J. Functional Programming **7** (4): 447–449, July 1997. Printed in the United Kingdom

Book review

ML for the Working Programmer (2nd edition) by L. C. Paulson, Cambridge University Press, 1996.

A Practical Course in Functional Programming Using Standard ML by R. Bosworth, McGraw Hill, 1996.

Introduction

The programming language Standard ML was defined (formally) in 1986 (Milner *et al*., 1990), although the family of ML languages dates back to 1974. Since the standardisation, it has been used widely for teaching in universities, sometimes as a first programming language, because it exemplifies careful language design with many advanced features to support good software engineering practice. It is now probably the most widely used functional programming language, although it has only seen limited use in commercial production of code.

In 1996, there was a minor revision of Standard ML (Gansner and Reppy, 1996), along with the incorporation of a well designed library as part of the standard. As Paulson points out: "It is worth stressing that the changes do not compromise ML's essential stability". The availability of very efficient implementations supporting the standard library could pave the way for much wider use of this language.

This review considers two ML related books. Firstly, a second edition of *ML for the Working Programmer* by L. C. Paulson, and secondly, *A Practical Course in Functional Programming Using Standard ML* by R. Bosworth. Both books claim to be practically oriented, which can be taken to mean that the emphasis is on building useful tools and programs rather than dwelling too much on the theoretical foundations and implementation of functional languages. However, the books are aimed at very different audiences. The first is aimed at experienced programmers and contains a wealth of useful advice on solving extensive programming problems with ML. Paulson's second edition is also timely in that it uses the new standard and library. The other book is a new book aimed at beginners with little or no prior experience of programming, and as such the practical advice and examples are at a more elementary level. Bosworth's book also predates the new standard, so it does not make use of the standard library.

We look at these books separately below.

Paulson

The first edition of this book was well received and has been reviewed elsewhere (Fourman, 1996). The overall structure and style of the first edition has been retained for the second edition. It still gives extensive coverage of most aspects of ML with very useful programs that go beyond just elementary illustrative examples. The book still covers a great deal of ground, but assumes a lot of basic computer science and programming knowledge as well as some mathematical competence of the reader.

However, there are some important changes in the second edition. The most important is probably the adoption of the new standard, which has required the reworking of examples to make use of the standard library, avoiding the construction of an alternative set of basic tools. This is to be applauded. Another important change is the introduction of ML modules in chapter 2 rather than chapter 7. This was not only done to facilitate the use of the library which makes extensive use of ML's sophisticated module system, but also to emphasise this important feature of ML for engineering software systems and for use in realistic examples. The use of the module system in the book is based on extensive experience by the author and others in the ML community, and the practical advice and examples are a rare and valuable source of information on this aspect of the language.

The earlier adoption of modules in the text would tend to exacerbate the alienation of readers who have little programming experience, so Paulson has also simplified the introductory text. This is well written and melds well with the rest of the book, but it does not make the book suitable for beginners. The subsequent chapters follow the same sequence as the first edition: names functions and types; lists; trees and concrete types; functions and infinite data; reasoning about functional programs; abstract types and functors (a more detailed look at the module system); imperative programming in ML; and two case study chapters (Lambda Calculus and Theorem Proving).

Good program structure (for maintenence and clarity) and correctness are emphasised in carefully honed examples throughout, although techniques for reasoning about correctness and equivalences are given a separate chapter rather than used pervasively. There is still appropriate attention to efficiency without sacrificing clarity and important functional programming transformation techniques are used to improve efficiency where it is important. Some of the examples of highly reusable general code have also been made more efficient than the versions appearing in the first edition.

The theorem prover case study is not an ideal choice. It would be a difficult read for anyone not already familiar with sequent calculus and theorem proving and many of the programming techniques used in the case study are illustrated well enough in the Lambda Calculus case study. However, this is Paulson's main area of expertise in applying ML to programming problems and consequently shows a great deal of insight into the problem. I would have preferred to see more elaboration of the new standard library (a case study using some of the low-level and operating system facilities), but the library was still under development when this edition was written.

Bosworth

This introductory text is based on a one semester course on functional programming, and is written at a more elementary level. It does not attempt to cover ML, but uses ML as a vehicle for introducing functional programming methods and concepts. It is not surprising, then, that there is no mention of modules or of the imperative features of ML. Higher order functions are introduced relatively early in chapter 6, and type definitions are introduced in chapter 7 before lists in chapter 8. Other structured types (trees in particular) and abstract types are not mentioned. Instead, the later chapters cover character lists and I/O culminating in a chapter of case studies.

The approach is said to be "a problem-solving one – a problem is stated, then the features of the language needed to solve that problem are introduced in a natural way" (quoted from the preface). This could be an engaging approach if done well because most texts on functional programming tend to emphasise concepts first, using examples for illustration before launching a direct attack on difficult problems.

The book does not always follow the approach stated in the preface. The examples are not always well-chosen, and are sometimes introduced just for illustrating some syntax. The resulting structure of the book is a little haphazard. For example, ML's *let* construct for introducing local definitions in expressions is introduced in section 4 of chapter 7 (which is about *Constructor Functions*) in between sections on *Polymorphic Constructor Functions* and *Stacks*. This could be avoided as the *let* construct is not needed or used at this point in the book. A glance at the contents pages reveals some other incongruities in the chapter sections as well. Explanations of important concepts are sometimes buried and can be perfunctory (for example, type variables and type inference rules on page 33, polymorphism on pages 113–114 are not well covered).

The program examples are not always well honed. For example, if a robust version of a recursive function needs to check arguments, a local auxiliary function can be used to perform the recursion when the check does not have to be repeated. This technique is used in an early example, but not in subsequent definitions of the same nature.

Although the book does not get far, a valiant attempt is made to introduce a purely functional approach to I/O in chapter 10 using ML's procedural input and output streams for implementation. The mechanism used relies on the programmer ensuring single-threaded use of the stream values. It would have been better to go one step further and use the more modern approach of monads – a functional mechanism to ensure single-threaded use (Peyton Jones and Wadler, 1993). This chapter also misses an opportunity to discuss ML's procedural I/O (the details of the stream implementation are not explained and are relegated to an appendix). A discussion of the problems arising from procedural/functional mixes and non-single-threadedness is not included.

The case studies are quite reasonable, although not very complex. Robust design of components is illustrated, but the limited coverage in prior chapters means that many generic solutions and tools cannot be deployed to demonstrate fully the power of a language like ML.

Summary

The two books reviewed have different aims, and are designed for different audiences. Paulson's second edition is a well judged revision of an already good book, and can be recommended to experienced programmers wanting to learn about ML. Bosworth's book is one of the few functional programming texts aimed at novices, but the book's structure and examples are not ideal and I would not choose it in preference to some existing introductory texts.

References

Bosworth, R. (1996) *A Practical Course in Functional Programming Using Standard ML.* McGraw-Hill.

Fourman, M. (1996) Book review. *J. Functional Programming*, **6**(1), January.

Gansner, E. R. and Reppy, J. H. (eds.) (1996) *The Standard ML Basis Library Reference Manual.* In preparation, URL http://www.research.att.com/ jhr/sml/basis/

Milner, R., Tofte, M. and Harper, R. (1990) *The Definition of Standard ML.* MIT Press.

Paulson, L. C. (1996) *ML for the Working Programmer (2nd edition).* Cambridge University Press.

Peyton Jones, S. L. and Wadler, P. (1993) Imperative functional programming. *Principles of Programming Languages*, January.

CHRIS READE
Department of Computer Science and Information Systems
Brunel University
Email: `chris.reade@brunel.ac.uk`