The Knowledge Engineering Review, Vol. 4: 1, 1989, 73–85

Book Reviews

Prolog for programmers reviewed by Ken Johnson, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh EH1 1HN.
Networks and distributed computation: Concepts, tools and algorithms reviewed by Raiju Trehan, Department of Artificial Intelligence, University of Edinburgh, Edinburgh EH1 1HN.
Natural language understanding and logic programming, II reviewed by John Beaven, Department of Artificial Intelligence, University of Edinburgh, Edinburgh EH1 1HN.
Expert systems development in Prolog and Turbo-Prolog reviewed by Ray Lai, Department of Artificial Intelligence, University of Edinburgh, Edinburgh EH1 1HN.
Database applications using prolog reviewed by Dr Paul Soper, Department of Electronics and Computer Science, University of Southampton, Southampton SO9 5NH.
The rise of the expert company reviewed by (1) Dr Rod N Cuff, IBM (UK) Scientific Centre, Winchester, Hampshire SO23 9DR. (2) Professor Geoffrey Trimble, Department of Civil Engineering, University of Technology, Loughborough, Leicestershire LE11 3TU. (3) Professor Peter Hammond, Applied Computing Centre for Computing and Computer Science, University of Birmingham, Birmingham B15 2TT. (4) Peter Russo, Advanced Technology Center, Boeing Computer Service 7L-64, PO Box 24346, Seattle, WA 98124-0346, USA.

Prolog for programmers by Feliks Kluzniak and Stanislaw Szpakowicz, with a contribution by Janusz S. Bien, Academic Press, 1985, reprinted 1987, Paperback edition, ISBN 0–12–416521–4, 308 pages including many diagrams and listings. Includes Toy-Prolog for IBM-PC on $5\frac{1}{4}$ inch floppy disk. £14.95.

Pascal is historic; Lisp is expressive; Prolog is exciting. In the few years since this book was first published, Prolog has consolidated its position as a powerful symbolic programming language for the Artificial Intelligence community, with its roots in automatic theorem proving and formal logic. Over the years, Prolog has gradually acquired a folklore, as neat tricks like stored lemmas, difference lists, failure driven loops and so on have been passed from one programmer in the language to another. The unique nature of Prolog means that Prolog's folklore has little in common with those of Lisp or Pascal. Anyone coming new to Prolog from Basic, Fortran, Pascal etc, experiences the excitement of discovering how the language works. The shock of watching a two- or three-line Prolog predicate do work that would take as many pages of Pascal feels like watching a magician pull a rabbitt out of an empty top hat.

It is the element of magic that is missing from the text. This book attempts both to teach a number of the more useful Prolog programming tricks and also to expound on the background to the development of the language. But the teaching is very formal, heavy going, and definitely unsuitable as a first reader in the language.

The authors implicitly presume that you know enough Lisp and Pascal to understand terms like *conscarcdr* and *record* without explanation. Indeed, to get the most from the section on implementation you need to have a good knowledge of Pascal. But apart from the long source listings, comparison with Pascal is certain to mislead and confuse: Prolog is completely different in its approach to problem solving.

There is a lot of obtrusive use of recondite notations: the authors oftern write *a.b.c*[] for the Prolog list [*a*,*b*,*c*] and restating Prolog code in the notation of formal logic; for example, of an insertion sort they state

$$L = X + + [A,B \mid Y] \wedge A > B \wedge LT = X + + [B,A \mid Y]$$
$$\Rightarrow sorted(L) = sorted(LT)$$

which is doubtless true, but the Prolog code itself is actually more straightforward to read.

The explanation of the Resolution method of theorem proving in chapter 2 is intrinsically interesting, but needs to be supplemented with other information on the historical development of the language; until you find this history in chapter 9, you wonder what the point of including this chapter is. It would still have been possible to show to what extent the histories of Resolution theorem proving, definite clause grammar and logic programming have contributed to making Prolog what it is.

The section of the book dealing with Prolog programming techniques is well worth reading. There is good coverage of trees and lists, difference lists and Pereira and Porto's strikingly simple map-colouring algorithm. I felt that the constant use of formalisms (I have quoted one of those given for insertion sort) generally did not help the development of the topics much. Again, it is a pity the discussions are couched in language too formal to show surprise, delight or joy. There is a long section describing metamorphosis grammars (better known as definite clause grammars). As with other parts of the book, by starting at a low level of detail and meticulously working up to a higher level, instead of giving a couple of examples first and then moving on to an examination of how the trick works.

There are two long programming projects in the book, one a version of David Warren's *WARPLAN*; the other looks at Prolog and relational data bases. These sections are also essential reading—but not compelling reading.

With the book comes a floppy disk carrying **Toy-Prolog** and its libraries and sources in Pascal. The Pascal code is supplied for pedagogical reasons; Toy-Prolog is slow but complete and it works well. If you do not have a Prolog system then Toy-Prolog will sufice to give you a feel for what using the language feels like, but it is enormously slower than the commercial implementations. If you have access to a commercial Prolog, use it and refer to the source when reading the relevant chapters.

The book spends a lot of time explaining the source, since taking advantage of the way an implementation works can help the programmer to design Prolog programs that run faster. The internal Prolog data structures are made clear with lots of box-and-pointer diagrams.

My feeling about the audience for this book is that, in general, someone beginning to learn to write programs in Prolog should look elsewhere, though the experienced Prolog programmer will see here techniques he may not have seen elsewhere. The explanation of Resolution and the historical notes will interest students of A.I., and computer scientists will appreciate the implementation details.

Networks and distributed computation: Concepts, tools and algorithms by Michal Raynal, North Oxford Academic Press Publs Ltd, 1987, 166 pp, £25.00.

Multiprocessor configurations hold much promise for future computing. These architectures may provide additional speed in cases where the computation can be partitioned either in terms of function or data. Another feature is that for some problems these architectures may provide a model of computation which provides a natural way to view the problem, hence providing a better mapping from problem to architecture. When dealing with multiprocessor architectures several issues arise, for example: how processes will communicate; how communication will be synchronised; how a problem will be partitioned into tasks; how tasks can be distributed; the topology of the processes.

This book addresses these issues in a somewhat formal manner. The text is split into three parts. The first introduces and considers in detail two example problems, a data transfer protocol and the issue of mutual dependence of logical clocks. These examples highlight issues of cooperation in different tasks and control of identical tasks. The second part considers how the system can gain information about the topology and state of the processor network. This provides a basis by which tasks can be distributed and controlled. The third and final part considers how constraints can be