The Haskell School of Music

This textbook explores the fundamentals of computer music and functional programming through the Haskell programming language. Functional programming is typically considered difficult to learn. This introduction in the context of creating music will allow students and professionals with a musical inclination to leverage their experience to help understand concepts that might be intimidating in more traditional computer science settings. Conversely, the book opens the door for programmers to interact with music by using a medium that is familiar to them.

Readers will learn how to use the Euterpea library for Haskell (www.euterpea.com) to represent and create their own music with code, without the need for other music software. The book explores common paradigms used in algorithmic music composition, such as stochastic generation, musical grammars, self-similarity, and real-time interactive systems. Other topics covered include the basics of signal-based systems in Haskell, sound synthesis, and virtual instrument design.

PAUL HUDAK was a professor of computer science at Yale University, Connecticut, from 1982 to 2015. He was best known for his contributions to the development of the Haskell programming language. A skilled saxophonist and jazz musician, Hudak had used a combination of his enthusiasm for music and computer science to create the Euterpea library for representing music in Haskell.

DONYA QUICK is Research Assistant Professor of Music and Computation at Stevens Institute of Technology, New Jersey. Her research explores the intersection of artificial intelligence and computational linguistics with music, and includes working on an automated composition system called Kulitta. In addition, she is also involved in the MUSICA project for interactive improvisation and composition by conversion, which is part of the DAPRA Communicating with Computers program.

The Haskell School of Music

From Signals to Symphonies

PAUL HUDAK

DONYA QUICK
Stevens Institute of Technology

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India

79 Anson Road, #06–04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University’s mission by disseminating knowledge in the pursuit of
education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781108416757
DOI: 10.1017/9781108241861

First published 2018

Printed in the United States of America by Sheridan Books, Inc.

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data
Names: Hudak, Paul, author. | Quick, Donya, author.
Title: The Haskell school of music : from signals to symphonies / Paul Hudak, Donya Quick.
Description: Cambridge, United Kingdom ; New York, NY :
Cambridge University Press, 2018. | Includes bibliographical references and index.
Identifiers: LCCN 2018016879 | ISBN 9781108416757 (hardback : alk. paper)
Subjects: LCSH: Haskell (Computer program language) | Computer music–Instruction
and study. | Functional programming (Computer science)
Classification: LCC ML74.4.H37 H84 2018 | DDC 781.3/45133–dc23
LC record available at https://lccn.loc.gov/2018016879

ISBN 978-1-108-41675-7 Hardback

Cambridge University Press has no responsibility for the persistence or accuracy
of URLs for external or third-party internet websites referred to in this publication
and does not guarantee that any content on such websites is, or will remain,
accurate or appropriate.

Contents

Cambridge University Press
978-1-108-41675-7 — The Haskell School of Music
Paul Hudak , Donya Quick
Frontmatter
More Information

Preface

There is a certain mind-set, a certain viewpoint of the world, and a certain approach to problem solving that collectively work best when programming in Haskell (this is true for any programming paradigm). If you teach only Haskell language details to a C programmer, he or she is likely to write ugly, incomprehensible functional programs. But if you teach how to think differently, how to see problems in a different light, functional solutions will come easily, and elegant Haskell programs will result.

Music has many ties to mathematics. Combining the elegant mathematical nature of Haskell with that of music is as natural as singing a nursery tune. Using a high-level language to express musical ideas is, of course, not new. But Haskell is unique in its insistence on purity (no side effects), and this alone makes it particularly suitable for expressing musical ideas. By focusing on *what* a musical entity is, rather than on *how* to create it, we allow musical ideas to take their natural form as Haskell expressions. Haskell's many abstraction mechanisms allow us to write computer music programs that are elegant, concise, yet powerful. We will consistently attempt to let the music express itself as naturally as possible, without encoding it in terms of irrelevant language details.

Of course, the ultimate goal of this book is not just to teach computer music concepts. Along the way you will also learn Haskell. There is no limit to what one might wish to do with computer music, and therefore the better you are at programming, the more success you will have. Many languages designed specifically for computer music – although fun to work with, easy to use, and cute in concept – face the danger of being too limited in expressiveness.

You do not need to know much, if any, music theory to read this book, and you do not need to play an instrument. Of course, the more you know about music, the more you will be able to apply the concepts learned in this text in musically creative ways.

This book's general approach to introducing computer music concepts is to first provide an intuitive explanation, then a mathematically rigorous definition, and finally fully executable Haskell code. It will often be the case that there is a close correspondence between the mathematical definition and the Haskell code. Haskell features are introduced as they are needed, rather than all at once, and this interleaving of concepts and applications makes the material easier to digest.

Seasoned programmers having experience only with conventional imperative and/or object-oriented languages are encouraged to read this text with an open mind. Many things will be different, and will likely feel awkward. There will be a tendency to rely on old habits when writing new programs, and to ignore suggestions about how to approach things differently. If you can manage to resist those tendencies, you will have an enjoyable learning experience. Those who succeed in this process often find that many ideas about functional programming can be applied to imperative and object-oriented languages as well, and that their imperative coding style changes for the better.

The experienced programmer should also be patient with earlier topics, such as "syntax," "operator precedence," etc., since the intent is for this text to be readable by someone having only modest prior programming experience. With patience, the more advanced ideas will appear soon enough.

If you are a novice programmer, take your time with the book; work through the exercises, and don't rush things. If, however, you don't fully grasp an idea, feel free to move on, but try to reread difficult material at a later time when you have seen more examples of the concepts in action. For the most part, this is a "show-by-example" textbook, and you should try to execute as many of the programs in this text as you can, as well as every program that you write. Learn-by-doing is the corollary to show-by-example.

Finally, some section titles are prefaced with the parenthetical phrase "[Advanced]". These sections may be skipped upon first reading, especially if the focus is on learning computer music concepts, as opposed to programming concepts.

Prerequisites

Basic algebra and familiarity with a terminal-style environment (often called a command prompt in Windows) on your computer are also assumed as prerequisites in this text. Some prior introduction to computer science concepts and data structures (primarily lists and trees) is also strongly recommended.

This book is not a substitute for an introductory music theory course. Rather, it is intended primarily for programmers with at least a small amount of musical experience (such as having taken a music appreciation course in school or played an instrument at some point) who want to then explore music in the context of a functional programming environment. Examples of musical concepts that are considered prerequisites to this text are reading Western music notation, the naming scheme for musical pitches as letters and octave numbers, and the major and minor scales. That said, it is certainly not impossible to learn Haskell and the Euterpea library from this book as a complete musical novice – but you will likely need to consult other music-related resources to fill in the gaps as you go along using a dictionary of musical terms. A wide array of free music theory resources and tutorials for beginners are also freely available online. Links to some useful music references and tutorials can be found on the Euterpea website, www.euterpea.com.

Music Terminology

Some musical concepts have more than one term to refer to them, and which synonym is preferred differs by region. For example, the following terms are synonyms for note durations:

American English	**British English**
Double whole note	Breve
Whole note	Semibreve
Half note	Minim
Quarter note	Crotchet
Eight note	Quaver
Sixteenth note	Semiquaver

This book uses the American English versions of these musical terms. The reason for this is that they more closely mirror the mathematical relationships represented by the concepts they refer to, and they are also the basis for names of a number of values used in the software this text describes. The American English standard for naming note durations is both more common in computer music literature and easier to remember for those with limited musical experience – who may struggle to remember what a hemidemisemiquaver is.

Software

There are several implementations of Haskell, all available free on the Internet through the Haskell website, haskell.org. However, the one that has dominated all others, and on which Euterpea is based, is *GHC* [1], an easy-to-use and easy-to-install Haskell compiler and interpreter. GHC runs on a variety of platforms, including Windows, Linux, and Mac. The preferred way to install GHC is using *Haskell Platform* [2]. Once Haskell is installed, you will have access to what is referred to as the *Standard Prelude*, a collection of predefined definitions that are always available and do not need to be specially imported.

Two libraries are needed to code along with this textbook: Euterpea and HSoM. Euterpea is a language for representing musical structures in Haskell, and many of its features are covered in this book. HSoM is a supplemental library containing many of the longer code examples in the text and two additional features: support for modeling musical performance (Chapter 9) and music-related graphical widgets (Chapter 17).

Detailed setup information for Haskell Platform, Euterpea, and HSoM is available on the Euterpea website: www.euterpea.com. Please note: software setup details for Haskell Platform and the Euterpea library varies by architecture (32-bit vs 64-bit), operating system, and compiler version. As the exact setup details are subject to change with every new release of Euterpea's dependencies, please see www.euterpea.com for the most up-to-date installation instructions. While most installations go smoothly with the relatively simple instructions described in the next section, there are many potential differences from one machine to another that can complicate the process. The Euterpea website also contains troubleshooting information for commonly encountered installation problems.

Installation Instructions

The following setup instructions require an Internet connection.

- Download the appropriate version of Haskell Platform from www.haskell.org/platform/ and install it on your machine.
- Open a command prompt (Windows) or terminal (Mac/Linux) and run the following commands:
  ```
  cabal update
  cabal install Euterpea
  cabal install HSoM
  ```
- Mac and Linux users will also need to install a MIDI software synthesizer. Please see the Euterpea website for instructions on how to do this.

The Euterpea website also contains basic walkthroughs for getting started working with the GHC compiler and interpreter within a command prompt or terminal, loading source code files, and so on.

Quick References

Brief references for the more commonly used features of Haskell, Euterpea, and HSoM are listed in Appendices E, F, and G. These are intended to serve as a fast way to look up function and value names when you already know a bit about how to use them. A note to students: these few pages of ultra-condensed material are not a substitute for reading the chapters!

Coding and Debugging

Errors are an inevitable part of coding. The best way to minimize the number of errors you have to solve is to code a little bit and then immediately test what you've done. If it's broken, don't wait – fix it then and there! Never press onward and try to work on other things within a file that is broken elsewhere. The reason for this is that one simple error can end up masking others. When the compiler hits a serious problem, it *may not even look at the rest of your file*. As a result, continuing to code without resolving error messages often results in an explosion of new errors once the original one is fixed. You will save yourself a lot of grief by developing good habits of incremental development and not allowing errors to linger unsolved.

Coding *style* is also important. There are two reasons for this in Haskell. The first is that Haskell is *extremely* sensitive to white space characters. Do not mix spaces and tabs! Pick one and be consistent (spaces are typically recommended). Indentation matters, and a small misalignment can sometimes cause bizarre-looking error messages. Style is important, as is readability, both by other programmers and by yourself at a later date. Many novice programmers neglect good coding hygiene, which involves naming things well, laying out code cleanly, and documenting complicated parts of the code. This extra work may be tedious, but it is worthwhile. Coding large projects is often very much dependent on the immediate state of mind. Without that frame of reference, it's not impossible that you could find your own code to be impenetrable if you pick it up again later.

Cambridge University Press
978-1-108-41675-7 — The Haskell School of Music
Paul Hudak , Donya Quick
Frontmatter
More Information

Acknowledgments

I wish to thank my funding agencies – the National Science Foundation, the Defense Advanced Research Projects Agency, and Microsoft Research – for their generous support of research that contributed to the foundations of Euterpea. Yale University has provided me a stimulating and flexible environment to pursue my dreams for more than thirty years, and I am especially thankful for its recent support of the Computing and the Arts initiative.

Tom Makucevich, a talented computer music practitioner and composer in New Haven, was the original motivator, and first user, of Haskore, which preceded Euterpea. Watching him toil endlessly with low-level csound programs was simply too much for me to bear! Several undergraduate students at Yale contributed to the original design and implementation of Haskore. I would like to thank in particular the contributions of Syam Gadde and Bo Whong, who coauthored the original paper on Haskore. Additionally, Matt Zamec helped me greatly in the creation of HasSound.

I wish to thank my more recent graduate students, in particular Hai (Paul) Liu, Eric Cheng, Donya Quick, and Daniel Winograd-Cort, for their help in writing much of the code that constitutes the current Euterpea library. In addition, many students in my computer music classes at Yale provided valuable feedback through earlier drafts of the manuscript.

Finally, I wish to thank my wife, Cathy Van Dyke, my best friend and ardent supporter, whose love, patience, and understanding have helped me get through some bad times, and enjoy the good.

Happy Haskell Music Making!

Paul Hudak,
January 2012