JFP **30**, e4, 2 pages, 2020. © Cambridge University Press 2020
doi:10.1017/S0956796820000015

Book Review

Review of “Thinking with Types”* by Sandy Maguire, LeanPub, 2019

Type-level programming is a growing subfield within Haskell programming. The vast majority of new extensions to Haskell proposed and implemented have to do with augmenting the type system to allow for tighter restrictions on what programs can be allowed, and many see the future of Haskell as being a fully dependently typed language. However, while many dependently typed languages have received their own introductory texts, to my knowledge, there has not yet been a book aiming to teach these concepts in the idiosyncratic context of Haskell. This book is intended to fill this niche.

Thinking with Types covers a broad range of topics in type-level programming, explaining the basics of Haskell’s kind system and notions of variance before moving on to discuss deeper topics including scoping, constraints, higher-rank polymorphism and existentials. These topics take up the first half of the book; the second half is dedicated to applying them, including a version of the printf problem, extensible records and dependent types via singletons. These examples are well motivated and presented in a way that should make it easy to generalise.

As far as I am aware, this is the first book to address these concepts in Haskell. However, there have been many academic papers published that discuss the same ideas; one need only to look at the proceedings of the Haskell Symposium to find a glut of such papers. Thinking with Types serves as a sort of digest for these ideas. As such, the concepts in the text are not new ideas but rather expositions of tried-and-tested techniques from type-level Haskell.

Given the title and aims of the book, it is somewhat unfortunate that it should take nearly half of its length to see any real, practical type-level programming. This is unlikely to be an impediment to the motivated reader, but it is something of a flaw that a reader should have to persevere through the book before seeing anything particularly practical. I would have liked to see these two elements of theory and practice more interleaved, perhaps by way of some running example code that becomes progressively safer as each new technique is applied. Broadly, the ideas in the book are interesting and clearly explained, although this separation between theory and practice did mean that sometimes the exact utility of an idea is unclear at the point it is introduced. Nevertheless, I think this book would provide a useful exposition to type-level programming concepts.

The book is pitched at intermediate Haskellers, with a friendly style that serves to make the ideas reasonably accessible. Knowledge of concepts like functors and monads is assumed, but this is likely to be familiar territory to anybody who considers themselves an ‘intermediate’ at Haskell. The introduction states that ‘the target reader should have a

*https://leanpub.com/thinking-with-types, LeanPub.

healthy sense of unease about the programs they write', however, and I think this is important: without this sense of unease, there will be little motivation to understand the concepts presented. In other words, the reader should have written enough Haskell to know that ordinary use of the type system does not prevent all sins.

I did take some issue with the explanation of the Curry–Howard correspondence, which the book presents as being between algebra (of numbers), logic and types. Technically, the Curry–Howard correspondence refers only to the relationship between logic and types; the correspondence between numerical operations and type-level operations has more to do with cardinal arithmetic. In any case, this section feels a little out of place; given that there is almost no material in the book pertaining to proofs, the relationship between proofs and programs seems little more than a side note.

Another technical error is that the continuation monad as presented is not the usual continuation monad: the standard continuation monad is not polymorphic. What is presented in the book is actually a version of the codensity monad. This may seem like a minor point, but the difference is significant, as the codensity monad does not permit the callCC operation that is the characteristic feature of the continuation monad. Given the importance of both continuations and codensity, I think that this is a point worth getting right.

I thought the exercises in the book, while generally good, were a little unevenly distributed, and I was surprised that the first major example, that of printf, was accompanied by no exercises at all. Overall, I would have liked to see more exercises, especially ones that directly involve type-level programming. I also found the bibliography a bit lacking. As I stated before, many of these ideas were originally introduced in Haskell Symposium papers; I would have liked to see these references provided for the interested reader to follow up on.

Overall, I think the strengths of this book outweigh its weaknesses. This text is timely, perhaps even a little overdue: a clear, integrated explanation of type-level programming has been needed for some time. The book's preface states that it was originally conceived as a series of blog posts, and the text has that kind of flavour to it. Many of the chapters could easily stand relatively independently of one another. Some might consider this a weakness, but I think it is actually a strength, making the ideas less intimidating than they might otherwise be. This book is not for everyone—beginners would find it hard to follow, and experts would likely already be familiar with the material—but for a certain type of reader, I believe it would be very useful.

J. HACKETT
Functional Programming Lab, School of Computer Science,
University of Nottingham
E-mail: jennifer.hackett@nottingham.ac.uk