14. J. M. Morris, Laws of data refinement. *Acta Informatica*, **26**, 287–308 (1989).
15. J. M. Morris, A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, **9** (3), 298–306 (1987).
16. E. Sekerinski, A calculus for predicative programming. *Proceedings of the 2nd International Conference on the Mathematics of Program Construction, Oxford, June 1992.* To be published in Lecture Notes in Computer Science, Springer Verlag, Heidelberg (1992).
17. J. M. Spivey, *The Z Notation: a Reference Manual.* Prentice-Hall International, London (1989).
18. J. C. P. Woodcock, A tutorial on the refinement calculus. In *VDM'91: Formal Software Development Methods.* Lecture Notes in Computer Science 552, pp. 79–140. Springer-Verlag, Heidelberg (1991).

Book Reviews

Y. TAKEFUJI. *Neural Network Parallel Computing*. Kluwer Academic Publishers Group, Dordrecht, 1992, £44.25 ISBN 0-7923-9190-X

In his ackowledgements Takefuji admits to being inspired by the work of Hopfield and Tank on using neural network architectures to solve problems in optimisation. The book sets out many extensions to the approach where mathematical models of constraint satisfaction problems are implemented and tested in neural nets. The variety of topics that are addressed and the variety of methods used to convey the ideas are distinctive characteristics.

The first ten chapters deal with many hard optimisation problems such as N-queen problems and k-colorability problems. The remaining chapters deal with hardware implementations and mathematical derivations. Some Turbo PASCAL code is provided and each chapter has its own list of references. Most of the chapters have student exercises attached. The exercises are clearly an attempt to make this a text book, but I feel that students generally benefit most when model answers are supplied. However, after a cursory glance, I did wonder just how the average undergraduate might cope with Exercise 11.5 No. 4 – 'Survey research on silicon neural network implementations and optical implementations'.

Takefuji's general approach is to provide various gradient descent methods for solving constraint satisfaction problems. The aim is to construct something known as a motion equation which species a 'fabricated computational energy function' (p. 4). An artificial neural net is then developed to implement parallel gradient descent as a method to minimise the fabricated energy function. Various types of artificial neurons are considered, and each is defined relative to a different input/output function. Takefuji discusses the basic McCulloch and Pitts neuron defined as a binary threshold logic unit (TLU). An alternative to this is a unit using a sigmoid input/output function as studied by Hopfield. Nets comprising either sort of unit are then examined. Detailed comparisons revealed that the TLU net tended to converge faster than the sigmoid net, although the TLU net did exhibit unfortunate oscillatory behaviour. To overcome this, Takefuji discusses using units that employ an hysteresis McCulloch–Pitts input/output function. With this, the idea is to have a unit with essentially two thresholds. The upper threshold sets a value above which inputs must be for the unit to turn on, the lower threshold sets a value below which the unit turns off. Input values between the upper and lower bounds have no effect on the unit. Nets with these units no longer exhibited oscillation, but problems remained over how best to set the threshold values.

Although Takefuji discusses Boltzmann machines in passing, his favoured model is one known as the 'maximum neuron model'. This is a variant on the theme of a winner-take-all net. One of the advantages of these nets is that they are 'guaranteed to generate satisfactory solutions' (p. 181). Another is that 'turing coefficients parameters in the motion equation is not required'. From this last statement I am happy to conclude either that my rather slender grasp of mathematics lets me down, or that something has gone wrong in the typesetting.

Nevertheless, my general impression was that the book has been cobbled together. Some of it is rather too obviously the product of 'cut-and-paste', there are inconsistencies in style (compare the formatting of the references for Chapters 1 and 13), and some of the writing is dreadful. Moreover, the formula typesetting is untidy. This can be off-putting to those who find unpacking mathematical expressions into natural language daunting. Equation 1.8 (p. 9) is a case in point (see also the proof on p. 191).

This is truly a book for computer scientists with a strong background in mathematics. Although a wide range of topics is covered – from natural brains to VLSI chips – the book is perhaps going to fit most comfortably in the hands of applied mathematicians.

P. QUINLAN
York

L. C. PAULSONS. *ML for the Working Programmer*, Cambridge University Press. £27.50. ISBN 0 521 39022 2

Standard ML (henceforth referred to as SML) is a major influence in the design of programming languages. It is widely used in the research community and is increasingly used for teaching computer science. It is even beginning to find its way into the commercial world, both as a prototyping tool and as a delivery language. Therefore there is a significant demand for a good introductory text. Paulson's book, while not perfect, meets that demand well.

As the title suggests, Paulson aims his book at people who already know how to program and who want to use SML on real programs. His book is also suitable for advanced undergraduate teaching.

Paulson introduces the features of SML by example. His explanations are generally clear and form a good introduction to both functional and imperative programming with SML. Most of his examples are based on code that he has used himself, rather than purely illustrative code from the classroom. So the reader is shown how to write basic tools such as binary trees, priority queues, tree searches, parsers and pretty-printers in a functional style. Some 'lazy' data structures are also examined in detail. Paulson also gives syntax diagrams at the back of the book, which newcomers to SML will find useful.

Other examples introduce the basic ideas of first-order classical logic and the lambda-calculus, which are used in two major case studies once the presentation of the language is complete. Obviously these examples aren't directly useful to programmers from other fields, but Paulson is on home ground here, and presents his examples well. The code for many of the examples in the book is available by anonymous FTP.

There is a particularly strong need for a textbook that explains how to use the SML modules system. Paulson does a good job of presenting the basics. There is more that he could have said, but he shows enough for people to use the language to build real programs. Similarly, although I disagree with his discussion of abstract types in SML, his presentation is good enough for people to get things done.

Paulson also offers a chapter on formal reasoning about functional programs. He cover the ground well, and discusses both the limitations and the virtues of the techniques. However, I felt that he would have done better to integrate this discussion with the main text. Putting it in a separate chapter may make it rather indigestible to the 'working programmer' of the title.

Working programmers need efficient programs, and Paulson does discuss the efficiency of his examples. However, he doesn't present any techniques for analysing the efficiency of recursive programs. Perhaps surprisingly for a book aimed at imperative programmers, he also doesn't deal directly with questions such as 'How do I write a loop in functional language?'. Although the techniques are described, readers are left to find them for themselves.

Overall, I like the book. The treatment of the core language is very good – I could quibble with details, but these are mainly matters of personal taste and style. I have more disagreements with his treatment of modules, and he leaves room for a more comprehensive coverage of the modules system, but his presentation is both adequate in itself and better than the competition. Apart from this, my main criticism is that there is not enough discussion of when to use which features of the language. However, the examples provide a useful guide. In my opinion, this is the best general SML textbook currently available.

DAVE BERRY
Edinburgh