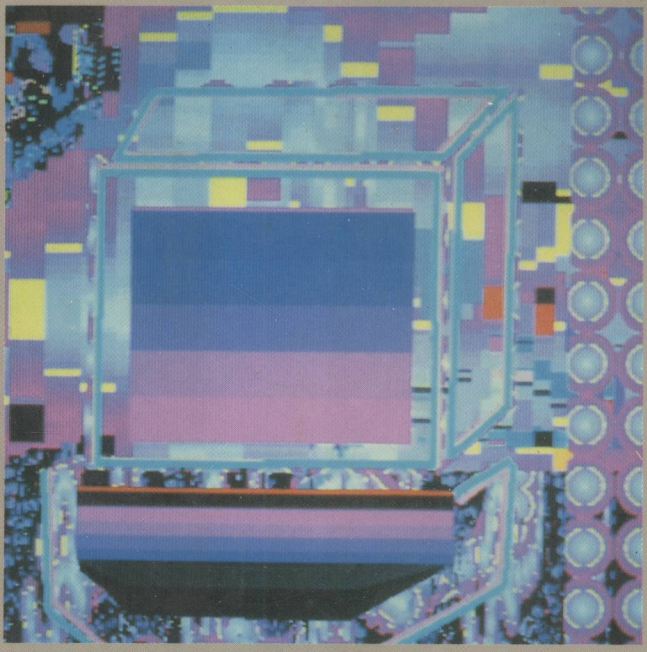
INTERNATIONAL
COMPUTER SCIENCE
SERIES

Logic Programming and Knowledge Engineering

Tore Amble



TP11
A493

8863951

Logic Programming and Knowledge Engineering

Tore Amble

ADDISON-WESLEY
PUBLISHING
COMPANY

Wokingham, England · Reading, Massachusetts · Menlo Park, California
New York · Don Mills, Ontario · Amsterdam · Bonn · Sydney
Singapore · Tokyo · Madrid · Bogota · Santiago · San Juan

© 1987 Addison-Wesley Publishers Ltd.
© 1987 Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the publisher.

The programs in this book have been included for their instructional value. They have been tested with care but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs.

Cover graphic by Laurence M. Gartel.
Typeset by Columns of Reading.
Printed and bound in Great Britain by T J Press (Padstow) Ltd, Cornwall.

First printed 1987.

British Library Cataloguing in Publication Data

Amble, Tore
Logic programming and knowledge engineering.
—(International computer science series).
1. Expert systems (Computer science)
2. Electronic digital computers—
Programming 3. Logic, symbolic and
mathematical
I. Title II. Series
005.13'1 QA76.76.E95

ISBN 0–201–18043–X

Library of Congress Cataloging in Publication Data

Amble, Tore, 1945–
Logic programming and knowledge engineering.

(International computer science series)
Bibliography: p.
Includes index.
1. Logic programming. 2. Expert systems (Computer science) I. Title. II. Series.
QA76.6.A464 1987 005.1 87–13556
ISBN 0–201–18043–X

Logic Programming and Knowledge Engineering

INTERNATIONAL COMPUTER SCIENCE SERIES

Consulting editors **A D McGettrick** University of Strathclyde
J van Leeuwen University of Utrecht

OTHER TITLES IN THE SERIES

Programming in Ada (2nd Edn.) *J G P Barnes*

Software Engineering (2nd Edn.) *I Sommerville*

An Introduction to Numerical Methods with Pascal *L V Atkinson and P J Harley*

The UNIX System *S R Bourne*

Handbook of Algorithms and Data Structures *G H Gonnet*

Microcomputers in Engineering and Science *J F Craine and G R Martin*

UNIX for Super-Users *E Foxley*

Software Specification Techniques *N Gehani and A D McGettrick* (eds.)

Introduction to Expert Systems *P Jackson*

Data Communications for Programmers *M Purser*

Local Area Network Design *A Hopper, S Temple and R C Williamson*

Modula-2: Discipline & Design *A H J Sale*

The UNIX System V Environment *S R Bourne*

Prolog Programming for Artificial Intelligence *I Bratko*

Prolog *F Giannesini, H Kanoui, R Pasero and M van Caneghem*

Programming Language Translation: A Practical Approach *P D Terry*

Data Abstraction in Programming Languages *J M Bishop*

System Simulation: Programming Styles and Languages *W Kreutzer*

The Craft of Software Engineering *A Macro and J Buxton*

UNIX System Programming *K F Haviland and B Salama*

An Introduction to Programming with Modula-2 *P D Terry*

Pop-11 Programming for Artificial Intelligence *A M Burton and N R Shadbolt*

The Specification of Computer Programs *W M Turski and T S E Maibaum*

Software Development with Ada *I Sommerville and R Morrison*

UNIX™ is a trademark of AT & T.

此为试读,需要完整PDF请访问: www.ertongbook.com

Preface

Pioneering research in computer science in the 1970s has led to a revolution in information technology this decade. Two of the most important new directions in software engineering are logic programming and knowledge engineering. They both embody the basic change in computing – a move away from program-controlled computing to computing derived by deduction and reasoning.

The key idea behind logic programming is to replace programming and computation by a logical description of a problem and an automatic proof mechanism for deducing the answers.

The basis of knowledge engineering is to make intelligent, problem-solving expert systems knowledge-based by finding and applying the rules and heuristics that govern experts' problem-solving processes. Knowledge-based systems differ from traditional programs in that they are derived directly from these rules, with no intervening programming.

The combination of these two ideas will change the future of software engineering and replace ideas that have become too formalized, inflexible and problematic. Computer technology will adapt itself to support this development.

This book is written for students and professionals with an interest in engineering, who need a theoretical as well as practical introduction to logic programming and how it can be used to build knowledge-based systems. It is suitable for an undergraduate course at third or fourth year level. For complete understanding, it requires two years of programming experience with some knowledge of Pascal, but parts of the book should be comprehensible to a wider readership.

The book falls roughly into two parts: an introduction to logic programming; and applications of logic programming. The first part introduces predicate logic, resolution and the programming language Prolog. It looks in detail at a subset of the language, list processing in Prolog and advanced programming techniques, with the emphasis on meta-level logic programming. The second part of the book looks at the application of Prolog for formula manipulation, including program verification, and as a sophisticated query language for relational databases. It also describes how to process formal languages, including compiler writing; how to process an interesting and useful subset of natural language; and how to apply knowledge for problem-solving. In addition, it describes an expert system shell in Prolog, with several applications, and gives a short overview of a knowledge engineering project using these ideas.

The appendix contains a collection of useful Prolog predicates.

The language used for the programming examples will be a subset of the Edinburgh Prolog, specifically the C-Prolog VAX version (Pereira, 1984) under the VMS operating systems. The language subset is kept simple, so that the ideas of logic programming are easier to follow. This text, therefore, is not intended to be a reference manual of the fast, user-friendly Prolog system.

Acknowledgements

This book about logic programming and knowledge engineering is a result of 12 years of research and teaching at the University of Trondheim's RUNIT Computing Centre and Division of Computing Science. It is a sequel to an earlier version (Amble, 1984). Many of my colleagues deserve to be mentioned for their help with that version, but I must confine the list to those who have been particularly helpful with the current volume.

First, I am grateful to my colleague Haakon Styri for his encouragement and good advice, based on his impressive knowledge of all aspects of logic programming.

I was fortunate enough to have the opportunity of working on two different projects, both applying logic programming for knowledge engineering. The first was an expert system for diagnosing and teaching television repair; in this field, Roger Eide is an enthusiastic expert and teacher. The other project was to make an expert system for designing welding processes. On this project, I had the pleasure of working with Bjarte H. Nes, who is co-author of the chapter on knowledge engineering. Both projects confirm that the idea of applying logic programming for knowledge engineering works.

Among the people who have scrutinized the manuscripts at various stages, Arild Waaler and Catherine Churchill deserve thanks for their helpful corrections and comments.

A draft version of the book was used for a logic programming course at the University of Trondheim in the fall of 1986. I gained invaluable feedback from the class.

Finally, my bad conscience leads me to express my gratitude towards my wife and children: young children's parents shouldn't really write books.

Figure 13.4 is taken from Hayes-Roth, Waterman and Lenat, *Building Expert Systems*, © 1983, Addison-Wesley Publishing Company Inc., Reading, Massachusetts· (p. 389, Figure A.1), and is reprinted with permission.

8863951

Contents

此为试读,需要完整PDF请访问: www.ertongbook.com

Chapter 1
Introduction

Make rules,
not programs!

1.1 Make specifications, not programs

The first computers were built to alleviate the burden of numerical computation. The computer could with little effort be programmed to perform any sequence of operations, and such was people's confidence in the machines that some engineers predicted that only a few computers would ever be needed for all the world's data processing.

Every day, computer applications once regarded as an unthinkable misuse of costly laboratory instruments are being developed. In fact, so many new applications are appearing that programming itself is losing its status as an art or handicraft, and is becoming more like traditional engineering. To cope with problems in large scale programming, the computer languages that control the computers have become more and more high level and standardized so that programmers can now exchange programs between systems.

To increase the level of programming, users have to specify what a program should do, but not how it should be done. However, as computing machinery became faster and cheaper, economics allowed more details to be automated by advanced computer software.

It is amusing today to read old arguments explaining why the computer should translate mathematical formulae, as in FORTRAN, instead of letting them be programmed more efficiently by skilled assembly language programmers (Backus, 1958). Few programmers today realize that FORTRAN formulae were once regarded as high-level mathematical specifications of a problem, rather than as programs themselves.

The tendency towards making specifications rather than programs is never ending. The ultimate goal would seem to be natural language specifications, therefore. However, natural language is imprecise, ambiguous and sensitive to context, and precision cannot be compromised. A formalism that approximates to precise natural language descriptions is logic. The ideal specification language of the future may well be some

kind of symbolic logic, leaving natural language for communication through the user interface.

Logic programming is the use of logic to define computer programs. The most notable programming language for logic programming is Prolog, an acronym for PROgramming in LOGic. What makes Prolog not just another programming language is its emphasis on the specifications of a problem. Rather than defining the algorithm for solving the problem, Prolog solves problems by systematically searching for a solution. Thus, Prolog is a significant step towards automatic programming.

Since the conception of Prolog in the early 1970s, the number of Prolog programmers has been doubling every year, which has created a market for fast and intelligent implementations of the language.

1.2 Symbolic processing language

Prolog is a language for processing symbolic information, that is, for processing general dynamic symbolic data structures and creating new structures during run-time. This is also a dominant feature of the programming language LISP (Winston and Horn, 1981). In symbolic processing languages, **identifiers** represent *themselves* rather than being names of storage locations as in languages such as Pascal. Prolog is also a **declarative** language. Programmers describe what they know in a precise manner, and leave the rest to the Prolog interpreter.

For example, a programmer wants to say that Harold is the father of Robin. In Pascal, he or she could imagine a linked list of name-records with pointers representing the father relations:

```
TYPE
  person = RECORD
      name : PACKED ARRAY [1..12] OF CHAR;
      father :^person ;
      next :^person ;
    END
var a,b:^person
  .
  .
  .
  a^.name :='HAROLD ';
  b^.name :='ROBIN ';
  b^.father := a ;
  .
  .
  .
```

In addition, separate programs must be written in Pascal before any information can be obtained.

In Prolog, this symbolic information can be expressed much more directly as

```
father(harold,robin),
```

which Prolog accepts as a true statement. If Prolog gets the query

```
?–father(X,robin),
```

it will respond with the answer

```
X=harold;
```

The query

```
?–father(harold,Y).
```

will give the answer

```
Y=robin;
```

while the query

```
?–father(X,Y).
```

will elicit the answer

```
X=harold
Y=robin;
```

1.3 Fifth generation computer systems

Prolog is now a generic name for various languages and dialects around a logic programming paradigm, and has gained great enthusiasm among an increasing number of programmers. This enthusiasm was boosted by the announcement of the Japanese Fifth Generation Computing Systems Programme (Moto-Oka, 1982), which adopted Prolog as the kernel language (Fifth Generation Kernel Language, FGKL) of the new type of computers called knowledge information processing systems (KIPS).

The fifth generation computer project aims to develop a completely new type of computer that can communicate in everyday language, reason intelligently and bring an enormous amount of stored knowledge

to help solve any problem that the user may have. The machines will be cheap and reliable, and so widespread in offices, factories and homes that society itself may be changed by the huge fund of expertise made available.

The fifth generation computer will have three main components:

- an intelligent man machine interface (natural language, written or spoken, graphical);
- problem solving (with logic and statistical reasoning);
- a knowledge base facility able to store and retrieve vast amounts of data, and also give judgements and advice.

The programme's detailed plans promise to make incredibly fast Prolog machines for the 1990s; with a reasoning capacity of up to a billion logic inferences per second (one gigalips), and with associated memory storage of hundreds of gigabytes.

By 1986, the Japanese were already producing high performance Prolog workstations as a tool to accelerate their own development. Needless to say, recent achievements in the Japanese high-technology industries have conditioned us to expect a new success.

In light of this, logic programming, and Prolog in particular, is becoming very important.

1.4 History of logic programming

1.4.1 Aristotelian logic

Logic programming is based on logic, so the right place to start is with the origin of logic, which has its roots with the philosopher Aristotle (382–324 BC). He had an enormous influence on scientific thinking, but some of his ideas did not deserve much reverence. For example, Aristotle said that horses had more teeth than men, without counting them, and that the brain was an organ for cooling the blood, which is only true metaphorically. However, he will be remembered forever for his classic work *Organon*, where he summarized the laws of correct systematic thinking. According to Aristotle, correct reasoning proceeds by the application of strict rules of inference called **syllogisms**. A small example is:

Premise 1: All humans are mortal.
Premise 2: All Greeks are humans.
Premise 3: Socrates is Greek.
Conclusion: Socrates is mortal.

此为试读,需要完整PDF请访问: www.ertongbook.com