Logic Programming with Prolog

Max Bramer

Logic Programming with Prolog

Second Edition

Springer

Max Bramer
School of Computing
University of Portsmouth
Portsmouth, UK

ISBN 978-1-4471-5486-0 ISBN 978-1-4471-5487-7 (eBook)
DOI 10.1007/978-1-4471-5487-7
Springer London Heidelberg New York Dordrecht

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Introduction

Logic Programming is the name given to a distinctive style of programming, very different from that of conventional programming languages such as C++ and Java. Fans of Logic Programming would say that 'different' means clearer, simpler and generally better!

Although there are other Logic Programming languages, by far the most widely used is **Prolog**. The name stands for Programming in Logic. This book teaches the techniques of Logic Programming through the Prolog language. Prolog is based on research by computer scientists in Europe in the 1960s and 1970s, notably at the Universities of Marseilles, London and Edinburgh. The first implementation was at the University of Marseilles in the early 1970s. Further development at the University of Edinburgh led to a *de facto* standard version, now known as Edinburgh Prolog. Prolog has been widely used for developing complex applications, especially in the field of Artificial Intelligence. Although it is a general-purpose language, its main strengths are for symbolic rather than for numerical computation.

The developers of the language were researchers working on automating mathematical theorem proving. This field is often known as *computational logic*. But if you are not a Computer Scientist, a logician or a mathematician do not let this deter you! This book is aimed at the 99.9 % of the population who are none of these. Those who are, already have a number of excellent textbooks from which to choose.

The idea that the methods developed by computational logicians could be used as the basis for a powerful general purpose programming language was revolutionary 30 years ago. Unfortunately most other programming languages have not yet caught up.

The most striking feature of Prolog for the newcomer is how much simpler the programs look than in other languages. Many language designers started out with good intentions but could not resist, or perhaps could not avoid, making their creations over elaborate. In some languages even writing the customary test program to print out the words *Hello World!* to the user's screen is hard work. All the user has to do in Prolog is to enter **write('Hello World!').**

Traditional programming languages have one feature in common. They all contain a series of instructions that are to be performed ('executed') one after another.

This style of programming is called *procedural*. It corresponds closely to the way computers are generally built. This is a tempting approach that has been used since the 1950s but is ultimately flawed. The way users write programs should depend as little as possible on how the underlying machine was built and as much as possible on what the user is trying to do. In just the same way, the facilities I use when I drive my car should depend as little as possible on how the engine was designed or the carburettor works. I want all that level of detail hidden from me, although in practice I understand that it may not be completely possible to ignore it.

Prolog programs are often described as *declarative*, although they unavoidably also have a procedural element. Programs are based on the techniques developed by logicians to form valid conclusions from available evidence. There are only two components to any program: facts and rules. The Prolog system reads in the program and simply stores it. The user then types in a series of questions, known as *queries*, which the system answers using the facts and rules available to it. This is a simple example, a series of queries and answers about animals. The program consists of just seven lines (blank lines are ignored).

```
dog(fido).
dog(rover).
dog(henry).
cat(felix).
cat(michael).
cat(jane).
animal(X):-dog(X).
```

This program is not too hard to decipher. The first three lines are *facts*, with the obvious interpretation that fido, rover and henry are all dogs. The next three facts say that felix, michael and jane are all cats.

The final line is a *rule* saying that anything (let us call it X) is an animal if it is a dog. Cat lovers may feel that cats can also claim to be called animals, but the program is silent about this.

Having loaded the program, the user is then faced with the two character symbol **?-** which is called the *system prompt*. To check whether fido is a dog all that is necessary is to type the query **dog(fido)** followed by a full stop and press the 'return' key, which indicates to the system that a response is needed. This gives the complete dialogue:

?- dog(fido).
true.

The user can enter a series of queries at the prompt, asking for further information.

?-dog(jane). [Is jane a dog? No - a cat]
false.

?- animal(fido). [Is fido an animal?]
true. [yes - because it is a dog and any dog is an animal]

?- dog(X). [Is it possible to find anything, let us call it X, that is a dog?]

X = fido; [All 3 possible answers are provided]
X = rover;
X = henry

?-animal(felix). [felix is a cat and so does not qualify as an animal, as far as the program is concerned]
false.

Although straightforward, this example shows the two components of any Prolog program, *rules* and *facts*, and also the use of *queries* that make Prolog search through its facts and rules to work out the answer. Determining that fido is an animal involves a very simple form of logical reasoning:

GIVEN THAT
any X is an animal if it is a dog

AND
fido is a dog

DEDUCE
fido must be an animal

This type of reasoning is fundamental to theorem proving in Mathematics and to writing programs in Prolog.

Even very simple queries such as:

?-dog(fido).

can be looked at as asking the Prolog system to prove something, in this case that fido is a dog. In the simplest cases it can do so simply by finding a fact such as **dog(fido)** that it has been given. The system answers '**true**' to indicate that this simple 'theorem' has been proved.

You have now seen all three elements needed for logic programming in Prolog: facts, rules and queries. There are no others. Everything else is built from them.

A word of warning is necessary at this stage. It is easy for the newcomer to get started with Prolog, but do not be fooled by these examples into thinking that Prolog is only capable of handling simple (Mickey Mouse?) problems. By putting these very basic building blocks together Prolog provides a very powerful facility for building programs to solve complex problems, especially ones involving reasoning, but all Prolog programs are simple in form and are soundly based on the mathematical idea of proving results from the facts and rules available.

Prolog has been used for a wide variety of applications. Many of these are in Mathematics and Logic but many are not. Some examples of the second type of application are

- programs for processing a 'natural language' text, to answer questions about its meaning, translate it to another language etc.
- advisory systems for legal applications
- applications for training
- maintaining databases for the Human Genome project
- a personnel system for a multi-national computer company
- automatic story generation
- analysing and measuring 'social networks'
- a software engineering platform for supporting the development of complex software systems
- automatically generating legally correct licences and other documents in multiple languages
- an electronic support system for doctors.

Prolog is also being used as the basis for a standard 'knowledge representation language' for the Semantic Web – the next generation of internet technology.

Prolog is one of the principal languages used by researchers in Artificial Intelligence, with many applications developed in that field, especially in the form of *Expert Systems* – programs that 'reason out' the solution to complex problems using rules.

Many textbooks on Prolog assume that you are an experienced programmer with a strong background in Mathematics, Logic or Artificial Intelligence (preferably all three). This book makes no such assumptions. It starts from scratch and aims to take you to a point where you can write quite powerful programs in the language, often with considerably fewer lines of program 'code' than would be needed in other languages.

You do not need to be an experienced programmer to learn Prolog. Some initial familiarity with basic computing concepts such as program, variable, constant and function would make it easier to achieve, but paradoxically too much experience of writing programs in other languages may make the task harder – it may be necessary to unlearn bad habits of thinking learnt elsewhere.

Some Technical Details

Experienced programmers will search this book in vain for such standard language features as variable declarations, subroutines, methods, for loops, while loops or assignment statements. (If you don't know what these terms mean, don't worry – you will not be needing them.)

On the other hand experienced readers may like to know that Prolog has a straightforward uniform syntax, programs that are equivalent to a database of facts and rules, a built-in theorem prover with automatic backtracking, list processing, recursion and facilities for modifying programs (or databases) at run-time. (You probably will not know what most of these mean either – but you will be using all of them by the end of this book.)

Prolog lends itself to a style of programming making particular use of two powerful techniques: recursion and list processing. In many cases algorithms that would require substantial amounts of coding in other languages can be implemented in a few lines in Prolog.

There are many versions of Prolog available for PC, Macintosh and Unix systems, including versions for Microsoft Windows, to link Prolog to an Oracle relational database and for use with 'object-oriented' program design. These range from commercial systems with many features to public domain and 'freeware' versions. Some of these are listed (in alphabetical order) below, together with web addresses at which more information can be found.

- Amzi! Prolog
  http://www.amzi.com/products/prolog_products.htm
- B-Prolog
  http://www.probp.com/
- Ciao Prolog
  http://clip.dia.fi.upm.es/Software/Ciao/
- GNU Prolog
  http://gnu-prolog.inria.fr/
- Logic Programming Associates Prolog (versions for Windows, DOS and Macintosh)
  http://www.lpa.co.uk
- PD Prolog (a public domain version for MS-DOS only)
  http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/lang/prolog/impl/prolog/pdprolog/0.html
- SICStus Prolog
  http://www.sics.se/isl/sicstuswww/site/index.html
- SWI Prolog (public domain versions for Windows, Linux and Macintosh)
  http://www.swi-prolog.org/
- Turbo Prolog (an old version that only runs in MS-DOS)
  http://www.fraber.de/university/prolog/tprolog.html

- Visual Prolog
  http://www.visual-prolog.com/
- W-Prolog (a Prolog-like language that runs in a web browser)
  http://waitaki.otago.ac.nz/~michael/wp/
- YAP Prolog
  http://www.ncc.up.pt/~vsc/Yap/

The programs in this book are all written using the standard 'Edinburgh syntax' and should run unchanged in virtually any version of Prolog you encounter (unfortunately, finding occasional subtle differences between implementations is one of the occupational hazards of learning any programming language). Features such as graphical interfaces, links to external databases etc. have deliberately not been included, as these generally vary from one implementation to another. All the examples given have been tested using version 6.2.6 of SWI-Prolog, a popular public-domain version of the language which is available on a range of platforms.

This second edition has been expanded by the addition of two further chapters illustrating the use of grammar rules to analyse English sentences and the use of Prolog for Artificial Intelligence applications.

Each chapter has self-assessment exercises to enable you to check your progress. A full glossary of the technical terms used completes the book.

Max Bramer
Emeritus Professor of Information Technology
University of Portsmouth, Portsmouth, UK
October 2013

Contents