basic uses, SMIL is far from simple. The timing model of SMIL 2.0, for example, is intricate and complex, and this book takes good care to subdivide the treatment of this topic so that its various aspects are introduced in an understandable but at the same time usable fashion. Thus, these sections of the nook will be of equal interest both to the reader who wishes to apply SMIL in producing powerful artifacts and to reader whose interest lies in the linguistic, formal aspects of temporal aspects of multimedia. The same approach is successfully used for other aspects of SMIL, including topics involving Layout, and Content Control.

Despite the fact that SMIL is not intended for media content creation, SMIL 2.0 incorporates a powerful animation module, permitting multimedia presentations to include a variety of cartoon-like items. This topic is well covered in chapter 15 the book, but it would have been preferable for the reader to be exposed to and to use aspects of animation earlier on. Animation is viewed by many creators as an central aspect of multimedia; indeed for many, multimedia *is* animation and since the ability to specify a wide variety of animations in a relatively simple, uniform fashion is one of the attractive features of SMIL 2.0, the reader may well be frustrated by this late treatment.

A good deal of attention is paid in the book to existing SMIL editing and rendering software, so that the reader is made well aware of which systems are available for download and use. This topic is discussed in Chapter 1, so that the reader is in a position to acquire and install appropriate software is when beginning to learn the SMIL language. Further reference to tools and players is made as the book progresses. Such information tends, of course, to be very dynamic in nature, as new products are created, new versions of existing products are released, and so on. This is a further reason why it is essential that the book's website be kept strictly up to date, which it appears not to be at the time of writing. Furthermore, many SMIL users, especially in their initial stages, are likely to want to make use of the XHTML+SMIL Profile, since this is readily available in most existing Internet Explorer browsers. This profile presents, at least from the user's perspective, a number of significant departures from "standard" SMIL, and it would have been helpful for the book to include more material on this version of SMIL.

An important aspect of SMIL relates to the matter of accessibility. SMIL is intended for the specification of artifacts which can run both on powerful computers and on small hand-held devices, or which are suitable for readers who are unable to perceive particular media types. This matter is dealt with late on in the book and somewhat tersely, in Chapter 17; it deserves more thorough exposure.

Overall the book is extremely well written and is a joy to read. Despite the inclusion of some rudimentary, introductory material, the reader is treated as intelligent and knowledgeable and the style of writing and presentation are informative, thorough, and leave little to be desired. In addition to the numerous examples referred to earlier, the book provides clear and useful illustrative descriptions of the structure of most features of SMIL, as well as a useful set of references to related material. Surprisingly, the authors make no specific claim about the use of this text by students; it is admirably suited to such a purpose and would be entirely suitable for a senior undergraduate, or more likely postgraduate, class.

Peter King
University of Manitoba, Canada

Programming Languages and Operational Semantics by Maribel Fernández, King's College Publications, 2004, ISBN 0954300637.
doi:10.1017/S0956796807006272

This slim volume by Maribel Fernáandez is Volume 1 of King's College Publications new 'Texts in Computing' series, edited by Ian Mackie. The book has been developed from notes for a second year undergraduate course at King's College London, taught by the author. One

of the book's aims is to present and compare alternative programming paradigms (specifically: imperative, functional and logic programming), using simple illustrative languages in each case. A second is to provide an introduction to operational semantics. Many of the example languages in the book are given formal operational semantics in the structural operational semantics style of Plotkin.

The book begins with an introductory chapter which reviews basic aspects of programming languages including the differing roles of syntax, semantics and implementation. Semantics, which is a central concern of the book, is introduced as defining "how [programs] behave when they are executed on a computer." This is reasonable as a first approximation for undergraduates, but it is a pity that the inevitable and desirable feature that semantics is a well-chosen abstraction from actual program behaviour is not discussed. The chapter ends with a quick review of the mathematical backround needed for understanding the remainder of the book. This comprises a short section on transition systems, which will later be used in specifying operational semantics, and a section reviewing proof by induction in its many guises. The latter section is sufficient as a brief review for the purposes of this book, but a typical undergraduate reader would benefit from having previously seen a more thorough treatment of the material.

The main body of the book is split into three parts, each consisting of two chapters.

Part I is an introduction to imperative languages. Its first chapter gives a quick review of general concepts that occur in imperative languages (e.g. different kinds of variable and their scoping). A reader unfamiliar with such concepts and with the languages used to illustrate them will probably find the treatment too brief. For those with more background knowledge, it is a useful overview.

The next chapter introduces formal operational semantics, using a standard simple 'while' language as the running example. Three semantics are given, an abstract machine, and two forms of structural operational semantics: small step and big step. Unfortunately, the more advanced language concepts discussed in the previous chapter are not treated formally, with the exception of a big step semantics for local variables and references, which apparently relies on the unexplained assumption that locations are given as numerical addresses.

Part II is an introduction to functional languages. Again, its first chapter explains the general features of functional languages, using Haskell for the examples. Overall, the discussion is clear, although there is an occasional slip (most notably, the eager evaluation of let expressions described on p. 67 does not agree with Haskell's lazy evaluation of such expressions). The next chapter is devoted to operational semantics, this time only big-step structural operational semantics, but with variants for call-by-value and call- by-name. First, a very simple language SFUN is considered, which allows only recursive definitions of first-order functions. Next, an extension FUN is defined that includes local definitions of both values and functions (let and let fun expressions). This exhibits some curious and not altogether desirable features. First, there is no way of producing values of function type, so the "functions as first-class citizens" mantra of functional programming is not addressed. Second, the given operational semantics employs dynamic rather than static binding, as a result of which the language is not even type sound. It is left as an exercise for the reader to fix this problem, but it would have been much better to have used static binding as the default (the solution is easier than the hint in the book suggests, rather than storing environments, one just needs to be careful about freshly renaming variables as their bindings are removed) and left it as an exercise for interested readers to discover for themselves the potential disastrous consequences of dynamic binding.

Part III is an introduction to logic programming, using Prolog for the examples. Again, the general concepts are introduced in the first chapter, and the operational behaviour described in the second. The discussion is restricted to pure Horn clauses using SLD resolution, so the complexities of Prolog's additional control structures (such as cut) are (rightly) avoided. It is worth commenting that, in contrast to the rest of the book, the operational semantics is defined informally. As a result, Part III is the most readable section of the book. However, it would have been interesting to have seen some discussion about how such an informal

presentation addresses the problems of imprecision and incompleteness, raised in Chapter 1 as potential defects of informal semantics in general.

This book will mainly appeal to undergraduate readers who are taking courses on formal semantics. It could serve as the main textbook for a low level course, provided that the lecturer is happy with the selection of material. Many, one suspects, would prefer to include more material in a course (e.g. object-oriented programming, concurrency), possibly at the expense of logic programming. Nonetheless this book could still be a useful source material for parts of such a course. In spite of the inexplicable choice of presenting a formal semantics for dynamic binding in Chapter 5, the book is, on the whole, nicely written and quite readable.

Alex Simpson

The Haskell School of Expression by Paul Hudak, Cambridge Univerity Press, 2000.[1]

doi:10.1017/S0956796807006284

As the title implies, The Haskell School of Expression (SOE) introduces functional programming through the Haskell programming language and through the use of graphics and music. It serves as an effective introduction to both the language and the concepts behind functional programming. This text was published in 2000, but since Haskell 98 is the current standard, it is still a very relevant book.

Haskell's standardization process gives us a window into two different facets of the community: Haskell is designed to be both a stable, standardized language (called Haskell 98), and a platform for experimentation in cutting-edge programming language research. So though we have a standard from 1998, the implementations (both compilers and interpreters) are continually evolving to implement new, experimental features which may or may not make it into the next standard.

For instance, the Glasgow Haskell Compiler has implemented a meta-programming environment called Template Haskell. Haskell is also easy to extend in directions that don't change the language itself, through the use of "Embedded Domain Specific Languages" (EDSLs) such as WASH for web authoring, Parsec for parsing, and Dance (more of Paul Hudak's work) for controlling humanoid robots.

Before we get too far, I should offer a disclaimer. The Haskell community is rather small, and if you scour the net, you may find conversations between myself and Paul Hudak or folks in his research group, since I use some of their software. That said, I don't work directly with Hudak or his research group.

In fact, the small size of the Haskell community is a useful feature. It is very easy to get involved, and folks are always willing to help newbies learn, since we love sharing what we know. You may even find that if you post a question about an exercise in SOE, you'll get a reply from the author himself.

I consider this book to be written in a "tutorial" style. It walks the reader through the building of applications, but doesn't skimp on the concepts (indeed, the chapters are meant to alternate between "concepts" and "applications.") In some ways, the code examples make it a little difficult to jump around, since you are expected to build upon previous code. The web site provides code, however, so you can always grab that and use it to fill in the missing pieces.

For readers who wish to use this book as a tutorial, and to implement all of the examples (which is highly recommended), I suggest that you grab the Hugs interpreter and read the

[1] A version of this review first appeared on Slashdot.