JFP **13** (6): 1087–1088, November 2003. © 2003 Cambridge University Press
DOI: 10.1017/S0956796803004842 Printed in the United Kingdom

Book review

Concepts in Programming Languages, by John C. Mitchell, Cambridge University Press, 2002, ISBN 0-521-78098-5

This book is concerned with concepts in programming languages, issues in their implementation, and how language design affects program development. It is aimed at upper-level undergraduate students and beginning graduate students with some experience in procedural and OO programming. Functional programming experience is claimed to be helpful but non-essential. As a teaching text, it competes with a similarly-named book by Sebesta, a book by (Wilson and) Clark, and others.

Like most books in this area, it is mostly organised by concept (rather than by language), with a (here, relatively short) history of programming languages at the beginning. The "foundational" chapters are then concerned with computability, Lisp, syntax, semantics and functional programming. Next, a number of chapters deal with the traditional concepts in procedural languages, followed by a third section on object-oriented programming. The final chapters, as usual in texts in this area, deal with concurrency and logic programming (the latter chapter is by Krzystof Apt).

As one might have expected from the author, this book is an extremely rich source of important information, subtle issues, and diverse and challenging exercises. This makes it an excellent text for further reading. As an introductory text, however, it is less suitable. This is mostly due to structuring and content, and partially to bias.

As an example of problems with structure and content, consider (parametric) polymorphism. Chapter 5 uses the ML "length" function as an example (without giving its type), and mentions polymorphism four pages later as a subject to be covered in Chapter 6. Then, Exercises 5.3(d)–5.8 assume familiarity with polymorphism. Finally, Chapter 6 treats the subject – in a very nice way indeed, introducing it as an effect of type inference, and contrasting it in full detail with the C++ template mechanism.

A few important concepts are not described at all, for example there is no mention of separate compilation in Chapter 9, on modules; this will make it hard for beginners to understand discussions on **when** things are compiled in Smalltalk, or in C++ using STL. Chapter 8, on control structures is unusual in not covering the traditional collection of loops and branching structures – in particular, the fall-through case statement in Exercise 10.2 must baffle some readers. In general, the exercises are quite stimulating, but by assuming language knowledge not covered in the preceding chapter(s) they may appear too difficult to students.

As a reference text, the book would need a much more extensive index (it is now only 4 pages) and a glossary (which now contains only 70 terms). Only Chapter 15 by Apt contains a section "Bibliographic Remarks" with pointers for further reading; similar sections in other chapters would have been very helpful.

The author's bias is most strongly felt where it concerns Lisp and ML. The introduction to Lisp is included in the "Functions and Foundations" part of the book, with the design of Lisp even described in a section called "Good Language Design". It is suggested that many of the principles covered later will be explained using Lisp. However, Lisp plays a marginal role in the rest of the book – maybe the strongest symptom of a Lisp inspiration is felt in an abundance of activation records and similar pointer structures, with diagrams, in later chapters. The lambda calculus is, as one would expect, given its proper place in the history

of programming languages. (Sebesta, in contrast, does not mention it until page 594.) This is then followed by the functional vs. imperative languages discussion.

The second part of the book starts with a chapter on "The Algol Family and ML". This is a somewhat unusual combination. The introduction claims that ML is used to "discuss some important concept in more detail" but the core of the chapter is actually a 20 page introduction to the basic concepts of ML. The low point of this book is where a hypothetical confused C programmer is explained that "unit" is a better term for the information-free type than "void", because "The ML type system is based on years of theoretical study of types; most of the typing concepts in ML have been considered with great care." In the chapter on concurrency, Concurrent ML as one of the two main models seems a peculiar choice – it is not a mainstream language, and the book does not relate its concepts to other more common, models of concurrency in any detail.

The coverage of OO languages is very detailed and discusses a myriad of subtleties that are not addressed in comparable texts. The organisation in this part is more per language than per concept – particular, the coverage of C++ is extensive – which risks a blurring between general and language-specific issues. This happens most notably in the section on multiple inheritance in the C++ chapter.

On the whole, the book is interesting, challenging and a pleasure to read. However, for a first introduction to the subject it is somewhat more eclectic than is desirable.

Dr Eerke Boiten
Computing Laboratory
University of Kent, UK

https://doi.org/10.1017/S0956796803004842 Published online by Cambridge University Press