

Cambridge University Press  
978-0-521-03311-4 - Compiling with Continuations  
Andrew W. Appel  
Frontmatter  
[More information](#)

---

COMPILING WITH CONTINUATIONS

Cambridge University Press  
978-0-521-03311-4 - Compiling with Continuations  
Andrew W. Appel  
Frontmatter  
[More information](#)

---

---

# COMPILING WITH CONTINUATIONS

---

ANDREW W. APPEL  
*Department of Computer Science, Princeton University*

CAMBRIDGE UNIVERSITY PRESS  
*Cambridge*  
*New York Port Chester Melbourne Sydney*

Cambridge University Press  
 978-0-521-03311-4 - Compiling with Continuations  
 Andrew W. Appel  
 Frontmatter  
[More information](#)

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

Cambridge University Press  
 The Edinburgh Building, Cambridge CB2 2RU, 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/9780521416955](http://www.cambridge.org/9780521416955)

© Cambridge University Press 1992

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 1992  
 This digitally printed first paperback version (with corrections) 2006

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

*Library of Congress Cataloguing in Publication data*

Appel, Andrew W., 1960-  
 Compiling with continuations / Andrew W. Appel.  
 p. cm.  
 Includes bibliographical references.  
 ISBN 0-521-41695-7  
 1. Compilers (Computer programs) 2. Standard ML of New Jersey.  
 I. Title.  
 QA76.76.C65A67 1992  
 005.4'53-dc20 91-26651  
 CIP

ISBN-13 978-0-521-41695-5 hardback  
 ISBN-10 0-521-41695-7 hardback

ISBN-13 978-0-521-03311-4 paperback  
 ISBN-10 0-521-03311-X paperback

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

---

<b>Acknowledgments</b>	<b>ix</b>
<b>1 Overview</b>	<b>1</b>
1.1 Continuation-passing style . . . . .	1
1.2 Advantages of CPS . . . . .	4
1.3 What is ML? . . . . .	6
1.4 Compiler organization . . . . .	9
<b>2 Continuation-passing style</b>	<b>11</b>
2.1 The CPS datatype . . . . .	11
2.2 Functions that escape . . . . .	16
2.3 Scope rules . . . . .	17
2.4 Closure conversion . . . . .	18
2.5 Spilling . . . . .	21
<b>3 Semantics of the CPS</b>	<b>23</b>
<b>4 ML-specific optimizations</b>	<b>37</b>
4.1 Data representation . . . . .	37
4.2 Pattern matching . . . . .	43
4.3 Equality . . . . .	45
4.4 Unboxed updates . . . . .	46
4.5 The mini-ML sublanguage . . . . .	46
4.6 Exception declarations . . . . .	49
4.7 The lambda language . . . . .	50
4.8 The module system . . . . .	52
<b>5 Conversion into CPS</b>	<b>55</b>
5.1 Variables and constants . . . . .	55
5.2 Records and selection . . . . .	56
5.3 Primitive arithmetic operators . . . . .	57
5.4 Function calls . . . . .	59
5.5 Mutually recursive functions . . . . .	60
5.6 Data constructors . . . . .	60
5.7 Case statements . . . . .	61

5.8	Exception handling . . . . .	63
5.9	Call with current continuation . . . . .	64
<b>6</b>	<b>Optimization of the CPS</b>	<b>67</b>
6.1	Constant folding and $\beta$ -contraction . . . . .	68
6.2	Eta reduction and uncurrying . . . . .	76
6.3	Cascading optimizations . . . . .	78
6.4	Implementation . . . . .	80
<b>7</b>	<b>Beta expansion</b>	<b>83</b>
7.1	When to do in-line expansion . . . . .	87
7.2	Estimating the savings . . . . .	89
7.3	Runaway expansion . . . . .	92
<b>8</b>	<b>Hoisting</b>	<b>93</b>
8.1	Merging FIX definitions . . . . .	93
8.2	Rules for hoisting . . . . .	95
8.3	Hoisting optimizations . . . . .	96
<b>9</b>	<b>Common subexpressions</b>	<b>99</b>
<b>10</b>	<b>Closure conversion</b>	<b>103</b>
10.1	A simple example . . . . .	104
10.2	A bigger example . . . . .	106
10.3	Closure-passing style . . . . .	109
10.4	The closure-conversion algorithm . . . . .	109
10.5	Closure representation . . . . .	112
10.6	Callee-save registers . . . . .	114
10.7	Callee-save continuation closures . . . . .	119
10.8	Stack allocation of closures . . . . .	122
10.9	Lifting function definitions to top level . . . . .	124
<b>11</b>	<b>Register spilling</b>	<b>125</b>
11.1	Rearranging the expression . . . . .	128
11.2	The spilling algorithm . . . . .	128
<b>12</b>	<b>Space complexity</b>	<b>133</b>
12.1	Axioms for analyzing space . . . . .	136
12.2	Preserving space complexity . . . . .	137
12.3	Closure representations . . . . .	142
12.4	When to initiate garbage collection . . . . .	144
<b>13</b>	<b>The abstract machine</b>	<b>147</b>
13.1	Compilation units . . . . .	147
13.2	Interface with the garbage collector . . . . .	148
13.3	Position-independent code . . . . .	150

*CONTENTS*

vii

13.4	Special-purpose registers . . . . .	151
13.5	Pseudo-operations . . . . .	154
13.6	Instructions of the continuation machine . . . . .	155
13.7	Register assignment . . . . .	158
13.8	Branch prediction . . . . .	160
13.9	Generation of abstract-machine instructions . . . . .	161
13.10	Integer arithmetic . . . . .	161
13.11	Unboxed floating-point values . . . . .	162
<b>14</b>	<b>Machine-code generation</b>	<b>165</b>
14.1	Translation for the VAX . . . . .	165
14.1.1	Span-dependent instructions . . . . .	167
14.2	Translation for the MC68020 . . . . .	168
14.3	Translation for the MIPS and SPARC . . . . .	169
14.3.1	PC-relative addressing . . . . .	170
14.3.2	Instruction scheduling . . . . .	170
14.3.3	Anti-aliasing . . . . .	171
14.3.4	Alternating temporaries . . . . .	173
14.4	An example . . . . .	174
<b>15</b>	<b>Performance evaluation</b>	<b>179</b>
15.1	Hardware . . . . .	181
15.2	Measurements of individual optimizations . . . . .	183
15.3	Tuning the parameters . . . . .	187
15.4	More about caches . . . . .	187
15.5	Compile time . . . . .	198
15.6	Comparison with other compilers . . . . .	200
15.7	Conclusions . . . . .	201
<b>16</b>	<b>The runtime system</b>	<b>205</b>
16.1	Efficiency of garbage collection . . . . .	205
16.2	Breadth-first copying . . . . .	206
16.3	Generational garbage collection . . . . .	207
16.4	Runtime data formats . . . . .	210
16.5	Big bags of pages . . . . .	211
16.6	Asynchronous interrupts . . . . .	212
<b>17</b>	<b>Parallel programming</b>	<b>215</b>
17.1	Coroutines and semaphores . . . . .	216
17.2	Better programming models . . . . .	219
17.3	Multiple processors . . . . .	220
17.4	Multiprocessor garbage collection . . . . .	221

<b>18 Future directions</b>	<b>223</b>
18.1 Control dependencies . . . . .	223
18.2 Type information . . . . .	225
18.3 Loop optimizations . . . . .	225
18.4 Garbage collection . . . . .	227
18.5 Static single-assignment form . . . . .	228
18.6 State threading . . . . .	228
<b>A Introduction to ML</b>	<b>229</b>
A.1 Expressions . . . . .	231
A.2 Patterns . . . . .	233
A.3 Declarations . . . . .	235
A.4 Some examples . . . . .	236
<b>B Semantics of the CPS</b>	<b>239</b>
<b>C Obtaining Standard ML of New Jersey</b>	<b>245</b>
<b>D Readings</b>	<b>247</b>
<b>Bibliography</b>	<b>249</b>
<b>Index</b>	<b>256</b>

---

## ACKNOWLEDGMENTS

---

The compiler described in this book—**Standard ML of New Jersey**—is the work of many people. **David B. MacQueen** and I began the work in 1986; we intended to spend about one year making a Standard ML front end as a tool for further research. David did most of the work on the type checker and module system; we worked together on the parser, abstract syntax, static environment mechanism, and semantic analysis. My interest has been in dynamic semantics, intermediate representations, optimization, code generation, and runtime system.

Of course, we've ended up spending more than five years on the implementation, as the scope of project has become much more ambitious: a complete, robust, efficient, portable programming environment for Standard ML. We could not have done it without the help we've received from many talented people. In alphabetical order:

**Bruce F. Duba** helped improve the pattern-match compiler, the CPS constant-folding phase, the in-line expansion phase, the spill phase, and numerous other parts of the compiler. He also helped to design the *call with current continuation* mechanism.

**Adam T. Dingle** implemented the Emacs mode for the debugger.

**Lal George** taught the code generator about floating-point registers and made floating-point performance respectable. He also fixed several difficult bugs introduced by me and others.

**Trevor Jim** helped design the CPS representation, and implemented the match compiler and the closure-conversion phase, the original library of floating-point functions, and the original assembly-language implementation of external primitive functions.

**James S. Mattson** implemented the first version of the lexical-analyzer generator used in constructing the compiler.

**James W. O'Toole** implemented the NS32032 code generator.

**Norman Ramsey** implemented the MIPS code generator.

**John H. Reppy** contributed many improvements and rewrites of the runtime system. He implemented the signal-handling mechanism, improved the *call with current continuation* mechanism, designed the current mechanism for calls to C functions, implemented the sophisticated new garbage collector, and generally made the runtime system more robust. He also implemented the SPARC code generator.

**Nick Rothwell** helped implement the separate compilation mechanism.



**Zhong Shao** implemented common-subexpression elimination, as well as the callee-save convention that uses multiple-register continuations for faster procedure calls.

**David R. Tarditi** improved the lexical-analyzer generator and implemented the parser generator used to build parts of the front end; he helped in implementing the type-reconstruction algorithm used by the debugger; and he implemented the ML-to-C translator with **Anurag Acharya** and **Peter Lee**.

**Mads Tofte** helped implement the separate compilation mechanism.

**Andrew P. Tolmach** implemented the SML/NJ debugger. He also rewrote static environments (symbol tables) in a more functional style.

**Peter Weinberger** implemented the first version of the copying garbage collector.

Finally, I would like to thank those who gave me helpful comments on early drafts of this book: Ron Cytron, Mary Fernandez, Lal George, Peter Lee, Greg Morrisett, Zhong Shao, David Tarditi, and Andrew Tolmach.