Computational Semantics with Functional Programming, by Jan van Eijck and Christina Unger.
doi: 10.1017/S0956796814000057

Context. This book intersects two disciplines: functional programming and computational linguistics (the study of natural language using computational techniques). This review is from a functional programming perspective.

In computational linguistics, the approach of *Montague semantics* (named after its pioneer Richard Montague) uses mathematics and logic as a metatheory for a mathematically precise semantics (Janssen, 2012). The core principle is that there is 'no important theoretical difference between natural languages and the artificial languages of logicians' (Montague, 1970). Central to the approach is a compositional, high-order semantics, with the λ-calculus and predicate logic as foundations (see example in Figure 1 on the facing page). It is within this framework that Van Eijck & Unger (2010) write *Computational Semantics with Functional Programming*. Their thesis: Typed functional languages can naturally express Montague semantics, providing the benefits of a rich, executable metalanguage. From this basis they provide a guidebook through the core principles of the *Montagovian* approach and subsequent research, written in the languages of predicate logic and Haskell.

Overview. While the book is interdisciplinary between functional programming and computational semantics, there is a slight bias towards the latter. The introductory material focusses more on logic, the λ-calculus, and Haskell than on natural language semantics. From a functional programmer's perspective, most of the natural language concepts are however easily understood from the examples and definitions.

Its 13 chapters span roughly 400 pages, about half of which are tutorial, explaining the mathematical and programming basis. The core aspects of the Montagovian approach are introduced, providing a survey. Chapters on the set theory, the simply typed λ-calculus, induction, polymorphism, Haskell, and parser combinators are already familiar to most functional programmers. Small examples with a linguistic flavour appear throughout these chapters, with larger examples uniting the introductory material, such as a model checker for predicate logic and an inference engine over first-order propositions written in English.

The second half applies the Montagovian principles using higher-order predicate logic combinators to more complex linguistic phenomena, such as quantification, context dependence, pronoun scoping, and the effects of common knowledge on meaning.

The book does not offer any new research results. Instead, it overviews various concepts of computational semantics from the literature, using Haskell to give a complete account, which can then be experimented with. References to key results over the last 40 years are provided throughout, giving a useful starting point for further study.

The technical material is sound and the writing is mostly easy to follow. However, particularly in the second half, the book would benefit from more discussion on how chapters relate to each other and to the wider research field.

Complete source code is provided throughout and is available free online without purchasing the book[1] (although the online version lacks any documentation or comments, which are in line in the book's text). The book's index includes function names from code examples, and is extremely helpful, given the backward dependencies on code presented in earlier chapters.

Concepts and ideas. Figure 1 on the next page gives an example of *compositional* interpretation of a sentence fragment (based on Chapter 7) comprising a *noun phrase* (NP), a *determiner*

[1] http://www.computational-semantics.eu/

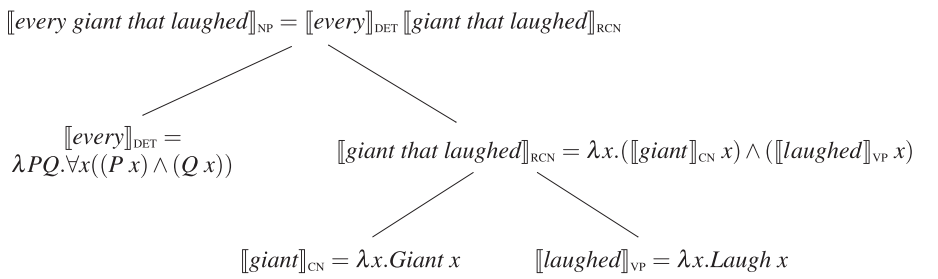

Fig. 1. Example of Montague-style semantics for a sentence fragment (based on chapter 7 of the book), where $[\![every\ giant\ that\ laughed]\!]_{NP} = \lambda Q.\forall x(Giant\ x \wedge Laugh\ x \wedge Q\ x)$.

(DET), a *relative clause* (RCN), a *common noun* (CN), and a *verb phrase* (VP). Figure 2 shows relevant fragments (from the book) of the Haskell interpretation for this example.

The key concept of this book (and the Montagovian approach) is *compositionality*, and maintaining this compositionality even for more complex language features, which, on the surface, appear to be non-compositional. Haskell serves the development well in this area.

One of the most interesting approaches is the use of continuations to denote linguistic contexts, in the sense of sentences with a 'hole'. This is used to describe different evaluation orders (as is done similarly in programming language semantics) for quantifiers and explains how ambiguities arise from different readings of quantifier scoping in a sentence.

A link is made between seemingly non-compositional linguistic phenomena and side-effecting computations, as developed by Shan (2005) using monads for natural language semantics. While the composition of impure computations using monads is well known in semantics and functional programming, this idea is not developed further. The book provides a good starting point for considering whether further programming language semantics results can be used for natural language semantics. For example, since *comonads* are used to structure context-dependent computations, e.g. Uustalu & Vene (2008), can they be used to abstract various forms of context in natural language semantics?

Discovering new abstractions and common idioms is a common activity in functional programming (both development and research). The advanced type systems found in many functional languages, particularly Haskell, help to organise and describe such abstractions. While the book makes use of well-known abstractions from the Montagovian approach (e.g. using parametric polymorphic definitions of relations), it does not extend these with the additional power provided by modern Haskell extensions. All of the examples are written using a simple subset of Haskell features: functions, pattern matching, algebraic data types, and parametric polymorphism. Type classes are used occasionally for defining custom *Show* instances. This is appropriate for the natural semantics audience, but there is a plenty of room for the advanced features of Haskell to be leveraged, particularly as a way of exposing new abstractions.

For example, the authors define mutually recursive type- and term-level functions for *intensionalizing* a semantics (lifting all entities and propositions to take an additional world parameter) and inversely *extensionalizing* (distributing a single world parameter to all subterms). The transformations are described on types and terms, and a proof is given that these transformations form an isomorphism. The chapter then specialises these operations to different types in the semantics, writing out the code for each instance. Instead, this could be straightforwardly implemented generically, once for all types, using type families (Chakravarty *et al*., 2005) (for the type functions) and type classes (for the overloaded value functions) in Haskell.

Conclusion. Using a programming language as a metalanguage is a standard approach in programming language research. It is encouraging to see this book applying the same principle

https://doi.org/10.1017/S0956796814000057 Published online by Cambridge University Press

```
intNP :: NP -> (Entity -> Bool) -> Bool
intNP (NP2 det rcn) = (intDET det) (intRCN rcn)
...

intDET :: DET -> (Entity -> Bool) -> (Entity -> Bool) -> Bool
intDET Every p q = all q (filter p entities)
...

intRCN :: RCN -> Entity -> Bool
intRCN (RCN1 cn _ vp) = \e -> ((intCN cn e) && (intVP vp e))
...
intVP Laughed = \x -> laugh x
...
intCN Giant = \x -> giant x
```

Fig. 2. Fragments of the Haskell implementation for the Montague semantics of Figure 1.

outside the field: using a programming language as a metalanguage for natural language semantics, reaping the benefits of the type checker and runtime. As the authors put it, 'your programming efforts will give you immediate feedback on your linguistic theories' (Van Eijck & Unger, 2010; p. 11). This book is therefore highly valuable for those natural language semanticists who ascribe to compositional, Montagovian approaches to semantics. It should be noted that compositional approaches are not universally accepted by linguists (see, for example, a brief survey of arguments against compositionality in Szabó, 2013).

For functional programming researchers, this book is a helpful starting point for anyone looking to get involved in natural language semantics, or at least looking for information on the Montagovian approach. There are many tantilising threads to follow and, as described briefly above, many areas of the book for which advanced, modern functional programming techniques apply. In addition, for those involved in language design, this book may provide some food for thought on how to add more natural language like features to an artificial language.

Acknowledgments

Thanks to Alan Mycroft, Tomas Petricek, and Martin Szummer for their comments on this review.

References

Chakravarty, M. M. T., Keller, G., Jones, S. P. & Marlow, S. (2005) Associated types with class. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'05)*, pp. 1–13.

Janssen, Theo M. V. (2012, Winter) Montague semantics. In *The Stanford Encyclopedia of Philosophy*, Zalta, E. N. (ed). Stanford, UK: Stanford University.

Montague, R. (1970) Universal grammar. *Theoria* **36**(3), 373–398.

Shan, C.-c. (2005) *Linguistic Side Effects.* PhD dissertation, Harvard University, Harvard, MA.

Szabó, Z. G. (2013, Fall) Compositionality. In *The Stanford Encyclopedia of Philosophy*, Zalta, E. N. (ed) .

Uustalu, T. & Vene, V. (2008) Comonadic notions of computation. *Electron. Notes Theor. Comput. Sci.* **203**(5), 263–284.

Van Eijck, J. & Unger, C. (2010) *Computational Semantics with Functional Programming*, 1st edn. Cambridge, UK: Cambridge University Press.

DOMINIC ORCHARD
Computer Laboratory, University of Cambridge, Cambridge, UK
e-mail: dominic.orchard@cl.cam.ac.uk

https://doi.org/10.1017/S0956796814000057 Published online by Cambridge University Press