BOOK REVIEWS

ML for the Working Programmer. *L.C. Paulson, Cambridge University Press, 1991. ISBN 0 521 39022 2*

The great strength of this book is that its title means what it says. This really is a text for programmers who want to use ML.

The text is essentially divided into three parts. The first is introductory and consists of chapters 1 (Introduction), 2 (Names, Functions and Types) and 3 (Lists). The second deals with more intricate aspects of ML and is made up of chapter 4 (Trees and Concrete Data), chapter 5 (Functions and Infinite Data), chapter 6 (Reasoning about Functional Programs), chapter 7 (Modules) and chapter 8 (Imperative Programming in ML). The third part consists of two case studies, one in chapter 9 on Writing Interpreters for the Lambda-Calculus and the other in chapter 10 on A Tactical Theorem Prover.

These chapter titles don't tell the whole story, however, and other important subjects are to be found in various places. There are discussions of orders of evaluation, equality types, exceptions, higher order functions and input/output for instance, and there are some smaller case studies mixed in with the explanations of the language including explorations of sorting methods and of priority queues. These two, and the various implementations of these concepts provide a theme which runs through several chapters.

I was particularly impressed with the large number of examples which litter the book and there are useful exercises at appropriate places in the text. It is a pity that at least some of these were not accompanied by solutions or even hints to them, because many of them make useful supplementary points of which it would be useful to see further discussion. (I have recently learned that answers to the exercises in the first five chapters may be found in the compressed text file `answers1-5.txt.Z`, which is available by anonymous ftp from `ftp.cl.cam.ac.uk`.) Very occasionally a less well thought out exercise creeps in - for instance one (3.21) asks the reader to work out what faulty program the author wrote which differed from the one given in the text!

Another feature of the book which makes an impact is the use of a large exclamation mark in the margin to indicate a warning about a possible hazard. Only a few of these appear but they are apt when they do.

I did not really like the discussion (in chapter 5) on how to implement lazy evaluation in ML. A simulation is presented which uses functions to delay the evaluation of elements of structures and calls to force their evaluation at some later stage. While this implements normal order evaluation it does not really cope with lazy evaluation because shared structures are evaluated many times rather than once only (it is not until much later, in the chapter on imperative programming, that this problem is admitted); and besides only normal order data structures are treated where lazy evaluation should also delay the evaluation of non-strict function parameters. I found this whole discussion unnecessarily apologetic for ML's strict evaluation regime.

By and large this is a well written book but occasionally the style of writing tends towards obscurities and unnecessary complications. A comment is made in chapter 2 that *n*-tuples are not needed if only pairs are available. While this is so, it is also true that we could make do with a classical Turing machine. This kind of arcane comment is not needed in a book which concentrates on the 'how' rather than the 'why' of functional programming. On the other hand, while grateful for a detailed chapter (7) on modules, the explanation of the more involved details of this feature needs attention. One is left feeling a bit bewildered by the complexity of the module system. For instance, what is the difference between a functor and a higher order function? Why is there a difference at all?

The final chapter on implementing a tactical theorem prover also requires extra explanation. The preface does indicate that an understanding of elementary logic is required of the reader. However I found the chapter hard going.

Please send books for review to Professor UHM Martin,
Computational Science Division, University of St Andrews, St Andrews,
Fife, Scotland KY16 6SS, UK

Let me end by reiterating that I found this an excellent book in most ways. Perhaps the fact that one has to read and re-read certain chapters is a positive quality. After all it indicates that functional programming in general and ML in particular are deep subjects and it is well that the working programmer should not see only a shallow view of the subject.

Tony Davie
St Andrews

Review of Programming Languages Syntax and Semantics. *David A. Watt, Prentice-Hall, 1991. 389+XVI pages.*

This is the second of three books by David Watt on programming languages to be published in the Prentice-Hall International Series in Computer Science. The first book *Programming Languages Concepts and Paradigms* provided an excellent introduction to various aspects of the design of various styles of programming languages. This book is intended to give a similarly broad introduction to the problem of specifying programming languages. The third book *Programming Language Processors* will provide a study of how to implement programming languages.

The book opens with an introductory chapter to the problems of defining programming languages relating to their syntax and semantics. The chapter also introduces a rudimentary block structured language with functions and procedures which is used as a case study throughout the book. The language is just complex enough to provide some difficulties to overcome in its formal definition but it does not contain any particularly troublesome constructs such as a heap or goto's. Also, expressions (like those occurring in the predicate part of a conditional) are required not to have side-effects. It should be noted that this language will also be used as the basis of the language processors book.

Chapter 2 discusses how to specify the syntax of a programming language using context free grammars and regular expressions. The chapter discusses the problems associated with producing a grammar and how grammars will admit programs that are ill-formed - for example, expressions which contain undeclared variables or do not type check.

There are then three chapters on denotational semantics. The first introduces the basic principles of how semantic functions are constructed and how storage, environments and abstractions are modelled; the second shows how these techniques can be applied to the semantics of programming languages. This includes the obvious specification of dynamic behaviour but also shows how contextual constraints can be specified, how formal semantics can be used to reason about programs and how to produce prototype implementations from a semantics. The third introduces the mathematical theory behind denotational semantics. This method of introducing denotational semantics 'without the maths' does have some appeal and should be popular with computing students. The method is presented well and the descriptions, particularly of the underlying theory, are very easy to follow.

Chapter 6 is an excellent introduction to algebraic semantics written by Muffy Thomas and serves two purposes. Firstly it presents a very useful technique for the specification of the semantics of data types. Secondly it acclimatises the reader to the style of presentation that is used in the remainder of the book which is on action semantics. This is a presentation of the work developed by Peter Mosses and the author. It is set out in two sections: one on basic principles and one on applications that directly parallel the presentation for denotational semantics. This makes it very easy for the reader to make comparisons between denotational and action semantics.

The book concludes with some points on how programming language specifications are useful to language designers, implementors and users. Each chapter of the book contains a section on further reading and is provided with a large number of exercises of varying difficulty. The end material of the book includes answers to selected exercises and a full description of the programming language that was used in the book. This presents an informal description and abstract syntax followed by complete denotational and action semantics.

This is a book that is unashamedly aimed at students on computer science courses who need an understanding of semantics, and does not require the mathematical background of some other books.

Anybody who has a working understanding of set theory, recursive functions and logic should be able to read and understand this book. The preface of the book mentions that 'where necessary, depth has been sacrificed for breadth'. This approach may not be compatible with the needs of every purchaser. Some may consider it is a better use of breadth to show a larger collection of language constructs can be modelled by one technique. Others may want an illustration of other techniques, such as axiomatic semantics or structured operational semantics. However, this book will satisfy a number of peoples requirements, particularly when used on a course with its two companion volumes.

Simon Brock
St Andrews