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.