

Cambridge University Press  
978-0-521-56543-1 - ML for the Working Programmer, Second Edition  
Lawrence C. Paulson  
Frontmatter  
[More information](#)

---

## **ML for the Working Programmer**

Cambridge University Press  
978-0-521-56543-1 - ML for the Working Programmer, Second Edition  
Lawrence C. Paulson  
Frontmatter  
[More information](#)

---

# ML for the Working Programmer

**2nd edition**

Lawrence C. Paulson  
*University of Cambridge*



Cambridge University Press  
978-0-521-56543-1 - ML for the Working Programmer, Second Edition  
Lawrence C. Paulson  
Frontmatter  
[More information](#)

---

CAMBRIDGE UNIVERSITY PRESS  
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore,  
São Paulo, Delhi, Dubai, Tokyo

Cambridge University Press  
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

[www.cambridge.org](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9780521565431](http://www.cambridge.org/9780521565431)

© Cambridge University Press 1991, 1996

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 edition published 1991  
First published in paperback (with corrections) 1992  
Reprinted 1993, 1995  
Second edition published 1996  
Eighth printing 2008

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

ISBN 978-0-521-57050-3 Hardback  
ISBN 978-0-521-56543-1 Paperback

Transferred to digital printing 2009

The cover illustration is taken from *Work* by Ford Madox Brown  
© Manchester City Art Galleries, and is reproduced with their permission.

*Trademarks.* Miranda is a trademark of Research Software Limited. Sun and SuperSPARC are trademarks of Sun Microsystems. Unix is a trademark of AT&T Bell Laboratories. Poplog is a trademark of the University of Sussex. MLWorks is a trademark of Harlequin Limited. DEC and PDP are trademarks of Digital Equipment Corporation.

*Dedication.* For Sue, Nathan and Sarah.

#### DISCLAIMER OF WARRANTY

**The programs listed in this publication are provided 'as is' without warranty of any kind. We make no warranties, express or implied, that the programs are free of error, or are consistent with any particular standard of merchantability, or that they will meet your requirements for any particular application. They should not be relied upon for solving a problem whose incorrect solution could result in injury to a person or loss of property. If you do use the programs or procedures in such a manner, it is at your own risk. The author and publisher disclaim all liability for direct, incidental or consequential damages resulting from your use of the programs, modules or functions in this book.**

## CONTENTS

<i>Preface to the Second Edition</i>	xiii
<i>Preface</i>	xv
<b>1 Standard ML</b>	<b>1</b>
Functional Programming	2
1.1 Expressions versus commands	2
1.2 Expressions in procedural programming languages	3
1.3 Storage management	5
1.4 Elements of a functional language	5
1.5 The efficiency of functional programming	9
Standard ML	11
1.6 The evolution of Standard ML	11
1.7 The ML tradition of theorem proving	12
1.8 The new standard library	13
1.9 ML and the working programmer	15
<b>2 Names, Functions and Types</b>	<b>17</b>
Chapter outline	18
Value declarations	18
2.1 Naming constants	18
2.2 Declaring functions	19
2.3 Identifiers in Standard ML	21
Numbers, character strings and truth values	22
2.4 Arithmetic	22
2.5 Strings and characters	24
2.6 Truth values and conditional expressions	26
Pairs, tuples and records	27
2.7 Vectors: an example of pairing	28
2.8 Functions with multiple arguments and results	29
2.9 Records	32

vi *Contents*

2.10	Infix operators	36
	The evaluation of expressions	38
2.11	Evaluation in ML: call-by-value	39
2.12	Recursive functions under call-by-value	40
2.13	Call-by-need, or lazy evaluation	44
	Writing recursive functions	48
2.14	Raising to an integer power	48
2.15	Fibonacci numbers	49
2.16	Integer square roots	52
	Local declarations	53
2.17	Example: real square roots	54
2.18	Hiding declarations using <code>local</code>	55
2.19	Simultaneous declarations	56
	Introduction to modules	59
2.20	The complex numbers	59
2.21	Structures	60
2.22	Signatures	62
	Polymorphic type checking	63
2.23	Type inference	64
2.24	Polymorphic function declarations	65
	Summary of main points	67
<b>3</b>	<b>Lists</b>	<b>69</b>
	Chapter outline	69
	Introduction to lists	70
3.1	Building a list	70
3.2	Operating on a list	72
	Some fundamental list functions	74
3.3	Testing lists and taking them apart	74
3.4	List processing by numbers	76
3.5	Append and reverse	78
3.6	Lists of lists, lists of pairs	81
	Applications of lists	82
3.7	Making change	83
3.8	Binary arithmetic	85
3.9	Matrix transpose	87
3.10	Matrix multiplication	89
3.11	Gaussian elimination	90
3.12	Writing a number as the sum of two squares	93

<i>Contents</i>	vii
3.13 The problem of the next permutation	95
The equality test in polymorphic functions	96
3.14 Equality types	97
3.15 Polymorphic set operations	98
3.16 Association lists	101
3.17 Graph algorithms	102
Sorting: A case study	108
3.18 Random numbers	108
3.19 Insertion sort	109
3.20 Quick sort	110
3.21 Merge sort	111
Polynomial arithmetic	114
3.22 Representing abstract data	115
3.23 Representing polynomials	116
3.24 Polynomial addition and multiplication	117
3.25 The greatest common divisor	119
Summary of main points	121
<b>4 Trees and Concrete Data</b>	<b>123</b>
Chapter outline	123
The datatype declaration	124
4.1 The King and his subjects	124
4.2 Enumeration types	127
4.3 Polymorphic datatypes	128
4.4 Pattern-matching with <code>val</code> , <code>as</code> , <code>case</code>	130
Exceptions	134
4.5 Introduction to exceptions	134
4.6 Declaring exceptions	135
4.7 Raising exceptions	136
4.8 Handling exceptions	138
4.9 Objections to exceptions	140
Trees	141
4.10 A type for binary trees	142
4.11 Enumerating the contents of a tree	145
4.12 Building a tree from a list	146
4.13 A structure for binary trees	148
Tree-based data structures	148
4.14 Dictionaries	149
4.15 Functional and flexible arrays	154

viii *Contents*

4.16	Priority queues	159
	A tautology checker	164
4.17	Propositional Logic	164
4.18	Negation normal form	166
4.19	Conjunctive normal form	167
	Summary of main points	170
<b>5</b>	<b>Functions and Infinite Data</b>	<b>171</b>
	Chapter outline	171
	Functions as values	172
5.1	Anonymous functions with <i>fn</i> notation	172
5.2	Curried functions	173
5.3	Functions in data structures	176
5.4	Functions as arguments and results	177
	General-purpose functionals	179
5.5	Sections	179
5.6	Combinators	180
5.7	The list functionals <i>map</i> and <i>filter</i>	182
5.8	The list functionals <i>takewhile</i> and <i>dropwhile</i>	184
5.9	The list functionals <i>exists</i> and <i>all</i>	184
5.10	The list functionals <i>foldl</i> and <i>foldr</i>	185
5.11	More examples of recursive functionals	188
	Sequences, or infinite lists	191
5.12	A type of sequences	192
5.13	Elementary sequence processing	194
5.14	Elementary applications of sequences	197
5.15	Numerical computing	199
5.16	Interleaving and sequences of sequences	201
	Search strategies and infinite lists	204
5.17	Search strategies in ML	204
5.18	Generating palindromes	207
5.19	The Eight Queens problem	208
5.20	Iterative deepening	210
	Summary of main points	211
<b>6</b>	<b>Reasoning About Functional Programs</b>	<b>213</b>
	Chapter outline	213
	Some principles of mathematical proof	214
6.1	ML programs and mathematics	214

<i>Contents</i>	ix
6.2 Mathematical induction and complete induction	216
6.3 Simple examples of program verification	220
Structural induction	224
6.4 Structural induction on lists	225
6.5 Structural induction on trees	229
6.6 Function values and functionals	233
A general induction principle	237
6.7 Computing normal forms	238
6.8 Well-founded induction and recursion	242
6.9 Recursive program schemes	246
Specification and verification	248
6.10 An ordering predicate	249
6.11 Expressing rearrangement through multisets	251
6.12 The significance of verification	254
Summary of main points	256
<b>7 Abstract Types and Functors</b>	<b>257</b>
Chapter outline	258
Three representations of queues	258
7.1 Representing queues as lists	259
7.2 Representing queues as a new datatype	260
7.3 Representing queues as pairs of lists	261
Signatures and abstraction	263
7.4 The intended signature for queues	263
7.5 Signature constraints	264
7.6 The <code>abstype</code> declaration	266
7.7 Inferred signatures for structures	269
Functors	271
7.8 Testing the queue structures	272
7.9 Generic matrix arithmetic	275
7.10 Generic dictionaries and priority queues	280
Building large systems using modules	285
7.11 Functors with multiple arguments	285
7.12 Sharing constraints	290
7.13 Fully-functorial programming	294
7.14 The <code>open</code> declaration	299
7.15 Signatures and substructures	305
Reference guide to modules	308
7.16 The syntax of signatures and structures	309



x *Contents*

7.17	The syntax of module declarations	311
	Summary of main points	312
<b>8</b>	<b>Imperative Programming in ML</b>	<b>313</b>
	Chapter outline	313
	Reference types	314
8.1	References and their operations	314
8.2	Control structures	317
8.3	Polymorphic references	321
	References in data structures	326
8.4	Sequences, or lazy lists	327
8.5	Ring buffers	331
8.6	Mutable and functional arrays	335
	Input and output	340
8.7	String processing	340
8.8	Text input/output	344
8.9	Text processing examples	346
8.10	A pretty printer	351
	Summary of main points	356
<b>9</b>	<b>Writing Interpreters for the <math>\lambda</math>-Calculus</b>	<b>357</b>
	Chapter outline	357
	A functional parser	357
9.1	Scanning, or lexical analysis	358
9.2	A toolkit for top-down parsing	360
9.3	The ML code of the parser	363
9.4	Example: parsing and displaying types	367
	Introducing the $\lambda$ -calculus	372
9.5	$\lambda$ -terms and $\lambda$ -reductions	372
9.6	Preventing variable capture in substitution	375
	Representing $\lambda$ -terms in ML	378
9.7	The fundamental operations	378
9.8	Parsing $\lambda$ -terms	381
9.9	Displaying $\lambda$ -terms	382
	The $\lambda$ -calculus as a programming language	384
9.10	Data structures in the $\lambda$ -calculus	385
9.11	Recursive definitions in the $\lambda$ -calculus	388
9.12	The evaluation of $\lambda$ -terms	389
9.13	Demonstrating the evaluators	393

<i>Contents</i>	xi
Summary of main points	396
<b>10 A Tactical Theorem Prover</b>	<b>397</b>
Chapter outline	397
A sequent calculus for first-order logic	398
10.1 The sequent calculus for propositional logic	399
10.2 Proving theorems in the sequent calculus	400
10.3 Sequent rules for the quantifiers	403
10.4 Theorem proving with quantifiers	404
Processing terms and formulæ in ML	407
10.5 Representing terms and formulæ	407
10.6 Parsing and displaying formulæ	411
10.7 Unification	416
Tactics and the proof state	420
10.8 The proof state	420
10.9 The ML signature	421
10.10 Tactics for basic sequents	424
10.11 The propositional tactics	426
10.12 The quantifier tactics	428
Searching for proofs	430
10.13 Commands for transforming proof states	430
10.14 Two sample proofs using tactics	433
10.15 Tacticals	436
10.16 Automatic tactics for first-order logic	440
Summary of main points	444
<i>Project Suggestions</i>	445
<i>Bibliography</i>	449
<i>Syntax Charts</i>	457
<i>Index</i>	469

## 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

xiv *Preface to the Second Edition*

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  $\lambda$ -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 . . . <sup>1</sup>

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 ranke 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

<sup>1</sup> 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](mailto: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](mailto: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](mailto: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](mailto: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. Pieter 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

xviii *Preface*

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.