JFP **34**, e11, 3 pages, 2024.

doi:10.1017/S095679682400008X

Book Review

Review of "Real World OCaml: Functional Programming for the Masses"
Second Edition, by Yaron Minsky and Anil Madhavapeddy, 2023

The authors present the book as "a fast-moving tutorial" whose aim is to "take [the reader] through the concepts of the language at a brisk pace" and "quickly [teach] how OCaml stands out as a tool for writing fast, succinct, and readable systems code using functional programming". I find that the authors do indeed "move fast". Although the tone is gentle, the progression is fairly quick. The book is not aimed at beginning programmers. I recommend it to readers who are fairly proficient programmers already, either in OCaml or in other languages, and who wish to discover the main high-level features of OCaml, some of its low-level aspects, or some key components of its ecosystem of libraries and tools. Rather than "a tutorial", I would describe the book as a coherent collection of tutorials on various aspects of programming with OCaml. The tone is definitely that of a tutorial: the book teaches by example and emphasizes practice over theory. It is about getting things done—with style.

The book clearly aims to be comprehensive both in breadth and in depth.

In terms of breadth, the book covers almost all aspects of the OCaml language as of version 4.x, including variables and functions, immutable and mutable data structures, generalized algebraic data types, modules and functors, first-class modules, objects, and classes. It offers tutorial introductions to several libraries and tools that help with a number of common tasks, including storing data in dictionaries, parsing the command line, programming concurrent applications that communicate over a network, testing, serializing data, and parsing computer languages. Finally, it describes several low-level aspects of the language, such as the foreign function interface, the layout of values in memory, and the garbage collector.

Although the book is thick (a little under 500 pages), the writing is pleasant. One could certainly read the book from cover to cover just for the sake of discovering the elegance of OCaml's objects and classes (Chapters 13 and 14), the pleasures of concurrent programming with Async (Chapter 17), or the inner workings of OCaml's garbage collector (Chapter 25), as well as many pieces of advice and tidbits of wisdom along the way, such as how to accurately measure the cost of a computation, in time and in space, using the small library `core_bench`. Besides, the book serves as a welcome reference on some topics that are scarcely documented elsewhere. Someone who is confronted with the task of setting up a test suite for an OCaml project (Chapter 18), packaging up and distributing an OCaml library (Chapter 22), or writing glue code to allow interaction between OCaml and C (Chapter 23), will find the book helpful. A beginning programmer is not likely to

face these questions, but someone who wishes to use OCaml for a real-world project is likely to encounter them at some point.

In terms of depth, the authors are not afraid to dig pretty early on and pretty far down into the more subtle, complex, and sometimes murky corners of the OCaml language, such as the rather obscure interaction between optional arguments and partial application, which is discussed as early as Chapter 3, or the nitty-gritty details of the problems that arise when several record types have a field label in common are discussed in Chapter 6.

Furthermore, the authors spend time explaining why certain programming styles should be avoided and propose remedies or alternative styles. For example, they describe situations where a programming mistake will cause an unmanageably large type error message or, on the contrary, will not be detected at all by the OCaml-type checker. They provide simple and solid advice to maximize the chances that mistakes are detected and lead to good error messages. For instance, minting abstract types (that is, creating several abstract types to avoid confusion between several distinct uses of a single concrete type) helps detect mistakes, whereas annotating every function with its intended type leads to more accurate and more concise type error messages. As another example, the authors warn against the use of OCaml's polymorphic equality and hashing functions, whose behavior can be questionable. The polymorphic equality function does not respect type abstraction and can view two sets as "different" even though they have the same elements. The polymorphic hash function performs a traversal whose depth is limited: therefore, the hash code of a list is computed based on its first 10 elements only! This expert knowledge and advice is valuable and is seldom put in writing.

The reader should be warned that throughout the book the authors rely on the facilities offered by `Base`. This library, which replaces OCaml's standard library, aims to offer more consistent and much more comprehensive APIs. Furthermore, they rely on `ppx_jane`, a set of preprocessor extensions that offer greater syntactic comfort and can generate boilerplate code (such as serialization and deserialization functions). Thus, to a certain extent, the authors describe a nonstandard dialect of OCaml. In my eyes, this is justified by the desire to get things done with as little fuss as possible. A reader who does not wish to rely on `Base` or `ppx_jane` can nevertheless find significant value in the book.

Because the authors rely on `ppx_jane`, they present its facilities for writing code in monadic style, including `let%bind`, `let%map`, `match%bind`, and `match%map`. Regrettably, they do not describe the "binding operators", also known as "`let`-operators", that have been introduced in OCaml 4.08 for the same purpose. Mentioning this feature of OCaml in Chapter 8 would have made the book even more comprehensive.

As a result of moving at a "brisk pace", the book can be slightly difficult to follow in places. Sometimes the reader must guess what the function `Or_error.try_with` does, or what the type of `Error.create` might be. Often, but not always, the answer is found later on in the book. In a few places, the authors use words or concepts that have not yet been presented. For instance, the discussion of memoization in Chapter 9 involves vocabulary such as "collected" and "leaking memory". This discussion assumes a good understanding of garbage collection, which is covered only in Chapter 25. These minor problems seem tolerable in view of the target audience.

Although the authors write that "this second edition brings the book up to date", one must note that the book regrettably does *not* cover OCaml 5, which was released in

https://doi.org/10.1017/S095679682400008X Published online by Cambridge University Press

December 2022, shortly after the book was published. OCaml 5 offers several important new features, such as support for hardware parallelism and effect handlers. Effect handlers promise to remove much of the need for monadic style and could therefore make `let%bind` and friends largely obsolete. I am looking forward to this and to a third edition of the book that describes the new features of OCaml 5 and their use in real-world libraries and applications!

Among the books recommended at ocaml.org, only "More OCaml: Algorithms, Methods, & Diversions", by John Whitington, seems comparable to "Real World OCaml". Whitington's book begins with a short presentation of several language features, then moves on to applications, with a focus on a single extended example, namely generating PDF files. "Real World OCaml" has broader coverage, including tools, packaging, low-level aspects of OCaml, and relies on Jane Street's library and syntax extensions, whereas Whitington's book sticks to the standard library and does not use syntax extensions.

In conclusion, this book covers many aspects of the OCaml language and of its use in real-world scenarios and can be valuable both as a fast-paced tutorial introduction and as a quick reference. I believe that many OCaml programmers, including experts, will want to keep it at hand. I should mention that the book can be read online for free in a form that is fairly pleasant to the eye. The source code of the book, as well as the fragments of OCaml code presented in the book, can also be found online. This is a valuable resource for the OCaml community.

FRANÇOIS POTTIER
Inria Paris
Paris, France
(e-mail: francois.pottier@inria.fr)

https://doi.org/10.1017/S095679682400008X Published online by Cambridge University Press