the useful notes and references at the ends of the chapters. It is a nice clear piece of mathematical exposition, and more or less self-contained, although readers who are not used to reading mathematics might find it heavy going. I particularly like the exercises and carefully chosen examples given at every stage of the development.

It has to be said that the book is much more demanding than many computer science texts which treat the same material, although of course it goes much further, and worlds away from the three page summary of resolution sometimes found in introductory PROLOG books! Is it worth the effort for anyone except the dedicated mathematician? The answer is yes, particularly for anyone who wants to develop new logical systems for applications in computer science, or to understand other people's attempts to do so. Sadly, sloppy and inaccurate accounts of supposed new results in logic, with inadequate proofs or no proofs at all, can easily be found in the computer science literature. Gallier's book shows what the standards ought to be.

Finally, a word to the publishers. Nice book—shame the hard-back binding is falling apart already.

Ursula MARTIN
Department of Computer Science
RHBNC, University of London
United Kingdom

Logic and Computation—Interactive Proof with Cambridge LCF. By L.C. Paulson. Cambridge University Press, Cambridge, 1987, Price £27.50, ISBN 0 521 34632 0.

The title of this book, derived from the name of the system it describes (LCF), summarizes its contents quite accurately: the relationship between logic and computation. The two parts of the book reflect the two levels of this relationship: LCF as a logic for reasoning about computations, and computation as a meta language for an implementation of LCF.

LCF stands for "Logic of Computable Functions" and was developed by Dana Scott in the late 1960s for reasoning about denotational semantics. In the early 1970s Robin Milner and colleagues at Edinburgh implemented this logical system in a theorem prover called *Edinburgh* LCF, documented in [1]. Edinburgh LCF made two seminal contributions to computer science: its meta language ML and its theorem proving style based on *tactics* and *tacticals*. ML has recently been standardized and has become a widely used functional programming language. To avoid confusion, the logic LCF, as opposed to the system, is often called *PP*λ.

Edinburgh LCF has spawned a number of descendents, among them *Cambridge* LCF, which extends the original logic with new connectives and tactics, and provides a better user interface. Pauson's book differs from [1] in that it covers the logic and domain theoretic background but assumes that the reader is familiar with Standard ML. The inclusion of the background material and the separation of ML, together with the superior layout, yield a much more accessible presentation.

The book consists of two parts, dealing with the underlying theory and the system itself. PPλ is a classical first order logic for reasoning about domain theory and functional programming. Consequently the first part starts with an account of three proof systems for first order logic: a natural deduction system and two sequent calculi. The presentation of each system is accompanied by a discussion of the relative merits of using the system as a basis for a computer implementation. This is followed by a gentle introduction to domain theory and a presentation of the relevant proof rules, thus concluding the description of the bare logic of LCF. The last chapter of the first part deals with semantics of data types, and how to reason about them, in particular by induction. Recursive domain equations are solved via the inverse limit construction, making this chapter mathematically quite demanding. On the logical level a theory of strict and lazy lists is developed, including structural induction principles.

For the second part of the book describing Cambridge LCF system, the reader needs to be familiar with the basic syntax of Standard ML, the meta language for computing with PPλ objects. After introducing the syntax of PPλ terms, formulae and theorems, forward reasoning via inference rules, and backward reasoning via tactics and tacticals is explained. A whole chapter is devoted to tactics for rewriting and simplification, real "workhorses" as he calls them. Care is taken to present both the computational aspect (i.e. the implementation as ML functions) and the relation to the proof systems of the first part. The second part closes with a collection of sample proofs, demonstrating important aspects of theorem proving in LCF like the definition of new theories, structural induction and rewriting.

Both parts are essentially self contained and can be read independently of each other. Despite Paulson's sometimes "breathless" prose, part I is well presented and easy to read (except for the inverse limit construction, which requires some mathematical sophistication). Part II suffers a bit from the user manual syndrome: some sections simply consist of lists of inference rules or tactics accompanied by a short description of their functionality. I would have liked to see more of a discussion of the relative merits of the LCF way of doing things. In particular a comparison with Paulson's Isabelle system would have been interesting. Throughout the book there are a host of well-chosen examples and exercises, which part II—in particular—benefits from; even without immediate feedback from an implementation, the examples enable the user to follow the text quite well.

If you are looking for a book on logic or domain theory, this is probably not the right choice. But I do recommend it to anyone teaching or doing research in the

area of automated theorem proving. Even if you are already familiar with [1], you will find many new ideas. In particular the coherent treatment of both logical and implementation issues should make this book attractive for a wide audience. If you don't have access to Cambridge LCF, the preface tells you how to obtain it free of charge.

Tobias NIPKOW
MIT Laboratory for Computer Science
Cambridge, U.S.A.

References

[1] M.J.C. Gordon, R. Milner and C.P. Wadsworth, *Edinburgh LCF*, Lecture Notes Comput. Sci. **78** (Springer, Berlin, 1987).

Understanding Z—A Specification Language and its Formal Semantics. By J.M. Spivey. Cambridge University Press, Cambridge, 1988, Price £17.50, ISBN 0 521 334292.

1. Overview

The primary purpose of this book is to give a formal semantics for the Z notation.

The first chapter introduces the language by means of a short example, discusses the reasons for providing a formal semantics and the problems arising from 'meta-circular' definitions. It compares Z with VDM and with algebraic specification techniques.

Chapter 2 identifies the semantic domains, and Chapter 3 provides the abstract syntax and the semantic mappings.

Chapter 4 offers an enhancement to the semantics to support generic definitions, discusses apparent instances of referential opacity, alternative treatments of partial functions, and the relationship between Z and Clear.

The final chapter discusses four further topics of interest. Proof rules for some of the operations of the schema calculus are provided and are shown to be sound by reference to the semantics. A method is shown for consistently introducing new types, illustrated by the natural numbers. The relationship between specifications and implementations is discussed, as is the need for non-determinism.

A summary of notation is provided, which appears to cover the variant of Z used as a metalanguage, and an index of definitions is conveniently condensed onto two facing pages.