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

defined over the state of the system and then presents some methods for controlling and managing distributed processes.

This book considers some of the fundamental issues involved in distributed computation. As such it could provide a core book of the field. The book is targeted at final year undergraduate and postgraduate students. However, the book has one major flaw, it is badly written or may be badly translated (the original being in French). In short, this book contains a great deal of information but the presentation of this information is obscured by the style in which the book is written. I cannot recommend it.

Natural language understanding and logic programming, II. Proceedings of the Second International Workshop on Natural Language Understanding and Logic Programming, Vancouver, Canada, 17–19 August 1987, Elsevier Science Publications, The Netherlands, June 1988, pp 346, Approx $84.25, ISBN 0-444-70408-6.

Natural Language (NL) parsing has been one of the main applications of Prolog since the language was developed in the early 1970s, and many logic based formalisms have been designed based on Logic Programming, such as Colmerauer's Metamorphosis Grammars, Pereira and Warren's Definite Clause Grammars, and many other Logic Grammars.

One of key ideas at the root of using Logic Programming for NL parsing consists of viewing parsing as theorem proving in a logic in which the lexicon and the grammar are interpreted as axioms and deduction rules. The Prolog language, as a theorem prover for Horn clauses, can easily be used as a parser if the grammar rules are written as Horn clauses. To take a simple example, we re-interpret the usual phrase structure rule that states that a Sentence rewrites as a Noun Phrase (NP) followed by a Verb Phrase (VP), and say instead that in order to prove that a given string is a sentence, we have to prove that it is the concatenation of two strings, the first of which is a Noun Phrase, and the second is a VP. Computationally efficient algorithms for theorem proving can then be used to parse NL.

More recently, this last decade has seen the appearance of a large number of unification-based grammars (i.e. those which use the logical operation of unification to pass information between linguistic objects), together with formalisms in which to express these, such as Shieber's PATR-II.

The general aim is to replace traditional transformational accounts of grammar (whose transformations are widely held to be computationally unmanageable), with one-level representations in which phonology, syntax and semantics have a similar status, and are assembled together at the same time. This would help to gain a better understanding of the relation between these aspects of NL.

In the past, much emphasis has been put on NL syntax and parsing, perhaps because of the unavailability of large and efficient Prolog systems. But formal logic has also an important part to play in the semantics of NL. However, it is generally acknowledged that first order logic, on which Prolog is based, is not powerful enough for NL applications, and many extensions have been proposed to cover these deficiencies.

This book may be seen as an attempt to redress the balance away from syntax and parsing. It presents 18 high quality papers, and two panel discussions, by many leading researchers in the field, covering such wide-ranging topics as syntax, parsing, generation, semantics, discourse, pragmatics and cooperative responses.

Broadly speaking, the papers on morphology and syntax aim to show how logic programming formalisms are capable of expressing well-known phenomena succinctly and effectively. The contributions on parsing and on new formalisms show how to express grammatical constraints in a declarative and, it is hoped, computationally tractable way. Those on semantics generally aim at showing how first-order logic is insufficient and/or clumsy for expressing NL semantics, and discuss ways of enriching logic programming languages in order to handle NL effectively. Finally, a group of