flaws belong only to me.

Kenneth A. Bowen

CONTENTS

PART ONE

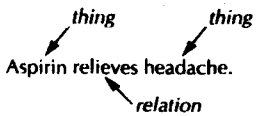
CORE PROLOG

CHAPTER 1

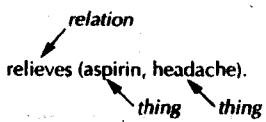
PROLOG DATABASES

1.1 ELEMENTARY PROLOG DATABASES

The best way to start programming is to write a program. We will start with a simple Prolog program consisting of several facts about drugs and diseases. Let us start by putting down a number of simple assertions: that aspirin relieves headaches, that it also relieves moderate pain, that KO Diarrhea relieves diarrhea, that Noasprinol relieves headache, and so on. Each of these assertions claims that a simple relationship holds:



This would be expressed in Prolog by



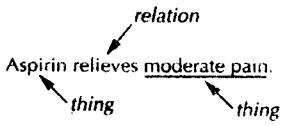
This is an example of a Prolog fact. Facts assert that relationships hold between things. The fact displayed above asserts that the relationship "relieves" holds between the two things "aspirin" and "headache." We will see later that Prolog facts can assert relationships holding between two things, three things, four things, etc. There can even be relations that involve only one thing or no things. We will make sense of that later.

Since relations are very important to Prolog, the name of the relation comes first in the Prolog statement. Then comes the list of things involved in the relationship. The

list is enclosed in parentheses, and the things are separated by commas. The entire expression is terminated with a period (.), sometimes called a full stop. Spaces can be used almost everywhere to improve readability with one exception: No space can occur between the relation name and the left-hand parenthesis.

Prolog generally insists that the names of relations and things begin with a lower-case letter. This is why we changed "Aspirin" to "aspirin."

Consider another assertion:



This also asserts that the relation "relieves" holds between two things, but in this case, the expression naming the second thing is a two-word English phrase, namely, "moderate pain." Since Prolog is basically very dumb, it can't handle names made up of more than one word. What can we do? One solution is just to leave out the space:

```
moderatepain
```

But this is hard to read. We can improve things by capitalizing the *p* in *pain*:

```
moderatePain
```

Even though Prolog insists that names begin with a lower-case letter, the rest of the letters in the name can be lower- or upper-case. You can also use digits after the first letter of a Prolog name, and you can use the underbar character (_). Using the underbar character can definitely improve the readability of Prolog names, for example:

```
moderate_pain
```

One can even write

```
moderate_Pain
```

Now that we see how to name "moderate pain" in Prolog, it is easy to see that the Prolog version of the assertion "Aspirin relieves moderate pain" would be

```
relieves(aspirin, moderate_pain).
```

Some other interesting assertions are that "aspirin aggravates asthma," "aspirin aggravates peptic ulcer," "DeCongest aggravates high blood pressure," etc. The assertion that aspirin aggravates peptic ulcer would translate into

```
aggravates(aspirin, peptic_ulcer).
```

The assertion that DeCongest aggravates high blood pressure would translate into

```
aggravates(de_congest,high_blood_pressure).
```

Each of these Prolog facts asserts that a particular relationship holds among several things. A collection of Prolog facts is called a Prolog *database*. Databases are the

FIGURE 1–1
A Prolog database of simple medical facts.

```
relieves(aspirin, headache).
relieves(aspirin, moderate_pain).
relieves(aspirin, moderate_arthritis).
relieves(aspirin_codeine_combination, severe_pain).
relieves(cough_cure_xm, cough).
relieves(pain_gone, severe_pain).
relieves(ko_diarrhea, diarrhea).
relieves(de_congest, cough).
relieves(de_congest, nasal_congestion).
relieves(penicillin, pneumonia).
relieves(bis_cure, diarrhea).
relieves(bis_cure, nausea).
relieves(noasprinol, headache).
relieves(noasprinol, moderate_pain).
relieves(triple_tuss, nasal_congestion).
aggravates(aspirin, asthma).
aggravates(aspirin, peptic_ulcer).
aggravates(ko_diarrhea, fever).
aggravates(de_congest, high_blood_pressure).
aggravates(de_congest, heart_disease).
aggravates(de_congest, diabetes).
aggravates(de_congest, glaucoma).
aggravates(penicillin, asthma).
aggravates(bis_cure, diabetes).
aggravates(bis_cure, gout).
aggravates(bis_cure, fever).
```

simplest kind of Prolog program. Figure 1–1 shows a Prolog database that includes the elementary facts we've developed above.

These Prolog facts have a natural intuitive translation back into English. Remember that the first part of a Prolog fact is the name of a relation holding between the list of things that follow. Thus,

```
relieves(de_congest, nasal_congestion).
```

would translate into

DeCongest relieves nasal congestion.

The fact

```
aggravates(ko_diarrhea, fever).
```

translates into

KO Diarrhea aggravates fever.

Besides relations involving just two things, there are many relations that involve more than two things. For example, consider the assertion that

Interstate 81 connects Binghamton and Syracuse.

The relationship here is the "connects" relation. The things involved in the relationship are Interstate 81, Binghamton, and Syracuse. How do we translate this into a Prolog fact? Remember that the name of the relation comes first, followed by the things involved in the relationship, so the corresponding Prolog fact is

```
connects(interstate_81, binghamton, syracuse).
```

The assertion

Binghamton and Syracuse are connected by Interstate 81.

conveys the same information as the original, but with a different emphasis. The Prolog version of this is

```
connected_by(binghamton, syracuse, interstate_81).
```

Another natural relation involving three things is the "parents of" relation:

Ken and Johanna are the parents of Melissa.

This translates to

```
parents_of(ken, johanna, melissa).
```

The original statement can also be phrased as

Melissa's parents are Ken and Johanna.

This version becomes

```
parents(melissa, ken, johanna).
```

Notice that we could have also translated the last assertion into the fact

```
parents_are(melissa, ken, johanna).
```

Whether or not to include such helping words as *is* and *are* in the names of relations is a matter of personal taste.

In everyday conversation, there aren't very many natural relationships involving four or more things. Of course, one can easily concoct lots of artificial relationships with four or five or more participants. Heres one natural relationship involving four things:

The states of Arizona, Colorado, New Mexico, and Utah meet in a common point.

We can translate this into

```
common_point(arizona, colorado, new_mexico, utah).
```

There are many occasions when we must assert that something has a certain property. For example:

Nasal congestion is a mild condition.

Heart disease is a serious condition.

In the first example, the property is "being a mild condition"; in the second, it is "being a serious condition." In English, we usually write the property after the thing for which it holds. However, in the corresponding Prolog facts, the property name occurs first:

```
mild_condition(nasal_congestion).
serious_condition(heart_disease).
```

Prolog thinks of properties and relations as being very similar sorts of notions. For this reason, we will use the word *predicate* to refer to either properties or relations. So "relieves," "aggravates," "connects," "parents," "mild_condition," and "serious_condition" are all predicates.

From the examples so far, we can see that the basic form of Prolog facts consists of a predicate name followed by the names of the things involved:

predicate(*thing*$_1$*name*, *thing*$_2$*name*, . . .).

Every Prolog fact must be terminated with a period (full stop). The *predicate name* part of a fact normally appears first, although we will see a few exceptions later. The *things* related to one another by a predicate are called its *arguments*. Thus,

aspirin, headache

is the list of arguments of "relieves" in the fact

```
relieves(aspirin, headache).
```

The arguments of "aggravates" in the fact

```
aggravates(bis_cure, gout).
```

are

bis_cure, gout.

The names of predicates are made up of letters, digits, and the underbar character (_); but remember, the first character of the name must be a lower-case letter.

The names of things are called *constants* (sometimes called *atomic constants* or simply *atoms*). Like the names of predicates, they are made up of letters, digits, and the underbar character, and must start with a lower-case letter.

If you choose the names of your predicates and constants to be as intuitive as possible, it is very easy to understand the intended meaning of your Prolog facts. The meaning we intended for the first "relieves" fact in Figure 1–1 is that

Aspirin relieves headache.

The meaning of the last fact in the database is that

Bis Cure aggravates fever.