Artificial Intelligence through Prolog by Neil C. Rowe

Prentice-Hall, 1988, ISBN 0-13-048679-5

Full text of book (without figures)

Table of contents

Preface

Artificial intelligence is a hard subject to learn. I have written a book to make it easier. I explain difficult concepts in a simple, concrete way. I have organized the material in a new and (I feel) clearer way, a way in which the chapters are in a logical sequence and not just unrelated topics. I believe that with this book, readers can learn the key concepts of artificial intelligence faster and better than with other books. This book is intended for all first courses in artificial intelligence at the undergraduate or graduate level, requiring background of only a few computer science courses. It can also be used on one's own.

Students often complain that while they understand the terminology of artificial intelligence, they don't have a gut feeling for what's going on or how you apply the concepts to a situation. One cause is the complexity of artificial intelligence. Another is the unnecessary baggage, like overly formal logical calculi, that some books and teachers saddle students with. But an equally important cause is the often poor connection made between abstract concepts and their use. So I considered it essential to integrate practical programming examples into this book, in the style of programming language and data structures books. (I stress *practical*, not missionaries and cannibals, definitions of "grandfather", or rules for identifying animals in zoos--at least rarely.) This book has about 500 chunks of code. Clear, concrete formalization of artificial intelligence ideas by programs and program fragments is all the more critical today with commercialization and media discovery of the field, which has caused a good deal of throwing around of artificial intelligence terms by people who don't understand them.

But artificial intelligence is a tool for complex problems, and its program examples can easily be forbiddingly complicated. Books attempting to explain artificial intelligence with examples from the programming language Lisp have repeatedly demonstrated this. But I have come to see that the fault lies more with Lisp than with artificial intelligence. Lisp has been the primary language of artificial intelligence for many years, but it is a low-level language, too low for most students. Designed in the early 1960s, Lisp reflects the then-primitive understanding of good programming, and requires the programmer to worry considerably about actual memory references (pointers). Furthermore, Lisp has a weird, hard-to-read syntax unlike that of any other programming language. To make matters worse, the widespread adoption of Common Lisp as a de facto standard has discouraged research on improved Lisps.

Fortunately there is an alternative: Prolog. Developed in Europe in the 1970s, the language Prolog has steadily gained enthusiastic converts, bolstered by its surprise choice as the initial language of the Japanese Fifth Generation Computer project. Prolog has three positive features that give it key advantages over Lisp. First, Prolog syntax and semantics are much closer to formal logic, the most common way of representing facts and reasoning methods used in the artificial intelligence literature. Second, Prolog provides automatic backtracking, a feature making for considerably easier "search", the most central of all artificial intelligence techniques. Third, Prolog supports multidirectional (or multiuse) reasoning, in which arguments to a procedure can freely be designated inputs and outputs in different

ways in different procedure calls, so that the same procedure definition can be used for many different kinds of reasoning. Besides this, new implementation techniques have given current versions of Prolog close in speed to Lisp implementations, so efficiency is no longer a reason to prefer Lisp.

But Prolog also, I believe, makes teaching artificial intelligence easier. This book is a demonstration. This book is an organic whole, not a random collection of chapters on random topics. My chapters form a steady, logical progression, from knowledge representation to inferences on the representation, to rule-based systems codifying classes of inferences, to search as an abstraction of rule-based systems, to extensions of the methodology, and finally to evaluation of systems. Topics hard to understand like search, the cut predicate, relaxation, and resolution are introduced late and only with careful preparation. In each chapter, details of Prolog are integrated with major concepts of artificial intelligence. For instance, Chapter 2 discusses the kinds of facts about the world that one can put into computers as well as the syntax of Prolog's way; Chapter 3 discusses automatic backtracking as well as Prolog querying; Chapter 4 discusses inference and inheritance as well as the definition of procedures in Prolog; Chapter 5 discusses multidirectional reasoning as well as the syntax of Prolog arithmetic; and so on. This constant tying of theory to practice makes artificial intelligence a lot more concrete. Learning is better motivated since one doesn't need to master a lot of mumbo-jumbo to get to the good stuff. I can't take much of the credit myself: the very nature of Prolog, and particularly the advantages of the last paragraph, make it easy.

Despite my integrated approach to the material, I think I have covered nearly all the topics in ACM and IEEE guidelines for a first course in artificial intelligence. Basic concepts mentioned in those guidelines appear towards the beginning of chapters, and applications mentioned in the guidelines appear towards the ends. Beyond the guidelines however, I have had to make tough decisions about what to leave out--a coherent book is better than an incoherent book that covers everything. Since this is a first course, I concentrate on the hard core of artificial intelligence. So I don't discuss much how humans think (that's psychology), or how human language works (that's linguistics), or how sensor interpretation and low-level visual processing are done (that's pattern recognition), or whether computers will ever really think (that's philosophy). I have also cut corners on hard non-central topics like computer learning and the full formal development of predicate calculus. On the other hand, I emphasize more than other books do the central computer science concepts of procedure calls, variable binding, list processing, tree traversal, analysis of processing efficiency, compilation, caching, and recursion. This is a computer science textbook.

A disadvantage of my integrated approach is that chapters can't so easily be skipped. To partially compensate, I mark some sections within chapters (usually sections towards the end) with asterisks to indicate that they are optional to the main flow of the book. In addition, all of Chapters 7, 10, and 14 can be omitted, and perhaps Chapters 12 and 13 too. (Chapters 7, 10, 13, and 14 provide a good basis for a second course in artificial intelligence, and I have used them that way myself.) Besides this, I cater to the different needs of different readers in the exercises. Exercises are essential to learning the material in a textbook. Unfortunately, there is little consensus about what kind of exercises to give for courses in artificial intelligence. So I have provided a wide variety: short-answer questions for checking basic understanding of material, programming exercises for people who like to program, "play computer"

exercises that have the reader simulate techniques described, application questions that have the reader apply methods to new areas (my favorite kind of exercise because it tests real understanding of the material), essay questions, fallacies to analyze, complexity analysis questions, and a few extended projects suitable for teams of students. There are also some miscellaneous questions drawing on the entire book, at the end of Chapter 15. Answers to about one third of the exercises are provided in Appendix G, to offer readers immediate feedback on their understanding, something especially important to those tackling this book on their own.

To make learning the difficult material of this book even easier, I provide other learning aids. I apportion the book into short labeled sections, to make it easier for readers to chunk the material into mind-sized bites. I provide reinforcement of key concepts with some novel graphical and tabular displays. I provide "glass box" computer programs (that is, the opposite of "black box") for readers to study. I mark key terms in boldface where they are defined in the text, and then group these terms into keyword lists at the end of every chapter. I give appendices summarizing the important background material needed for this book, concepts in logic, recursion, and data structures. In other appendices, I summarize the Prolog dialect of the book, make a few comments on Micro-Prolog, and provide a short bibliography (most of the artificial intelligence literature is now either too hard or too easy for readers of this book). The major programs of the book are available on tape from the publisher for a small fee. Also, I have prepared an instructor's manual.

It's not necessary to have a Prolog interpreter or compiler available to use this book, but it does make learning easier. This book uses a limited subset of the most common dialect of Prolog, the "standard Prolog" of *Programming in Prolog* by Clocksin and Mellish (second edition, Springer-Verlag, 1984). But most exercises do not require programming.

I've tried to doublecheck all examples, programs, and exercises, but some errors may have escaped me. If you find any, please write me in care of the publisher, or send computer mail to rowe@nps-cs.arpa.

Acknowledgements

Many people contributed ideas to this book. Michael Genesereth first suggested to me the teaching of introductory artificial intelligence in a way based on logic. David H. Warren gradually eroded my skepticism about Prolog. Harold Abelson and Seymour Papert have steered my teaching style towards student activity rather than passivity.

Judy Quesenberry spent many long hours helping me with the typing and correction of this book, and deserves a great deal of thanks, even if she ate an awful lot of my cookies. Robert Richbourg has been helpful in many different ways, in suggesting corrections and improvements and in testing out some of the programs, despite his having to jump through all the hoops Ph.D. students must jump through. Richard Hamming provided valuable advice on book production. Other people who provided valuable

comments include Chris Carlson, Daniel Chester, Ernest Davis, Eileen Entin, Robert Grant, Mike Goyden, Kirk Jennings, Grace Mason, Bruce MacLennan, Norman McNeal, Bob McGhee, James Milojkovic, Jim Peak, Olen Porter, Brian Rodeck, Derek Sleeman, Amnon Shefi, and Steve Weingart. Mycke Moore made the creative suggestion that I put a lot of sex into this book to boost sales.

Besides those named, I am grateful to all my students over the years at the Massachusetts Institute of Technology, Stanford University, and the Naval Postgraduate School for providing valuable feedback. They deserve a good deal of credit for the quality of this book--but sorry, people, I'm poor and unwilling to share royalties.

To the reader

Artificial intelligence draws on many different areas of computer science. It is hard to recommend prerequisites because what you need to know is bits and pieces scattered over many different courses. At least two quarters or semesters of computer programming in a higher-level language like Pascal is strongly recommended, since we will introduce here a programming language several degrees more difficult, Prolog. If you can get programming experience in Prolog, Lisp, or Logo that's even better. It also helps to have a course in formal logic, though we won't use much of the fancy stuff they usually cover in those courses; see Appendix A for what you do need to know. Artificial intelligence uses sophisticated data structures, so a data structures course helps; see Appendix C for a summary. Finally, you should be familiar with recursion, because Prolog is well suited to this way of writing programs. Recursion is a difficult concept to understand at first, but once you get used to it you will find it easy and natural; Appendix B provides some hints.

Solving problems is the best way to learn artificial intelligence. So there are lots of exercises in this book, at the ends of chapters. Please take these exercises seriously; many of them are hard, but you can really learn from them, much more than by just passively reading the text. Artificial intelligence is difficult to learn, and feedback really helps, especially if you're working on your own. (But don't plan to do all the exercises: there are too many.) Exercises have code letters to indicate their special features:

--the code R means a particularly good problem recommended for all readers;
--the code A means a question that has an answer in Appendix G;
--the code H means a particularly hard problem;
--the code P means a problem requiring actual programming in Prolog;
--the code E means an essay question;
--the code G means a good group project.

In addition to exercises, each chapter has a list of key terms you should know. Think of this list, at the end of the text for each chapter, as a set of "review questions".

The symbol "*" on a section of a chapter means optional reading. These sections are either significantly harder than the rest of the text, or significantly far from the core material.

Go to book index

Introduction

What artificial intelligence is about

Artificial intelligence is the getting of computers to do things that seem to be intelligent. The hope is that more intelligent computers can be more helpful to us--better able to respond to our needs and wants, and more clever about satisfying them.

But "intelligence" is a vague word. So artificial intelligence is not a well-defined field. One thing it often means is advanced software engineering, sophisticated software techniques for hard problems that can't be solved in any easy way. Another thing it often means is nonnumeric ways of solving problems, since people can't handle numbers well. Nonnumeric ways are often "common sense" ways, not necessarily the best ones. So artificial-intelligence programs--like people--are usually not perfect, and even make mistakes.

Artificial intelligence includes:

--Getting computers to communicate with us in human languages like English, either by printing on a computer terminal, understanding things we type on a computer terminal, generating speech, or understanding our speech (*natural language*);

--Getting computers to remember complicated interrelated facts, and draw conclusions from them (*inference*);

--Getting computers to plan sequences of actions to accomplish goals (*planning*);

--Getting computers to offer us advice based on complicated rules for various situations (*expert systems*);

--Getting computers to look through cameras and see what's there (*vision*);

--Getting computers to move themselves and objects around in the real world (*robotics*).

We'll emphasize inference, planning, and expert systems in this book because they're the "hard core" of artificial intelligence; the other three subareas are getting quite specialized, though we'll mention them too from time to time. All six subareas are hard; significant progress in any will require years of research. But we've already had enough progress to get some useful programs. These programs have created much interest, and have stimulated recent growth of the field.

Success is hard to measure, though. Perhaps the key issue in artificial intelligence is *reductionism*, the degree to which a program fails to reflect the full complexity of human beings. Reductionism includes how often program behavior duplicates human behavior and how much it differs when it does differ. Reductionism is partly a moral issue because it requires moral judgments. Reductionism is also a social issue because it relates to automation.

Understanding artificial intelligence

Artificial intelligence techniques and ideas seem to be harder to understand than most things in computer science, and we give you fair warning. For one thing, there are lots of details to worry about. Artificial intelligence shows best on complex problems for which general principles don't help much, though there are a few useful general principles that we'll explain in this book. This means many examples in this book are several pages long, unlike most of the examples in mathematics textbooks.

Complexity limits how much the programmer can understand about what is going on in an artificial-intelligence program. Often the programs are like simulations: the programmer sets conditions on the behavior of the program, but doesn't know what will happen once it starts. This means a different style of programming than with traditional higher-level languages like Fortran, Pascal, PL/1, and Ada | REFERENCE 1|, .FS | REFERENCE 1| A trademark of the U.S. Department of Defense, Ada Joint Program Office. .FE where successive refinement of a specification can mean we know what the program is doing at every level of detail. But artificial-intelligence techniques, even when all their details are hard to follow, are often the only way to solve a difficult problem.

Artificial intelligence is also difficult to understand by its content, a funny mixture of the rigorous and the unrigorous. Certain topics are just questions of style (like much of Chapters 2, 6, and 12), while other topics have definite rights and wrongs (like much of Chapters 3, 5, and 11). Artificial intelligence researchers frequently argue about style, but publish more papers about the other topics. And when rigor is present, it's often different from that in the other sciences and engineering: it's not numeric but *logical*, in terms of truth and implication.

Clarke's Law says that all unexplained advanced technology is like magic. So artificial intelligence may lose its magic as you come to understand it. Don't be discouraged. Remember, genius is 5% inspiration and 95% perspiration according to the best figures, though estimates vary.

Preview

This book is organized around the important central ideas of artificial intelligence rather than around application areas. We start out (Chapters 2-5) by explaining ways of storing and using knowledge inside computers: facts (Chapter 2), queries (Chapter 3), rules (Chapter 4), and numbers and lists (Chapter 5). We examine rule-based systems in Chapters 6-8, an extremely important subclass of artificial intelligence programs. We examine search techniques in Chapters 9-11, another important subclass. We

address other important topics in Chapters 12-14: Chapter 12 on frame representations extends Chapter 2, Chapter 13 on long queries extends Chapter 3, and Chapter 14 on general logical reasoning extends Chapter 4. We conclude in Chapter 15 with a look at evaluation and debugging of artificial intelligence programs; that chapter is recommended for everyone, even those who haven't read all the other chapters. To help you, appendices A-C review material on logic, recursion, and data structures respectively. Appendix D summarizes the Prolog language subset we use in this book, Appendix E summarizes the Micro-Prolog dialect, Appendix F gives a short bibliography, and Appendix G provides answers to some of the exercises.

Keywords:

artificial intelligence
natural language
inference
planning
expert systems
vision
robotics
reductionism

Go to book index

Representing facts

If we want computers to act intelligent, we must help them. We must tell them all the common-sense *knowledge* we have that they don't. This can be hard because this knowledge can be so obvious to us that we don't realize that a computer doesn't know it too, but we must try.

Now there are many different kinds of knowledge. Without getting deep into philosophy (or specifically *epistemology*, the theory of knowledge), there are two main kinds: facts and reasoning procedures. Facts are things true about the world, and reasoning procedures (or *inferences*) are ways to follow reasoning chains between facts. Since facts are easier to represent than procedures, we'll consider them first, and postpone procedures to Chapter 4.

Predicates and predicate expressions

To talk about facts we need a "language". Artificial intelligence uses many languages and sub-languages. But in this introductory book we don't want to confuse you. We'll use only one, simple (*first-order*) *predicate logic* (sometimes called *predicate calculus* and sometimes just *logic*). And we'll use a particular notation compatible with the computer programming language Prolog | REFERENCE 1|. .FS | REFERENCE 1| In this book we use a subset of the "standard Prolog" in Clocksin and Mellish, *Programming in Prolog*, second edition, Springer-Verlag, 1984. For a complete description of what we use, see Appendix D. .FE Prolog isn't predicate logic itself; computer languages try to do things, whereas logic just says that certain things are true and false. But Prolog does appear close to the way logic is usually written. That is, its *grammar* or *syntax* or form is that of logic, but its *semantics* or meaning is different.

And what is that grammar? Formally, a *predicate expression* (or *atomic formula*, but that sounds like a nuclear weapons secret) is a name--a *predicate*--followed by zero or more arguments enclosed in parentheses and separated by commas (see Figure 2-1) | REFERENCE 2|. .FS | REFERENCE 2| Several terms closely related to "predicate expression" are used in the logic and artificial-intelligence literature. A *literal* is like a predicate expression only it can have a negation symbol in front of it (negations will be explained in Section 3.6). A *structure* or *compound term* is like a predicate expression only it isn't necessarily only true or false. A *logical formula* is a structure or a set of structures put together with "and"s, "or"s, and "not"s. .FE Predicate names and arguments can be composed of any mixture of letters and numbers, except that predicate names must start with a lower-case letter. (Upper-case letters first in a word have a special meaning in Prolog, as we'll explain shortly.) The underscore symbol "_" also counts as a letter, and we will often use it to make names more readable. So these are all predicate expressions:

```
p(x)
q(y,3)
```

```
r(alpha,-2584,beta)
city(monterey,california)
tuvwxy(abc345)
noarguments
pi(3.1416)
long_predicate_name(long_argument_name,3)
```

We can put predicate expressions like these into computers. They can represent facts true about the world. But what exactly do these expressions *mean* (their *semantics*)? Actually, anything you want--it's up to you to assign reasonable and consistent interpretations to the symbols and the way they're put together, though there are some conventions. The better job you do, the more reasonable the conclusions you'll reach from all these facts.

Predicates indicating types

Predicates can mean many things. But they do fall into categories. We summarize the major categories in Figure 2-2.

One thing they can mean is something like data-type information in a language like Pascal or Ada. Except that in artificial intelligence there are generally a lot more types than there are in most programming, because there must be a type for every category in the world that we want the computer to know about.

For instance, suppose we want the computer to know about some U.S. Navy ships | REFERENCE 3|. We .FS | REFERENCE 3| The occasional use of military examples in this book is deliberate: to serve as a reminder that much artificial intelligence work in the United States has been, and remains, supported by the military. We make no endorsements. .FE could tell it

```
ship(enterprise).
```

to say that the Enterprise is a ship (remember we must use lower case). Or in other words, the Enterprise is an example of the "ship" type. We will put periods at the end of facts because Prolog uses the period to signal the end of a line. We could also tell the computer

```
ship(kennedy).
ship(vinson).
```

to give it the names of two more ships--two more things of the "ship" type. Here **ship** is a *type predicate*. If we knew code numbers for planes we could tell the computer about them too, using the code numbers as names:

```
plane(p54862).
```

```
plane(p79313).
```

Similarly, we can label people with types:

```
commodore(r_h_shumaker).
president(r_reagan).
```

and label more abstract things like institutions:

```
university(naval_postgraduate_school).
university(stanford_university).
```

and label concepts:

```
day_of_week(monday).
day_of_week(tuesday).
day_of_week(wednesday).
```

A thing can have more than one type. For instance:

```
ship(enterprise).
american(enterprise).
```

And types can have subtypes:

```
carrier(vinson).
ship(carrier).
```

These are all *type predicates*, and they are all have one argument. The argument is the name of some thing in the world, and the predicate name is the class or category it belongs to. So the predicate name is *more general* than the argument name; this is usual for predicate names in artificial intelligence. So it wouldn't be as good to say

```
enterprise(ship).
kennedy(ship).
```

About types

We've said these predicates are like the types in computer languages, but there are some differences. The main one is that they need never be defined anywhere. If for instance we are using Pascal, we either use the built-in types (integer, real, character, array, and pointer) or define the type we want in terms of those