Dedication. For Sue, Nathan and Sarah.

CONTENTS

PREFACE TO THE SECOND EDITION

With each reprinting of this book, a dozen minor errors have silently disappeared. But a reprinting is no occasion for making improvements, however valuable, that would affect the page numbering: we should then have several slightly different, incompatible editions. An accumulation of major changes (and the Editor's urgings) have prompted this second edition.

As luck would have it, changes to ML have come about at the same time. ML has a new standard library and the language itself has been revised. It is worth stressing that the changes do not compromise ML's essential stability. Some obscure technical points have been simplified. Anomalies in the original definition have been corrected. Existing programs will run with few or no changes. The most visible changes are the new character type and a new set of top level library functions.

The new edition brings the book up to date and greatly improves the presentation. Modules are now introduced early — in Chapter 2 instead of Chapter 7 — and used throughout. This effects a change of emphasis, from data structures (say, binary search trees) to abstract types (say, dictionaries). A typical section introduces an abstract type and presents its ML signature. Then it explains the ideas underlying the implementation, and finally presents the code as an ML structure. Though reviewers have been kind to the first edition, many readers have requested such a restructuring.

The programs have not just been moved about, but rewritten. They now reflect modern thoughts on how to use modules. The `open` declaration, which obscures a program's modular structure, seldom appears. Functors are only used where necessary. Programs are now indented with greater care. This, together with the other changes, should make them much more readable than hitherto. They are also better: there is a faster merge sort and simpler, faster priority queues.

The new standard library would in any case have necessitated an early mention of modules. Although it entails changes to existing code, the new library brings ML firmly into the fold of realistic languages. The library has been designed, through a long process of consultation, to provide comprehensive support without needless complication. Its organization demonstrates the benefits

of ML modules. The string processing, input/output and system interface modules provide real gains in power.

The library forced much rewriting. Readers would hardly like to read about the function *foldleft* when the library includes a similar function called *foldl*. But these functions are not identical; the rewriting involved more than a change of name. Many sections that previously described useful functions now survey corresponding library structures.

The updated bibliography shows functional programming and ML used in a wide variety of applications. ML meets the requirements for building reliable systems. Software engineers expect a language to provide type safety, modularity, compile-time consistency checking and fault tolerance (exceptions). Thanks in part to the library, ML programs are portable. Commercially supported compilers offer increasing quality and efficiency. ML can now run as fast as C, especially in applications requiring complicated storage management. The title of this book, which has attracted some jibes, may well prove to be prophetic.

My greatest surprise was to see the first edition in the hands of beginning programmers, when the first page told them to look elsewhere. To help beginners I have added a few especially simple examples, and removed most references from the main text. The rewritten first chapter attempts to introducing basic programming concepts in a manner suitable both to beginners and to experienced C programmers. That is easier than it sounds: C does not attempt to give programmers a problem-solving environment, merely to dress up the underlying hardware. The first chapter still presupposes some basic knowledge of computers. Instructors may still wish to start with Chapter 2, with its simple on-line sessions.

At the end of the book is a list of suggested projects. They are intentionally vague; the first step in a major project is to analyse the requirements precisely. I hope to see ML increasingly adopted for project work. The choice of ML, especially over insecure languages like C, may eventually be recognized as a mark of professionalism.

I should like to thank everyone whose comments, advice or code made an impact on this edition. They include Matthew Arcus, Jon Fairbairn, Andy Gordon, Carl Gunter, Michael Hansen, Andrew Kennedy, David MacQueen, Brian Monahan, Arthur Norman, Chris Okasaki, John Reppy, Hans Rischel, Peter Sestoft, Mark Staples and Mads Tofte. Sestoft also gave me a pre-release of Moscow ML, incorporating library updates. Alison Woollatt of CUP coded the LaTeX class file. Franklin Chen and Namhyun Hur reported errors in previous printings.

PREFACE

This book originated in lectures on Standard ML and functional programming. It can still be regarded as a text on functional programming — one with a pragmatic orientation, in contrast to the rather idealistic books that are the norm — but it is primarily a guide to the effective use of ML. It even discusses ML's imperative features.

Some of the material requires an understanding of discrete mathematics: elementary logic and set theory. Readers will find it easier if they already have some programming experience, but this is not essential.

The book is a programming manual, not a reference manual; it covers the major aspects of ML without getting bogged down with every detail. It devotes some time to theoretical principles, but is mainly concerned with efficient algorithms and practical programming.

The organization reflects my experience with teaching. Higher-order functions appear late, in Chapter 5. They are usually introduced at the very beginning with some contrived example that only confuses students. Higher-order functions are conceptually difficult and require thorough preparation. This book begins with basic types, lists and trees. When higher-order functions are reached, a host of motivating examples is at hand.

The exercises vary greatly in difficulty. They are not intended for assessing students, but for providing practice, broadening the material and provoking discussion.

Overview of the book. Most chapters are devoted to aspects of ML. Chapter 1 introduces the ideas behind functional programming and surveys the history of ML. Chapters 2–5 cover the functional part of ML, including an introduction to modules. Basic types, lists, trees and higher-order functions are presented. Broader principles of functional programming are discussed.

Chapter 6 presents formal methods for reasoning about functional programs. If this seems to be a distraction from the main business of programming, consider that a program is worth little unless it is correct. Ease of formal reasoning is a major argument in favour of functional programming.

Chapter 7 covers modules in detail, including functors (modules with parameters). Chapter 8 covers ML's imperative features: references, arrays and input/output. The remainder of the book consists of extended examples. Chapter 9 presents a functional parser and a λ-calculus interpreter. Chapter 10 presents a theorem prover, a traditional ML application.

The book is full of examples. Some of these serve only to demonstrate some aspect of ML, but most are intended to be useful in themselves — sorting, functional arrays, priority queues, search algorithms, pretty printing. Please note: although I have tested these programs, they undoubtedly contain some errors.

Information and warning boxes. Technical asides, descriptions of library functions, and notes for further study appear from place to place. They are highlighted for the benefit of readers who wish to skip over them:

King Henry's claim. There is no bar to make against your highness' claim to France but this, which they produce from Pharamond, *In terram Salicam mulieres ne succedant*, 'No woman shall succeed in Salique land': which Salique land the French unjustly gloze to be the realm of France, and Pharamond the founder of this law and female bar. But their own authors faithfully affirm that the land Salique is in Germany . . .[1]

ML is not perfect. Certain pitfalls can allow a simple coding error to waste hours of a programmer's time. The new standard library introduces incompatibilities between old and new compilers. Warnings of possible hazards appear throughout the book. They look like this:

Beware the Duke of Gloucester. O Buckingham! take heed of yonder dog. Look, when he fawns, he bites; and when he bites, his venom tooth will rankle to the death. Have not to do with him, beware of him; Sin, Death, and Hell have set their marks on him, and all their ministers attend on him.

I hasten to add that nothing in ML can have consequences quite this dire. No fault in a program can corrupt the ML system itself. On the other hand, programmers must remember that even correct programs can do harm in the outside world.

How to get a Standard ML compiler. Because Standard ML is fairly new on the scene, many institutions will not have a compiler. The following is a partial list of existing Standard ML compilers, with contact addresses. The examples in this

[1] No technical aside in this book is as long as the Archbishop's speech, which extends to 62 lines.

book were developed under Moscow ML, Poly/ML and Standard ML of New Jersey. I have not tried the other compilers.

To obtain ***MLWorks***, contact Harlequin Limited, Barrington Hall, Barrington, Cambridge, CB2 5RG, England. Their email address is `web@harlequin.com`.

To obtain ***Moscow ML***, contact Peter Sestoft, Mathematical Section, Royal Veterinary and Agricultural University, Thorvaldsensvej 40, DK-1871 Frederiksberg C, Denmark. Or get the system from the World Wide Web:

```
http:/www.dina.kvl.dk/~sestoft/mosml.html
```

To obtain ***Poly/ML***, contact Abstract Hardware Ltd, 1 Brunel Science Park, Kingston Lane, Uxbridge, Middlesex, UB8 3PQ, England. Their email address is `lambda@ahl.co.uk`.

To obtain ***Poplog Standard ML***, contact Integral Solutions Ltd, Berk House, Basing View, Basingstoke, Hampshire, RG21 4RG, England. Their email address is `isl@isl.co.uk`.

To obtain ***Standard ML of New Jersey***, contact Andrew Appel, Computer Science Department, Princeton University, Princeton NJ 08544-2087, USA. Better still, fetch the files from the World Wide Web:

```
http://www.cs.princeton.edu/~appel/smlnj/
```

The programs in this book and answers to some exercises are available by email; my address is `lcp@cl.cam.ac.uk`. If possible, please use the World Wide Web; my home page is at

```
http://www.cl.cam.ac.uk/users/lcp/
```

Acknowledgements. The editor, David Tranah, assisted with all stages of the writing and suggested the title. Graham Birtwistle, Glenn Bruns and David Wolfram read the text carefully. Dave Berry, Simon Finn, Mike Fourman, Kent Karlsson, Robin Milner, Richard O'Keefe, Keith van Rijsbergen, Nick Rothwell, Mads Tofte, David N. Turner and the staff of Harlequin also commented on the text. Andrew Appel, Gavin Bierman, Phil Brabbin, Richard Brooksby, Guy Cousineau, Lal George, Mike Gordon, Martin Hansen, Darrell Kindred, Silvio Meira, Andrew Morris, Khalid Mughal, Tobias Nipkow, Kurt Olender, Allen Stoughton, Reuben Thomas, Ray Toal and Helen Wilson found errors in previous printings. Piete Brooks, John Carroll and Graham Titmus helped with the computers. I wish to thank Dave Matthews for developing Poly/ML, which was for many years the only efficient implementation of Standard ML.

Of the many works in the bibliography, Abelson and Sussman (1985), Bird

and Wadler (1988) and Burge (1975) have been especially helpful. Reade (1989) contains useful ideas for implementing lazy lists in ML.

The Science and Engineering Research Council has supported LCF and ML in numerous research grants over the past 20 years.

I wrote most of this book while on leave from the University of Cambridge. I am grateful to the Computer Laboratory and Clare College for granting leave, and to the University of Edinburgh for accommodating me for six months.

Finally, I should like to thank Sue for all she did to help, and for tolerating my daily accounts of the progress of every chapter.