Preface

As we approach the fiftieth anniversary of the first programmable computer, the twenty-fifth anniversary of the 'software crisis' is already long past, that expression first having been used at an international conference in 1968. Thus more than half of the history of computer science has been lived under the shadow of our inability to manage the complexity of the artifacts we have created. Under these circumstances, few would dare to suggest that the problems of our discipline have a single technological solution. It is certainly not the purpose of this book to suggest that logic programming, interesting and powerful though it may be, is a panacea for the problems programmers face today.

A more encouraging possibility is that we may be able to find theories and programming paradigms that link together different ways of understanding programs and computer systems. The purpose of this book is to explore to what extent logic programming provides such a theory. Based on predicate logic, it allows computing problems to be expressed in a completely 'declarative' way, without giving instructions for how the problem is to be solved. An execution mechanism, like the one embodied in implementations of Prolog, can then be used to search efficiently and systematically for a solution to the problem. For some problems, the simplest expression of the problem in logical terms also leads to an effective procedure for solving it when a simple execution mechanism is used. Other problems require either a more intelligent execution mechanism, or need to be recast in such a way that a simple execution mechanism can find solutions effectively. Through the medium of logic, we can separate the task of capturing the problem from the task of finding an effective way to solve it.

The implementation of Prolog provides an excellent example of the construction of a software system that satisfies a strong, mathematical specification. In the case of Prolog, this specification is the mathematical meaning that underlies the declarative interpretation of logic programs, and the relevant mathematical foundation is the model theory of Horn clause logic. The thread that links the first part of this book (which presents the mathematical logic behind Prolog) with the last part (which describes how Prolog can be implemented) is this: that the implementation of Prolog can be viewed as carrying out symbolic reasoning with logical formulas, and its correctness is expressed in the fact that it faithfully realizes the inference rule of resolution, which is itself sound with respect to the declarative meaning of programs. The soundness of the resolution rule is established in the first part of the book, and its (almost) faithful implementation in Prolog is explained informally in the last part, but in a way that reflects the structure of a formal development by stepwise (data) refinement.

Another attractive feature of logic programming is the rich web of links it has with other topics in computer science. These are some of the links that are explored in this book:

- Relational databases, stripped of their inessentials, provide operations on relations that are closely linked to ways of combining relations in logic programming. We touch on these links in Chapter 2.
- Mathematical logic, important in formal methods of software development and in artificial intelligence, is also the foundation of logic programming. Studying logic programming is a good introduction to mathematical logic, because the logic behind logic programming is simple, and allows results like the soundness and completeness of inference systems to be proved in the simplest possible setting. In these books, these results are established for the Horn clause logic of Prolog in Chapters 5 to 7.
- Automated theorem proving is increasingly used in the verification of hardware and software systems. It is closely related to logic programming, both because they share some of the same foundations, and because logic programming is a useful vehicle for implementing theorem provers. Some simple applications of logic programming to theorem proving are explored in Chapter 11.
- Type systems for modern programming languages like ML are expressed as systems of inference rules that are in effect logic programs. Compilers for these languages infer types for the expressions in a program by using the same techniques that we shall use to implement Prolog in Chapters 15 to 18.

In a wider sense, every computer system implements a kind of logic. By providing input data, we give the system information about some part of the world. The computer derives some other information which it presents as its output. If the input data is accurate, and the rules we have built into the computer system are sound, then the output data will describe a valid conclusion. Logic programming depends explicitly on this view of computer systems by allowing both the program and its input and output data to be expressed as sentences in formal logic.

Oriel College, Oxford J. M. S.
January, 1996

Using this book

The chapters of this book can be grouped into four parts, each developing different themes from the theory, application and implementation of logic programming. Chapters 1 to 3 introduce the ideas of logic programming; writing programs by defining relations, combining relations to define new ones, recursion in data and programs. The exposition here is mainly by example, and many topics are touched upon that are explored fully in later parts of the book.

Chapters 4 to 8 develop the 'logical' theme by presenting the semantics of logic programs and developing the inference system of SLD–resolution that is the logical basis of Prolog implementations. This is the most mathematical part of the book, and develops in miniature the standard theory of mathematical logic, including proofs that various inference systems for Horn clause logic are sound and complete.

Chapters 9 to 13 present more practical topics, from the formulation of graph-searching problems so that they can be solved by Prolog's simple search strategy, to applications of logic programming in parsing, algebraic simplification and simulating hardware circuits.

The final part of the book, in Chapters 14 to 18, picks up where the second part left off. It explains how SLD–resolution can be implemented efficiently by machine, using the conventional technology of Prolog implementation. These chapters describe the functioning of an actual interpreter for a Prolog subset, and the complete source code for this interpreter is included as Appendix C of this book. The presentation in this part of the book is based on stepwise refinement of data representations. The account begins with a simple implementation of depth-first search that uses abstract data types like sequences, terms and substitutions with corresponding abstract operations. Later chapters explain how these abstract data types can be implemented using the concrete data types provided by a machine.

Getting a copy of picoProlog

A distribution kit is available that contains the Pascal source code of the picoProlog interpreter, code for all the example programs from the book, the '`ppp`' macro processor that is needed to pre-process the picoProlog source and C source code for a Pascal–to–C translator that can be used to compile it via C. The kit is ready-to-build for Sun and Linux machines, and can be ported easily to MS–DOS using either Turbo Pascal directly, or Turbo C and the Pascal–to–C translator.

You can obtain the kit by anonymous FTP from `ftp.comlab.ox.ac.uk` in the directory `/pub/packages/picoProlog`. Teachers adopting the book who have no access to FTP may obtain the distribution kit on floppy disk from the publisher.