9. T. Johnsson, Lambda lifting – transforming programs to recursive equations. In *Functional Programming Languages and Computer Architecture*, edited J. P. Jouannaud. Lecture Notes in Computer Science 201, Springer, Heidelberg (1985).
10. T. Johnsson, Compiling Lazy Functional Languages. *Ph.D. thesis*, Chalmers Tekniska Hogskola (1987).
11. G. Kahn (ed.), *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science 274, Springer, Heidelberg (1987).
12. P. J. Landin, The mechanical evaluation of expressions. *The Computer Journal* **6**, 308–320 (1964).
13. R. D. Lins, On the Efficiency of Categorical Combinators in Applicative Languages. *Ph.D. thesis*, University of Kent at Canterbury (1986).
14. R.D. Lins, Categorical multi-combinators. In Ref. 11.
15. R. D. Lins and S. Thompson, *On the Equivalence Between CM-C and TIM*. Computing Laboratory Report 67, University of Kent at Canterbury, (1989, revised 1990). (Submitted for publication in *Journal of Functional Programming*.)
16. D. S. Scott, Relating theories of the lambda calculus. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisation*, edited J. P. Seldin and J. R. Hindley, pp. 403–450. Academic Press, London (1980).
17. D. A. Turner, An overview of Miranda. In *Research Topics in Functional Programming*, edited D. A. Turner, pp. 1–16. Addison-Wesley, Wokingham (1990).
18. C. P. Wadsworth, Semantics and pragmatics of the lambda calculus. *D.Phil. thesis*, Oxford University (1971)

Book Review

PETER LINZ
An Introduction to Formal Languages and Automata
D. C. Heath, Lexington, MA, 1990. 373 pp. US $37.00. ISBN 0-669-17342-8

The preface to the book identifies it as an introduction to the 'Theory of Computation', corresponding closely to course CS 16 in the ACM Curriculum 78. The author's stated aim is to familiarise students with the foundations of computer science, and to teach material that will be useful later: though mathematical theorems are stated precisely, formal proofs are seldom given.

In my experience this is a difficult course to give, since students vary greatly both in their degree of mathematical preparation and in their readiness to commit themselves to an abstract approach. The solution here is to develop the skills needed to handle abstract computational structures by a wealth of practical examples and exercises, concentrating on languages and automata for the most part. Only the basic elements of the theory of recursive functions are included, in four pages that consist entirely of definitions. Complexity theory is hardly touched on: the final section quotes some of the major results, giving references to Hopcroft and Ullman's *Introduction to Automata Theory, Languages and Computation* (1979) for the proofs.

Although the approach leans heavily on examples, Linz emphasises that his aim is not to supplement texts on compilers, and the treatment is not slanted towards applications in more practical computer science. The many students who see themselves first and foremost as programmers will probably be more attracted by Glenn Brookshear's *Theory of Computation: Languages, Automata and Complexity* (1989). That book also manages to present the elements of recursive function theory in very much a programming style.

The difficulties that many students encounter with the mathematics of computation theory are more to do with notation (the language of mathematics, if you like) than with content. This new book does not make any attempt to soften the blow, and some students will be unnecessarily intimidated because of this (see for example Exercise 15 at the end of Section 7.1). The essential ideas that have to be conveyed to the reader are intuitive but quite sophisticated: in formal language theory, non-determinism and ambiguity; in computation theory, the notions of the decidable and the semi-decidable. This latter distinction is arguably the most important insight that arises in the whole theory, and it is essential that recursive enumeration is tied in with the idea of an effective process. At various stages I wished that the author had been more prepared to go into detail, none more so than in the discussion of the Universal Turing machine. I do not believe that computer scientists find an explicit construction exhausting, rather the reverse. Further, having seen and believed in such a construction, students have a sharper intuition for the enumeration theorem.

This is where I think that the new book falls down. Too little attention is devoted to establishing a feel for what is constructive, and sometimes the approach is downright misleading. An example is in the proof of Theorem 11.3, where the word *construct* is used loosely in the second paragraph. The context at that point is that of *a countable set with some order on its elements*: it is only later that the enumeration procedure is invoked. Technically this doesn't matter in the least – the first part of the proof is concerned with a language that is not recursively enumerable – but it blurs the distinction between effective construction and existential definition, and that distinction should be one of the main messages.

Where the new book scores is in motivation by example, particularly for context-free languages. In particular I found Example 5.13 (on inherent ambiguity) and Example 7.9 (which explores the power of deterministic pushdown automata in recognizing context-free languages) to be especially illuminating, clearly presented and convincing. Those sections are both thoroughly successful, and the advantages of proceeding by example are apparent. Throughout the book the quality of the printing is excellent, with generous amounts of space given to both displays and diagrams. Ideally the left-hand margin should have been made narrower on left-hand pages only (in my review copy the text slid off towards the spine), but the physical layout is way above average.

Overall this book can be welcomed for the clear treatment of regular and context-free languages, and for the wealth of examples and exercises. However, I do not think it is the front runner among general introductions to computation theory. That is a highly competitive field. Explicit interpretation of encoded algorithms is an essential feature of computational models, and students need to understand it in order to appreciate crucial diagonal arguments. I have already mentioned two books that I think are better balanced than this new one. Thomas Sudkamp's recent *Languages and Machines* (1988) is also good; it covers most of the material in Linz's book and also includes a lot about complexity theory. Davis and Weyuker's *Computability, Complexity and Languages* (1983) can claim to be a classic in much the same way as the book by Hopcroft and Ullman.

I have mentioned four books that serve as an excellent and balanced introduction to language theory and computability, and there must be others that I am not aware of. This new book can be recommended to anyone whose primary interest is in the language aspects of the subject, but it is not in my view suitable as a sole reference for a course on computation theory.

KEN MOODY
Cambridge