

THINKING FUNCTIONALLY WITH HASKELL

Richard Bird is famed for the clarity and rigour of his writing. His new textbook, which introduces functional programming to students, emphasises fundamental techniques for reasoning mathematically about functional programs. By studying the underlying equational laws, the book enables students to apply calculational reasoning to their programs, both to understand their properties and to make them more efficient.

The book has been designed to fit a first- or second-year undergraduate course and is a thorough overhaul and replacement of his earlier textbooks. It features case studies in Sudoku and pretty-printing, and over 100 carefully selected exercises with solutions. This engaging text will be welcomed by students and teachers alike.

Cambridge University Press
978-1-107-08720-0 - Thinking Functionally with Haskell
Richard Bird
Frontmatter
[More information](#)

Cambridge University Press
978-1-107-08720-0 - Thinking Functionally with Haskell
Richard Bird
Frontmatter
[More information](#)

THINKING FUNCTIONALLY WITH HASKELL

RICHARD BIRD
University of Oxford



Cambridge University Press
978-1-107-08720-0 - Thinking Functionally with Haskell
Richard Bird
Frontmatter
[More information](#)

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

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/9781107087200

© Richard Bird 2015

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2015

3rd printing 2015

Printed in the United Kingdom by Clays, St Ives plc

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

Library of Congress Cataloguing in Publication data

Bird, Richard, 1943-

Thinking functionally with Haskell / Richard Bird, University of Oxford.

pages cm

ISBN 978-1-107-08720-0 (hardback) – ISBN 978-1-107-45264-0 (paperback)

1. Functional programming (Computer science) I. Title.

QA76.62.B573 2014

005.1'14–dc23

2014024954

ISBN 978-1-107-08720-0 Hardback

ISBN 978-1-107-45264-0 Paperback

Additional resources for this publication at www.cambridge.org/9781107087200

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

	<i>page ix</i>
<i>Preface</i>	
1 What is functional programming?	1
1.1 Functions and types	1
1.2 Functional composition	3
1.3 Example: common words	3
1.4 Example: numbers into words	7
1.5 The Haskell Platform	12
1.6 Exercises	14
1.7 Answers	17
1.8 Chapter notes	20
2 Expressions, types and values	22
2.1 A session with GHCi	22
2.2 Names and operators	25
2.3 Evaluation	27
2.4 Types and type classes	30
2.5 Printing values	33
2.6 Modules	35
2.7 Haskell layout	36
2.8 Exercises	37
2.9 Answers	42
2.10 Chapter notes	47
3 Numbers	49
3.1 The type class Num	49
3.2 Other numeric type classes	50
3.3 Computing floors	52
3.4 Natural numbers	56

vi	Contents	
	3.5 Exercises	59
	3.6 Answers	61
	3.7 Chapter notes	62
4	Lists	63
	4.1 List notation	63
	4.2 Enumerations	65
	4.3 List comprehensions	66
	4.4 Some basic operations	68
	4.5 Concatenation	69
	4.6 <code>concat</code> , <code>map</code> and <code>filter</code>	70
	4.7 <code>zip</code> and <code>zipWith</code>	73
	4.8 Common words, completed	75
	4.9 Exercises	77
	4.10 Answers	82
	4.11 Chapter notes	87
5	A simple Sudoku solver	89
	5.1 Specification	89
	5.2 Lawful program construction	95
	5.3 Pruning the matrix of choices	97
	5.4 Expanding a single cell	101
	5.5 Exercises	105
	5.6 Answers	107
	5.7 Chapter notes	109
6	Proofs	110
	6.1 Induction over natural numbers	110
	6.2 Induction over lists	113
	6.3 The function <code>foldr</code>	117
	6.4 The function <code>foldl</code>	122
	6.5 The function <code>scanl</code>	125
	6.6 The maximum segment sum	127
	6.7 Exercises	131
	6.8 Answers	135
	6.9 Chapter notes	144
7	Efficiency	145
	7.1 Lazy evaluation	145
	7.2 Controlling space	149
	7.3 Controlling time	154
	7.4 Analysing time	156
	7.5 Accumulating parameters	159

	Contents	vii
7.6	Tupling	164
7.7	Sorting	167
7.8	Exercises	172
7.9	Answers	175
7.10	Chapter notes	180
8	Pretty-printing	181
8.1	Setting the scene	181
8.2	Documents	183
8.3	A direct implementation	187
8.4	Examples	189
8.5	The best layout	191
8.6	A term representation	193
8.7	Exercises	199
8.8	Answers	203
8.9	Chapter notes	209
9	Infinite lists	210
9.1	Review	210
9.2	Cyclic lists	212
9.3	Infinite lists as limits	215
9.4	Paper–rock–scissors	221
9.5	Stream-based interaction	226
9.6	Doubly-linked lists	228
9.7	Exercises	231
9.8	Answers	234
9.9	Chapter notes	237
10	Imperative functional programming	239
10.1	The IO monad	239
10.2	More monads	244
10.3	The State monad	247
10.4	The ST monad	251
10.5	Mutable arrays	254
10.6	Immutable arrays	259
10.7	Exercises	263
10.8	Answers	267
10.9	Chapter notes	275
11	Parsing	276
11.1	Parsers as monads	276
11.2	Basic parsers	279
11.3	Choice and repetition	281

viii	Contents	
	11.4 Grammars and expressions	285
	11.5 Showing expressions	288
	11.6 Exercises	291
	11.7 Answers	294
	11.8 Chapter notes	297
12	A simple equational calculator	298
	12.1 Basic considerations	298
	12.2 Expressions	304
	12.3 Laws	310
	12.4 Calculations	312
	12.5 Rewrites	315
	12.6 Matchings	317
	12.7 Substitutions	319
	12.8 Testing the calculator	321
	12.9 Exercises	331
	12.10 Answers	333
	12.11 Chapter notes	337
	<i>Index</i>	338

Preface

The present book is a completely rewritten version of the second edition of my *Introduction to Functional Programming using Haskell* (Prentice Hall). The main changes are: a reorganisation of some introductory material to reflect the needs of a one or two term lecture course; a fresh set of case studies; and a collection of over 100 exercises that now actually contain answers. As before, no knowledge of computers or programming is assumed, so the material is suitable as a first course in computing.

Every author has his or her own drum to beat when writing a textbook, and the present one is no different. While there are now numerous books, tutorials, articles and blogs devoted to Haskell, few of them emphasise what seems to me the main reason why functional programming is the best thing since sliced bread: the ability to think mathematically about functional programs. And the mathematics involved is neither new nor difficult. Any student who has come to grips with, say, high-school trigonometry and has applied simple trigonometric laws and identities to simplify expressions involving sines and cosines (a typical example: express $\sin 3\alpha$ in terms of $\sin \alpha$) will quickly appreciate that a similar activity is being proposed for programming problems. And the payoff is there at the terminal: faster computations. Even after 30 years I still get a great deal of pleasure from writing down a simple, obvious, but inefficient way to solve a problem, applying some well-known equational laws, and coming up with another solution that is ten times faster. Well, if I'm lucky.

If the message of the last paragraph turns you off, if you are perpetually running away from the Mordor of Mathematics, then the present book is probably not for you. Probably, but not necessarily so (nobody likes to lose customers). There is still pleasure to be gained in learning a novel and exciting way to write programs. Even programmers who for one reason or another do not or cannot use Haskell

in their daily work, and certainly do not have the time to spend calculating better answers to their problems, have still been inspired by the enjoyment of learning Haskell and are hugely appreciative of its ability to express computational ideas and methods simply and briefly. In fact, the ability to express programming ideas in a purely functional style has been slowly incorporated into mainstream imperative programming languages, such as Python, Visual Basic, and C#.

One final but important point: Haskell is a large language and this book by no means covers all of it. It is not a reference guide to Haskell. Although details of the language appear on almost every page, especially in the earlier chapters, my primary intention is to convey the essence of functional programming, the idea of thinking functionally about programs, not to dwell too much on the particulars of one specific language. But over the years Haskell has absorbed and codified most of the ideas of functional programming expressed in earlier functional languages, such as SASL, KRC, Miranda, Orwell and Gofer, and it is difficult to resist the temptation to explain everything in terms of this one super-cool language.

Most of the programs recorded in this book can be found on the website

`www.cs.ox.ac.uk/publications/books/functional`

It is hoped to add more exercises (and answers), suggestions for projects, and so on, in due course. For more information about Haskell, the site `www.haskell.org` should be your first port of call.

Acknowledgements

The present book arose out of lecture notes I prepared based on the second edition. It has benefited enormously from the comments and suggestions by tutors and students. Others have emailed me with constructive comments and criticisms, or simply to point out typos and silly mistakes. These include: Nils Andersen, Ani Calinescu, Franklin Chen, Sharon Curtis, Martin Filby, Simon Finn, Jeroen Fokker, Maarten Fokkinga, Jeremy Gibbons, Robert Giegerich, Kevin Hammond, Ralf Hinze, Gerard Huet, Michael Hinchey, Tony Hoare, Iain Houston, John Hughes, Graham Hutton, Cezar Ionescu, Stephen Jarvis, Geraint Jones, Mark Jones, John Launchbury, Paul Licameli, David Lester, Iain MacCullum, Ursula Martin, Lambert Meertens, Erik Meijer, Quentin Miller, Oege de Moor, Chris Okasaki, Oskar Permvall, Simon Peyton Jones, Mark Ramaer, Hamilton Richards, Dan Russell, Don Sannella, Antony Simmons, Deepak D'Souza, John Spanondakis, Mike Spivey, Joe Stoy, Bernard Sufrin, Masato Takeichi, Peter Thiemann, David Turner, Colin Watson, and Stephen Wilson. In particular, Jeremy Gibbons, Bernard Sufrin

and José Pedro Magalhães have read drafts of the manuscript and suggested a number of corrections.

I would also like to thank David Tranah, my editor at CUP, for continued advice and support. My status now is emeritus professor at the Department of Computer Science at Oxford, and I would like to thank the department and its head, Bill Roscoe, for continuing to make facilities available.

Richard Bird

Exercises

Exercise A

Express $\sin 3\alpha$ in terms of $\sin \alpha$.

Answers

Answer to Exercise A

$$\begin{aligned}
 & \sin 3\alpha \\
 = & \quad \{\text{arithmetic}\} \\
 & \sin(2\alpha + \alpha) \\
 = & \quad \{\text{since } \sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta\} \\
 & \sin 2\alpha \cos \alpha + \cos 2\alpha \sin \alpha \\
 = & \quad \{\text{since } \sin 2\alpha = 2 \sin \alpha \cos \alpha\} \\
 & 2 \sin \alpha \cos^2 \alpha + \cos 2\alpha \sin \alpha \\
 = & \quad \{\text{since } \cos 2\alpha = \cos^2 \alpha - \sin^2 \alpha\} \\
 & 2 \sin \alpha \cos^2 \alpha + (\cos^2 \alpha - \sin^2 \alpha) \sin \alpha \\
 = & \quad \{\text{since } \sin^2 \alpha + \cos^2 \alpha = 1\} \\
 & \sin \alpha(3 - 4 \sin^2 \alpha)
 \end{aligned}$$

The above proof format was, I believe, invented by Wim Feijen. It will be used throughout the book.