therefore recommend the book to someone with a background and interest in theorem proving techniques, especially related to algebraic specification.

Without detracting from the merits of this book, let me end with a tribute to Donald Knuth: we have got so used to seeing TeX-quality mathematical typesetting, that one wishes the publishers of this book had also typeset it in TeX.

K. LODAYA
Institute of Mathematical Sciences, Madras

RACHEL HARRISON
Abstract Data Types in Standard ML. Wiley, 1993, £19.95, 212 pp softbound, ISBN 0-471-93844-0

The notion of abstraction is a vital one in software engineering—addressing the different phases of software development at the appropriate level of detail is a precondition for the construction of high-quality, reusable software. This book tackles the issue of data abstraction in a functional framework, using the language Standard ML (SML) as the implementation language.

In keeping with conventional texts on abstract data types, the book looks at the normal range of data structures such as lists, stacks, queues and trees. Each data structure is specified informally in terms of pre- and post-conditions and implemented generically using SML's structures. Throughout the book the twin goals of correctness and modularity are emphasised.

The book is not intended as an introductory text to functional programming nor to SML. Herein lies its principle problem—the amount of knowledge assumed on the part of the reader is not clear. The preface states that some familiarity with functional programming is assumed; however on opposite pages we find the term 'head of list' defined whereas curried functions are used without explanation—it is difficult to imagine a reader who understands the latter without already understanding the former! Other minor quibbles are that the use of pre/post specifications is at times inconsistent (varying from vague type requirements to precise relationships between function arguments) and the inexplicable lack of use of SML's exception handling mechanism (using instead hd □ to raise *every* exception).

These points aside, this book has admirable objectives and, by and large, achieves them. The advantage of the functional approach is that the relationship between specifications and implementations is clear, unlike in conventional treatments where this relationship is easily lost in the mire of code. For those students of software engineering with a good grasp of functional programming, who wish to learn the rudiments of data abstraction, this is an excellent text.

P. MUKHERJEE
University of Birmingham

COLIN MYERS, CHRIS CLACK AND ELLEN POON
Programming with Standard ML. Prentice-Hall, 1993, £19.95, 301 pp softbound, ISBN 0-13-722075-8

This book is intended as an introduction to functional programming using a functional subset of Standard ML. It is aimed at those with little programming experience as well as more experienced programmers who wish to learn about functional programming. It intentionally differs from other texts on the same subject by avoiding mathematics and mathematical reasoning whenever possible, instead concentrating on more practical programming issues.

The book begins by introducing fundamental functional programming concepts such as expressions, types and referential transparency. From there, the notion of a function is introduced. Simple examples of functions are presented and the ideas of recursion and polymorphism are described. The principle of allotting a single purpose to each function when decomposing a problem is continually stressed, both explicitly and by example.

The third chapter is a long chapter on lists. The notion of a list is introduced, functions over lists are described and different kinds of recursion over lists are discussed. An extended example is presented—a function similar to the Unix *grep* command—to demonstrate that 'real' software can be written in a functional programming language. The remaining distinguishing property of functional programming, namely curried and higher order functions, is described in chapter 4, having been motivated by the introduction of lists. Standard functions such as *map, foldl* and *foldr* are introduced.

Chapters 5–7 give a detailed description of the features available in Standard ML, based on the ideas of local definitions, encapsulation, information hiding and modularity. Thus these chapters are concerned with more practical issues than the preceding chapters which attempt to teach the principles of sound program design in a functional style.

A consequence of the intentional absence of mathematics is that there is no mention of proving properties of programs. In particular, correctness is not treated at all. Related ideas such as validation of functions are similarly absent. These aspects of software construction are hardly dry theoretical issues; on the contrary, they are vital for software construction to be considered as an engineering discipline. Moreover, this is one area where functional programming languages score heavily over imperative languages because referential transparency greatly eases reasoning. Even if a detailed treatment of program proof and correctness was undesirable, a brief discussion of the merits of correctness by construction would have been sufficient.

The book also occasionally runs into problems due to its stated ambition of appealing to both those with little or no programming experience and experienced programmers wishing to learn a functional language. For instance, the relative simplicity of the functional