concerns, *Lambda Calculi* is a very good book and is a timely addition to the theoretical computer scientist's bookshelf.

MIKE JOY
Warwick University

AMOS R. OMONDI
Computer Arithmetic Systems. Prentice Hall International. 1994. ISBN 0 13 334301 4 £22.95. 520pp. softbound.

Computer systems are built from various sub-modules and peripheral devices. One of the vital elements is the computer arithmetic module. This excellent book deals with the various arithmetic operations, different associated algorithms and their detailed analysis. Cost-performance comparison of different algorithmic models have also been shown.

Case studies of different actually implemented units have been discussed in each chapter. Annotated bibliographic notes provide the readers with material for further studies. And the chosen set of exercises makes sure that the fundamentals do get absorbed.

This book consists of 8 chapters. There is a functional division of the contents into three parts.

Chapters 1 to 4 cover the basic algorithms and associated hardware details for fixed point number systems. One point may be noteworthy: coverage of topic in the Chapter 1 does touch upon the generalized number systems, including an introduction to floating point numbers and a more general title for this chapter might have been more apt.

Chapters 5 to 7 cover the floating point number concepts, associated operations and the hardware details.

Originality, clarity of expression and depth of knowledge stand out very clearly in this book. The case-studies giving the practical implementation details of the theoretical algorithms and particularly some different unusual implementations (for very fast operations in both fixed and floating point number systems) place the book at two levels simultaneously : it can be treated as a text book as well as excellent reference material for further research in the respective fields.

Chapter 8 deals with unconventional number systems and associated arithmetic, namely—

a. the residue number system with its unique carry-lessfast addition, subtraction, multiplication;
b. the decimal number system and arithmetic; and
c. redundant signed-digit number systems and arithmetic.

An extensive bibliography has been compiled at the end.

The appendices on pipelining with respect to high performance computers, shifter design and the separate copy of the design library are well appreciated. But I strongly feel that the appendices and design library should be amplified and recompiled into a separate volume which emphasises real-world designs and acts as a perfect complement to the book being reviewed.

I enthusiastically recommend this book as an useful addition to any academic or highly design-oriented scientific library. Moreover, this book will also be extremely suitable as a textbook for many universities.

S. SANYAL
Tata Institute of Fundamental Research, Bombay

JENS PALSBERG and MICHEAL I. SCHWARTZBACH
Object-Oriented Type Systems. John Wiley. 1994. ISBN 0 471 94128 X £24.95. 180pp. hardbound.

Object-oriented languages have often seemed removed from the mainstream. Although we can all accept simple notions of 'class' and 'object with methods' belonging to it as covered in many introductory courses, the necessary various (and varied) extensions to this concept to construct a generally acceptable language appear to lack a comprehensive conceptual model. Partly this is due to the variation in terminology between (say) C++ and Smalltalk exacerbated by discussions as to what being 'object-oriented' really means, and to the rôle (or otherwise) of types and by the 'our language does it this way' effect. It is also due to very real theoretical problems which mean that the progression from untyped Smalltalk to typed languages like Eiffel is sufficiently complicated to result in the latter containing type insecurities. Indeed the whole situation has echoes of the Lisp vs. Algol debate (recall Pascal's early type insecurities concerning function types). Indeed, languages like ML have become popular to a large extent because they offer the type security of Pascal with much of the flexibility of their weakly typed counterparts.

Palsberg and Schwartzbach centre their book around their BOPL (Basic Object Programming Language) and extensions thereto adding types, inheritance and genericity. BOPL is easy to understand and has a simple evaluation mechanism (although I disapprove of the authors following the trend to present such a machine only informally in English). Moreover, extensions for inheritance and genericity are defined by source-to-source transformation into the original BOPL. This has the great advantages of simplicity and of exposing otherwise non-obvious semantic choices which are normally glossed over by language designers who have dismissed (or even overlooked) alternatives. In particular, the interaction between inheritance and recursion is well exposed.

Type systems are defined in a similar style—a typed program is one whose type annotations satisfy certain invariants. Type inference is viewed as the minimum solution of constraints derivable from the program. Types are defined as *sets of classes* which a value might belong to; operations accepting any class derived from an original class are seen as accepting the *cone* consisting of the set of all such derived classes. The syntactic explanations of inheritance and genericity above can be thus shown to be type correct in a pleasing manner. Palsberg and Schwartzbach successfully (to my mind) argue the need to explain object-oriented semantics as discussed rather than to force-fit the ideas to a λ-calculus meta-language (note also that Abadi and Cardelli have since proposed a calculus of objects which shares a similar aim but focuses on more theoretical issues concerning typing). This action encourages them to propose that genericity be explained by substitution of classes in others rather than by the notion of application which they successfully criticise. The overall effect is that of having secrets explained.

By way of (slight) criticism, I would have welcomed more explanation of C++, particularly the rôle of `virtual` and non-`virtual` member functions which could have justified late binding rather better than its proclamation in Section 2.2.5. Moreover, the authors' claim of the leniency of C++ assignment between pointers to class and derived-class is not supported by the C++ annotated reference manual (Ellis and Stroustrup). If another chapter were to be added I would strongly suggest one on multiple inheritance and sharing—I believe the authors' methods would cast light here just as well as they have succeeded for types, inheritance and genericity. I personally would also be interested in discussion of the way in which ML-style polymorphism differs from object-polymorphism especially given the problems of the former with mutability which is at the heart of object-orientation—but this would probably have resulted in a very different book.

In addition to the book itself, the main translations and algorithms have all been implemented and made available to the public at large via *ftp*. I found these easy to obtain and use (`daimi.aau.dk` warns one to use the new file server at `ftp.daimi.aau.uk`) but one needs to have access to Scheme.

This is a book which I would commend to those from outside the object-oriented community (and many inside) who wish to understand the range of object-oriented language design possibilities and understand the semantic choices in detail. I welcome its emphasis on type security which is achieved without the excess formalism sometimes found. I concur with the authors' assessment of its utility for final year undergraduate teaching and I would further recommend it for anyone interested in semantics and types of object-oriented languages. In addition to the technical content, the English is indistinguishable from that of many a native speaker (with the notable exception of the use of 'overwrite' for 'override'), the authors have taste in graphic design as befits camera-ready copy and there are very few, unimportant, typographical errors. I enjoyed reading it.

ALAN MYCROFT
Laboratoire d'Informatique, École Polytechnique
(on leave from Computer Laboratory, Cambridge University)

J.G. TURNER and T.L. MCCLUSKEY
The Construction of Formal Specifications. McGraw-Hill. 1994. ISBN 0 07 707735 0 £21.95. 420pp. softbound.

Formal methods seem to be assured a place in the Computing Science curriculum. Some universities base their degrees entirely on formal methods; others offer a single course on formal methods, in final year. That reflects growing commitment in industry to the application of formal methods at various stages in the design cycle.

Applied to the early stages of the design cycle formal methods provide a notation for requirements capture. Formalising requirements forces the customer to decide at the very start of the development process what he wants, when a change of mind is far less costly; the result acts as a contract between customer and system developer whose properties may be either derived by proof or observed by prototyping. Applied to the latest stages of the design cycle, formal methods provide techniques for producing correct code: for the correct termination of loops, invocation of procedures, and so on. In between those extremes of abstraction they offer criteria for ensuring the functional correctness of one description with respect to a more abstract one. The lower-level description whilst chosen to incorporate bottom-up constraints and efficiency concerns must meet those criteria to be valid.

Thus crucial to formal methods is a formal description, or specification, at a given level of abstraction. How it is constructed depends on its location in the development cycle; techniques for requirements capture are quite different from those for downcoding into a programming language.

This book is about the construction of specifications. However it considers description only at the level of requirements capture, and discusses neither the development cycle nor criteria for a specification to meet a more abstract one (though the VDM terminology for that, reification, is mentioned once by name). Without some treatment of such issues it is difficult to see what to make of a specification once it has been constructed! In what way does it constrain its refinements? Readers of this book must presumably rely on intuition for that, which seems to undermine the whole benefit of formality.