JFP **21** (1): 115–117, 2011. © Cambridge University Press 2010

Book review

Review: Lambda-Calculus and Combinators: An Introduction, Second Edition by J. R. Hindley and J. P. Seldin
doi:10.1017/S0956796810000237

This book is the third revision of a classic overview of the lambda-calculus and associated combinators by two major contributors to the field. I was interested in it for the purposes of self-study, and was particularly looking for insights into the design of systems of combinators, such as those used in Turner's compilation scheme (1979), Smullyan's famous birds (1990), and Hughes's arrows (2000). Even more tantalising are those that Smullyan uses to encode a form of self-reference (1992), which clearly leads away from the lambda-calculus, building on the tradition of using combinators as a foundation for logical enterprises.

The authors are determined to make the material accessible and interesting. This is critical in the first third as motivating the selection of combinators is tough, and the central idea of completely eliminating variables is not immediately persuasive from a functional programmer's perspective. This part of the book shows the basic properties of the calculii using syntactic methods that are very accessible to people coming from computer science, but one almost certainly needs to have an application in mind, or perhaps to have seen how binders complicate formal proofs in the lambda-calculus, not to be lost in a sea of technical results. That so much of the theory has an operational flavour explains its utility in reasoning about compilers for functional languages, especially before Plotkin's structural operational semantics (SOS) revoluton of the early 1980s.

The use of the traditional **SKI** combinators for a basis is well-motivated by the early presentation of an abstraction result in Section 2C that effectively shows how variables can be handled on a meta-level. The book really shines here, and it is something of a let down when later combinators, such as those for Church arithmetic, are magicked from thin air. Patience pays off though, such as when they explain why **I** is included in their combinatory basis in Remark 8.23. These sorts of comments are highly valuable to the student for they show the right perspective to judge these technicalities, especially when there is a huge distance between the initial carefully-crafted definitions and the advantage they confer.

I have no doubt that generations of mathematical semantics students have benefited from this particularly clear presentation of combinator theory at an elementary level. However, for a book to be useful for self-study, it must be reasonably self-contained. Clearly this conflicts with a broad treatment, and leads almost inevitably to relegating the detailed treatment of many topics to the literature, which is also not without its problems.

To take a concrete example, it is well-known within the functional programming community that call-by-name is a complete reduction strategy, viz that if a term has a normal form then we can expect such an evaluator to find it. That this is so is summarised in theorem 3.22 (the quasi-leftmost reduction theorem), but explaining why it is so is deferred to theorem 13.2.6 of Barendregt (1984) and a remark that it hinges on a standardisation theorem. Barendregt's text uses different notation, and those results build on even more foundational ones, so our query results in a large detour from our original textbook.

In this and some other instances, a proof sketch would have been invaluable, even if it came at the cost of breadth; often one is satisfied with just the key insights of these proofs, especially if one has no particular interest in adapting them to other ends. The best exemplar of this approach that I know of is Sipser's computability and complexity text (1996). Perhaps most vexing here are those results where the proof has been elided, but the authors still comment on it; by that time the reader has learnt that these observations provide the real value of this text, and so feels compelled to seek out the references.

Similarly the cornerstone middle chapters 10–12 on type systems à la Church and à la Curry are weakened by their excessive deferral of proof to Barendregt's encyclopedia, and lead me to marvel once again at the masterful balance that Pierce (2002) struck when covering similar material. Chapter 13 on pure type systems may be distinctly useful, however, and the historical remarks are always fascinating.

Missing from this edition is a deeper treatment of how combinators have been used as foundations for formal logic, particularly the type-free higher-order systems studied by Curry. They have never seemed too appealing for use in mechanical proof assistants, largely because they are so user-unfriendly; in essence, combinators make the implicit plumbing of the lambda-calculus explicit, which often obscures the nub of an expression. One might wonder how more recent approaches to bound variables (De Bruijn indices and nominal terms, to name just two) relate to these variable-free calculi, and so a broader perspective on contemporary research, along the lines of what Pierce presented for types, would have been invaluable.

Also missing is much discussion about expanding the calculus in novel ways, such as adding Smullyan's quotation combinators; this text handles self-reference only using the traditional **Y** combinator in chapters 3 and 4. Some new work by Barry Jay and Thomas Given-Wilson (2010)[1] proposes a combinator that "factorises" its argument, demonstrating that even now there are new ways of thinking about combinatory completeness. The first part of this book is an excellent departure point for that paper, providing a way of assessing the tradeoffs they made, especially the sacrifice of extensionality for the extra expressiveness of pattern matching.

That combinators are a useful program-structuring concept is not obvious from this text as it does not treat the generalised structures underpinning arrows and monads, or supercombinators and such like. Compiler authors are likely better served by Peyton Jones's classic (1987).

The final third of the book discusses models for the lambda calculus and the system of combinators. The material is presented well but suffers from a lack of motivation, as no applications are presented. The summary of alternative models that closes the book does not explain how they improve on those shown here. Incidentally the best treatment I have yet found of the inverse-limit construction due to Scott, generalised to solve recursive domain equations, is contained in the notes by Bos & Hemerik (1988).

The two earlier versions of this text were reviewed by a pair of legends of the field: Barendregt (1973), and Krivine (1988). Barendregt had a series of technical reservations about this book, some of which have been addressed by simply discarding the material regarding combinators as a logical foundation. Krivine observes that the choice of topics and concepts is quite subjective, and he was doubtlessly alluding to the absence of a treatment of abstract machines for evaluating lambda expressions amongst other things. The hazard of having people of this calibre reviewing a textbook is that they know the literature too well, and don't experience the student's frustration that excessive referentiality brings. Educators may find the chapter-by-chapter review by van Raamsdonk (2009) more useful.

Without doubt this is a valuable treatment of a venerable topic that still rewards those who understand it. The authors successfully promulgate their tradition, and that is certainly more important than providing full proofs for every result. One comes away feeling that they do indeed care for their combinators according to the rules put forth in Appendix 4.

[1] Discussed at Lambda the Ultimate: `http://lambda-the-ultimate.org/node/3993`

Acknowledgements

Thanks to Ben Lippmeier and Bernie Pope for helpful feedback on this review.

References

Barendregt, H. P. (1973) Review of *Introduction to Combinatory Logic* by J. R. Hindley, B. Lercher and J. P. Seldin. *J. Symb. Logic*, **38**(3): 518.

Barendregt, H. P. (1984) *The Lambda Calculus: Its Syntax and Semantics (revised edition)*. North-Holland.

Bos, R., & Hemerik, C. (1988) *An Introduction to the Category-Theoretic Solution of Recursive Domain Equations*. Tech. Rep. Computing Science Notes No. 88-15. Technische Universiteit Eindhoven.

Hughes, J. (2000) Generalising Monads to Arrows. *Sci. Comput. Program.*, **37**: 67–111.

Krivine, J. L. (1988) Review of *Introduction to Combinators and Lambda-Calculus* by J. R. Hindley and J. P. Seldin, *J. Symb. Log.*, **53**(3): 985–986.

Pierce, B. C. (2002) *Types and Programming Languages*. MIT Press.

Sipser, M. (1996) *Introduction to the Theory of Computation*. International Thomson Publishing.

Smullyan, R. M. (1990) *To Mock a Mockingbird*. Oxford University Press.

Smullyan, R. M. (1992) *Satan, Cantor, And Infinity and Other Mind-Boggling Puzzles*. Knopf.

Turner, D. A. (1979) A new implementation technique for applicative languages. *Softw. – Prac. Exp.*, **9**(1): 31–49.

van Raamsdonk, F. (2009) Review of *Lambda-Calculus and Combinators, An Introduction, Second Edition*, J. R. Hindley and J. P. Seldin. *Theory Prac.Logic Program.*, **9**(2): 239–243.

PETER GAMMIE
The Australian National University,
Canberra ACT 0200, Australia
Peter.Gammie@anu.edu.au