functional abstraction, and laziness, that are new to students often find hard on first encounter but are treated steadily and sympathetically through generally well-conceived, if occasionally irritating, practical examples to illuminate wider theory.

The book's main strength is the thorough treatment of I/O, monads, monoids, and stateful functional programming, where many introductory Haskell texts are covered just enough to enable basic interaction. The equally substantial coverage of these topics in *Real World Haskell* (O'Sullivan *et al.*, 2009) may be better arranged for people who know what they are looking for, and lacks the distracting asides and cartoons, but the *Real World Haskell* examples are dry and have low motivation in comparison to *Learn You a Haskell*....

Frankly, if I had not been sent this book for review, I would have not bought it. But I will have no hesitation in lending it to my students.

References

Alcock, D. (1977) *Illustrating Basic* (*A Simple Programming Language*). Cambridge, UK: Cambridge University Press.

Felleisen, M. & Friedman, D. (1998) *The Little MLer*. Cambridge, MA: MIT.

Friedman, D. (1974) *The Little LISPer*. Cambridge, MA: MIT.

Friedman, D. & Felleisen, M. (1998) *The Little Schemer*. Cambridge, MA: MIT.

Kaufman, R. (1977) *A FORTRAN Coloring Book*. Cambridge, MA: MIT. Available at: http://www.seas.gwu.edu/~kaufman1/FortranColoringBook/ColoringBkCover.html.

O'Sullivan, B., Goerzen, J. & Stewart, D. (2009) *Real World Haskell*. Cambridge, MA: O'Reilly.

GREG MICHAELSON
Heriot-Watt University, Scotland

OCaml from the Very Beginning, by John Whitington, Coherent Press, 2013, £25.99, US $37.99. ISBN-10: 0957671105 (paperback), 204pp. doi:10.1017/S0956796813000087

The publisher's blurb for this book says that it "will appeal both to new programmers and experienced programmers," and that it is suitable for "an undergraduate or graduate curriculum, and for the interested amateur." That is a lot to expect of a book, and the author's preface is more modest. He explains that the book arose from his experiences working as a tutor at Cambridge for the first-year course taught by Larry Paulson using Paulson's classic text "ML for the Working Programmer." It is reasonable to judge this book, then, on how it might work with first-year undergraduates majoring in a STEM discipline.

Some of these students will have prior programming experience (usually in an imperative language) and some will not. Both groups will be impatient, which accounts for the brevity of this book. It has 16 chapters, some as short as two pages. Each chapter is followed by a half-page of questions and a cumulative summary of the book so far, typeset to fit onto a single page. Hints and solutions to the questions are at the end of the book. The style is concise but not terse; the author takes care to write plainly and clearly. The book tries in this fashion to deal with a reluctant reader, or one who dips into the book in a nonlinear fashion, or to whom English is a foreign language.

The magnitude of the task is made clear in the single page of "Getting Ready," preceding the first chapter, which starts with the expression $1 + 2$ entered into the OCaml REPL. How we are to get to this point is not specified. Of course, a complete description (for each of three major platforms, one with several variants) of how to locate the software, download it, install it, navigate to a working directory using a command-line interface, and start the interpreter would make the book half again as long, and parts of it would be obsolete before the book was even printed. It is not the author's fault that OCaml, like most programming languages, lacks a beginner-friendly IDE. The course instructor is going to have to supplement, and the "interested amateur" doing this on their own will need a fair amount of background.

The first chapter continues to discuss entering simple arithmetic expressions, which immediately requires a discussion of operator precedence. Some precedence relations are given, but the notions of levels of precedence and associativity are never discussed, and there is no comprehensive table. The introduction of a second type (Boolean) allows the author to approach type errors by means of the expression 1+ `true`, which gives a clear error message. Further discussion is deferred to the appendix titled "Coping with Errors." But there, the same example is given, and the only other advice on type errors provided is "look at each function and its arguments, and try to find your mistake." The inadequacy of this approach can be seen by taking any list-processing function (such as length) and surrounding the head–tail deconstructing pattern with list-creating square brackets. The resulting error message is incomprehensible to the beginner. Again, this is not the author's fault, and it is not unique to OCaml. I have seen senior CS majors struggle with type errors in SML and Haskell until they have fully grasped the Hindley–Milner type inference algorithm (not a reasonable option for a beginner). The word "infer" does not appear in this book, let alone "ascription." Once functions are introduced in the second chapter, their types as inferred by the OCaml REPL appear above them in displayed examples, and this works for correct code because the result of type inference matches our intuition. However, the student who writes incorrect and untypable code has nothing to fall back on when intuition fails. Graham Hutton's otherwise excellent *Programming in Haskell* suffers from the same defect. This is not an easy issue to address, but an effort should be made.

The eight questions at the end of the first chapter are each made up of two or three short interrogatory sentences, but take the reader through more precedence and associativity issues, integer limits, edge cases and overloading of arithmetic operators. This is as it should be. Active learning is an important component of mastery and students should learn new concepts through exercises as well as the main text. However, the book could do more to emphasize that future chapters expect this learning to have taken place. The instructor will probably have to enforce compliance.

Subsequent chapters introduce pattern matching, lists, higher-order functions, exceptions, tuples, curried functions, algebraic data types, side-effects and sequencing, I/O, references, arrays, floating-point numbers, and a brief look at the OCaml Standard Library. The choice of examples is fairly standard: sorting algorithms, association lists and binary search trees, and batch file processing. After the fifth chapter, there is a page suggesting the use of an unspecified text editor to enter and save programs which can then be manually loaded into the REPL with the `use` command.

Algebraic substitution is used to explain the result of applying recursive functions, starting in the second chapter, and to sketch ideas of time efficiency without introducing recurrences or order notation. Its use in discussing space complexity is more problematic (for example, it does not show sharing). The author tries to motivate tail recursion by showing expanding intermediate expressions, but omits proper parenthesization at a crucial point: the student seeing `length [5; 5; 5]` replaced by 1 + 1+`length[5]` is going to wonder why the leftmost addition cannot be completed. (The words "strict" and "eager" are not used.) The chapters on I/O and mutation (the latter being the longest in the book at nine pages) focus mostly on "how" rather than "why," and do not present a computational model (which would be difficult given the self-imposed space constraints). They are going to be effective only with students who have prior experience with imperative programming.

The last chapter takes seven pages to cover the OCaml `ocamlc` and `ocamlopt` compiler commands, modules, interfaces, and standalone programs. Much is left out, here and before; the other constituencies mentioned in the publisher's blurb, graduate students and experienced programmers, are likely to be frustrated by the omissions, and by the pace of the book overall. Arguably, both groups could benefit from a slow and careful introduction to the subject, but they are likely to decamp in favour of tutorials offering more immediate red meat.

There is much to like about this book. Its economy and reach for elegance form a welcome contrast to the overstuffed awkwardness of many introductory texts. However, it is perhaps a trifle too lean. It could have afforded to spend a little more space dealing honestly with the messy edges, rather than trying to smooth them over or pretend they do not exist. I am not aware of any other OCaml text trying as hard to be friendly to beginners, and this effort is to be commended. I hope the author will consider an expanded second edition.

PRABHAKAR RAGDE
University of Waterloo