Compiling Esterel

Compiling Esterel

Dumitru Potop-Butucaru

Stephen A. Edwards

Gérard Berry

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN 978-0-387-70626-9 (HB)
ISBN 978-0-387-70628-3 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

All Rights Reserved
© 2007 Springer
No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

To Maria, Eleonora, and Victoria

Preface

This book is about compiling the Esterel language. One of us (Berry) created Esterel in the early 1980s for programming embedded reactive systems. Over time, we and others have developed both academic and commercial compilers for it that generate software and hardware. The newest version of Esterel, called Esterel V7, is being standardized by the IEEE as of 2006.

This book had its genesis in 2002 as Potop-Butucaru's thesis. Written under Berry and Robert de Simone, it made two main contributions: the first semantics for Esterel that included data manipulation (long a part of the language, this was its first formalization), and a very sophisticated code generation technique that remains one of the best developed so far.

At Potop-Butucaru's thesis defense, held during the SYNCHRON workshop in La Londes les Maures, France in November 2002*, Edwards, a jury member, remarked that the thesis would make a good book. Potop-Butucaru interpreted this as meaning only that the thesis would make a good starting point for the book. Thus, one of us (Edwards) expected to have a book within a few months, but instead it took years.

Little more than the general outline of the original thesis remains at this point; much has been added and clarified. We added chapters on the behavioral semantics of Esterel (Chapter 4), recently-developed compilation techniques used in the Columbia Esterel compiler (Chapter 10), and appendices on the extra constructs in the language that are not covered formally but must be handled by all compilers (Appendix A), the first formal language reference manual for the dialect of Esterel described in this book (Appendix B), the C language interface (Appendix C), and a description of the new Esterel V7 dialect of Esterel (Appendix D).

How to read this book

We divided this book into three parts. In the first part (Chapters 1–2), we provide an intuitive description of the Esterel language—enough to familiarize the reader with Esterel's synchronous model of time and the implications it

*A major side benefit of working on Esterel is that most of the meetings take place along the Côte d'Azur, which, not coincidentally, includes the Esterel region after which the language was named.

has for compilation. This should also be enough to get one started writing Esterel programs, but it does not presume to be a comprehensive tutorial on coding in Esterel.

The second part (Chapters 3–6) describes the formal semantics of Esterel, extending the work of Berry [7]. Why do we need three very different presentations of Esterel semantics? Each takes a different approach to describing how an Esterel program is to be executed and is appropriate for addressing different issues in code generation. The behavioral semantics (Chapter 4) is the simplest we present and most clearly addresses some of the thornier aspects of Esterel's rich control constructs and intra-cycle data dependencies. Unfortunately, it is a poor starting point for implementing the language as it represents control by rewriting the program. This makes for a convenient formalism but would make for an extremely inefficient implementation.

The practical shortcomings of the behavioral semantics lead us to the operational semantics (Chapter 5), which uses a more complicated formalism that represents program state by decorating the program text with all manner of diacritical marks. It is a much more complicated formalism as a result, but much closer to a practical implementation that represents program state as some sort of persistent marking of a program.

Unlike the behavioral and operational semantics, which use Plotkin's structural operational style, the third style presented in the second part—the circuit translation—shows how to translate an Esterel program into a circuit netlist. While this may appear surprising for a compiler that produces software, it turns out to be quite effective as the semantics of the circuit model very closely match those of Esterel and are much simpler. In fact, all the efficient compilation techniques that we present in the third part of the book start from GRC—a circuit-netlist-like intermediate representation.

Finally, the third part of the book (Chapters 7–10) gets at our main point: how to translate Esterel programs into efficient software implementations. After an overview, we present the GRC intermediate format (Chapter 8)—a representation developed by Potop-Butucaru as part of his thesis work that has become the foundation for a number of compilers. GRC is a hybrid representation that resembles both a traditional control-flow graph and a circuit netlist; it embodies all that we have learned about the structure of Esterel semantics in the past twenty years. Generating efficient code from it is natural; we describe the basic translation in Chapter 9. Finally, in Chapter 10, we describe how a slight variant of the representation has been used as the basis of two back ends in the open-source Columbia Esterel compiler.

The four appendices contain information that had previously been scattered in various technical reports and tutorials. Appendix A discusses parts of the language that are not usually included in the formal presentation of the language semantics, including valued signals, the `pre` operator, concurrent `trap-exit` abortion handling, and task control. Most amount to syntactic sugar, but deserve the more formal treatment we give them here.

Appendix B is the first semi-formal language reference manual for the V5 dialect of Esterel we use in this book. Previously, this information has been presented in a less formal tutorial style that was not as suitable for writing a compiler. See the Esterel V5 primer [8].

Appendix C presents the C language interface used by all the compilers described in this book. This interface was first used in the INRIA compiler and has since become the de facto standard for code generated from Esterel. The appendix explains, among other things, how to interface with the C code generated by any Esterel compiler and actually make use of it, which in some sense is the whole point of this book.

Finally, Appendix D presents the new Esterel V7 dialect, which has been under development at Esterel Technologies since 2001 as a broad extension of Esterel V5. The compilation techniques presented in this book are being applied to this dialect, as its core semantics remain the same. The Esterel V7 language is open (not proprietary) and its language reference manual [29] has been submitted for IEEE standardization.

Acknowledgements

Potop-Butucaru would like to acknowledge the supervision and help of Robert de Simone in writing of the original Ph.D. thesis.

Contents

Preface **vii**

I The Esterel Language **1**

1 Introduction to Esterel **3**
1.1 Reactive Systems 3
1.2 The Synchronous Hypothesis 4
1.3 Implementation Issues 5
1.4 Causality 6
1.5 Related work 6
1.6 A First Esterel Example 7
1.7 Causality Cycles 8
1.8 Code Generation 9
1.8.1 Translation to Explicit State Machines 9
1.8.2 Translation to Circuits 10
1.8.3 Direct Compilation to C Code 12
1.9 Executing the Generated Code 12
1.9.1 Existing Solutions 13

2 The Esterel Language **15**
2.1 Syntax and Naïve Semantic Principles 15
2.2 The Kernel Esterel Language 20
2.3 Esterel Through Examples 24
2.4 Host Language 31
2.5 Program Structure and Interface 31
2.5.1 Data Handling 32
2.5.2 Signal and Signal Relation Declarations 34
2.5.3 The `run` Pseudo-Statement 34

II Formal Semantics 39

3 Introduction to Esterel Semantics 41
3.1 Intuition and Mathematical Foundations 41
3.1.1 The Constructive Approach 43
3.2 Flavors of Constructive Semantics 47
3.3 Conventions and Preliminary Definitions 50
3.3.1 Global Correctness of an Esterel Program 50
3.3.2 Restriction to Kernel Esterel 51
3.3.3 Signal Events 51
3.3.4 Trap Handling and Completion Codes 51

4 Constructive Behavioral Semantics 55
4.1 Behavioral Transitions 55
4.1.1 Transition syntax 55
4.1.2 States as Decorated Terms 56
4.1.3 State Syntax 57
4.2 Analysis of Potentials 58
4.2.1 The Definition of *Must*, *Can*, and Can^+ 60
4.2.2 Elementary Properties 68
4.3 Semantic Rules 68
4.4 Proof 72
4.5 Determinism 72
4.6 Loop-Safe Programs. Completion Code Potentials 73
4.7 Program Behavior 75

5 Constructive Operational Semantics 79
5.1 Microsteps 80
5.2 COS Terms 80
5.2.1 Control Flow Propagation 81
5.2.2 State-Dependent Behavior 81
5.2.3 Syntax of Semantic Terms 82
5.3 Data Representation 84
5.4 Semantic Rules 85
5.4.1 Rules for Pure Esterel Primitives 85
5.4.2 Rules for Data-Handling Primitives 92
5.5 Analysis of Potentials 95
5.5.1 Reduction to Non-Dotted Terms 96
5.5.2 Non-Dotted Terms over Dataless Primitives 97
5.5.3 Non-Dotted Terms over Data-Handling Primitives . . 99
5.6 Behaviors as Sequences of Microsteps 99
5.7 COS versus CBS 101

6 Constructive Circuit Translation 103
6.1 Digital Circuits with Data 104
6.1.1 Circuit Semantics. Constructive Causality 104
6.1.2 Extension to Circuits with Data 107
6.1.3 Formal Definitions 110
6.2 Translation Principles 112
6.2.1 The Selection Circuit 114
6.2.2 The Surface and Depth Circuits 115
6.2.3 The Global Context 116
6.3 Translation Rules 117
6.3.1 Dataless Primitives 117
6.3.2 Data-Handling Primitives 126
6.4 Circuit Translation versus COS 128

III Compiling Esterel 133

7 Overview 135
7.1 Compiler Classes 135
7.2 A Brief History 136
7.3 The INRIA Compiler 137
7.4 The Synopsys Compiler 139
7.5 The Saxo-RT Compiler 140
7.6 The Columbia Esterel Compiler 144

8 The GRC Intermediate Format 145
8.1 Definition and Intuitive Semantics 146
8.1.1 The Hierarchical State Representation 146
8.1.2 The Control/Data Flowgraph 149
8.1.3 Implementation Issues 154
8.2 Esterel to GRC Translation 158
8.2.1 Translation Principles 158
8.2.2 Translation Rules 159
8.2.3 The Global Context 167
8.3 Formal Simulation Semantics and Translation Correctness . 167
8.4 Format Optimizations 168
8.4.1 State Representation Analysis 170
8.4.2 Flowgraph Optimizations 173

9 Code Generation from GRC 179
9.1 Defining “Acyclic” 180
9.2 Code Generation for Acyclic Specifications 185
9.2.1 State Encoding 185
9.2.2 Flowgraph Transformations 190
9.2.3 Scheduling 192

9.3 Code Generation for Cyclic Specifications 193
9.4 Benchmarks 200

10 The Columbia Compiler 203
10.1 The Dynamic Technique 203
10.1.1 An Example 204
10.1.2 Sequential Code Generation 211
10.1.3 The Clustering Algorithm 214
10.2 The Program Dependence Graph Approach 215
10.2.1 Program Dependence Graphs 216
10.2.2 Scheduling 218
10.2.3 Restructuring the PDG 221
10.2.4 Generating Sequential Code 228
10.3 Benchmarks 229

A Language Extensions 235
A.1 Signal Expressions 235
A.1.1 Syntactic Aspects and Limitations 236
A.1.2 Combinational Expressions 236
A.1.3 The `pre` Operator 238
A.1.4 Delay Expressions. Preemption Triggers 241
A.2 Traps and Trap Expressions 242
A.2.1 Concurrent Traps and Trap Expressions 243
A.2.2 Valued Traps 244
A.3 The `finalize` Statement 245
A.4 Tasks 247
A.4.1 Task Synchronization Semantics 248
A.4.2 Multiple `exec` 251

B An Esterel Reference Manual 253
B.1 Lexical Conventions 253
B.1.1 Tokens 253
B.1.2 Comments 253
B.1.3 Identifiers 253
B.1.4 Reserved Words 254
B.1.5 Literals 254
B.2 Namespaces and Predefined Objects 255
B.2.1 Signals and Sensors 256
B.2.2 Variables and Constants 256
B.2.3 Traps 256
B.2.4 Types 257
B.2.5 Functions and Procedures 257
B.2.6 Tasks 257
B.3 Expressions 258
B.3.1 Data Expressions 258

B.3.2 Constant Atoms 258
B.3.3 Signal Expressions 259
B.3.4 Delay Expressions 259
B.3.5 Trap Expressions 259
B.4 Statements 260
B.4.1 Control Flow Operators 260
B.4.2 `abort`: Strong Preemption 261
B.4.3 `await`: Strong Preemption 262
B.4.4 `call`: Procedure Call 264
B.4.5 `do-upto`: Conditional Iteration (deprecated) 264
B.4.6 `do-watching`: Strong Preemption (deprecated) 264
B.4.7 `emit`: Signal Emission 264
B.4.8 `every-do`: Conditional Iteration 265
B.4.9 `exec`: Task Execution 265
B.4.10 `exit`: Trap Exit 266
B.4.11 `halt`: Wait Forever 266
B.4.12 `if`: Conditional for Data 266
B.4.13 `loop`: Infinite Loop 267
B.4.14 `loop-each`: Conditional Iteration 267
B.4.15 `nothing`: No Operation 268
B.4.16 `pause`: Unit Delay 268
B.4.17 `present`: Conditional for Signals 268
B.4.18 `repeat`: Iterate a Fixed Number of Times 269
B.4.19 `run`: Module Instantiation 270
B.4.20 `signal`: Local Signal Declaration 271
B.4.21 `suspend`: Preemption with State Freeze 272
B.4.22 `sustain`: Emit a Signal Indefinitely 273
B.4.23 `trap`: Trap Declaration and Handling 273
B.4.24 `var`: Local Variable Declaration 273
B.4.25 `weak abort`: Weak Preemption 274
B.5 Modules 275
B.5.1 Interface Declarations 276

C The C Language Interface 281
C.1 Overview 281
C.2 C Code for Data Handling 283
C.2.1 Defining Data-handling Objects 283
C.2.2 Predefined Types 283
C.2.3 User-defined Types 283
C.2.4 Constants 286
C.2.5 Functions 286
C.2.6 Procedures 287
C.3 The Reaction Interface 288
C.3.1 Input Signals 288
C.3.2 Return Signals 289

C.3.3 Output Signals 289
C.3.4 Inputoutput Signals 289
C.3.5 Sensors 290
C.3.6 Reaction and Reset 290
C.3.7 Notes 291
C.4 Task Handling 292
C.4.1 The Low-level Layer: ExecStatus 292
C.4.2 The Functional Interface to Tasks 295

D Esterel V7 297
D.1 Data Support 298
D.1.1 Basic Data Types 298
D.1.2 Arrays 298
D.1.3 Generic Types 299
D.1.4 Bitvectors 299
D.1.5 From Numbers to Bitvectors and Back 300
D.1.6 Data Units 300
D.2 Signals 301
D.2.1 Value-only Signals 301
D.2.2 Temporary Signals 301
D.2.3 Registered Signals 301
D.2.4 Signal Initialization 302
D.2.5 Oracles 302
D.3 Interfaces 303
D.3.1 Interface Declaration 303
D.3.2 Interfaces and Modules 303
D.3.3 Mirroring an Interface 304
D.3.4 Interface Refinement in Modules 304
D.4 Statements 304
D.4.1 Expressions and Tests 304
D.4.2 Static Replication 304
D.4.3 Enhanced Emit and Sustain Statements 305
D.4.4 Explicit and Implicit Assertions 306
D.4.5 Weak Suspension 307
D.4.6 Signal Connection by Module Instantiation 309
D.5 Multiclock Design 310
D.5.1 Clocks and Multiple Units 311
D.5.2 Simulation of Multiclock Designs by Single-clocked Designs 312

Bibliography 315

Index 323

Figures

1.1 Execution cycle 5
1.2 A possible execution trace for ABRO 8
1.3 The automaton and explicit FSM code for the ABRO example 10
1.4 Synchronous circuit model 11
1.5 Circuit code for ABRO (the reaction function) 11
1.6 Control-flow (pseudo-)code for ABRO 12
1.7 A possible asynchronous run of ABRO 13

2.1 An execution trace of `MainExample` 25
2.2 Interconnections between `Cell` modules in `Arbiter2` 28
2.3 A 3-cell FIFO in Esterel. 35

3.1 A correct, but complex Esterel example 43
3.2 Boolean circuits for two intuitionistic formulas 46
3.3 Truth table for ternary logic ($\mathcal{B}_\perp$) operators 46
3.4 Completion codes in the start instant of a simple statement 53

6.1 A possible circuit translation for ABRO 105
6.2 The two possible evaluation sequences for a small circuit . . 106
6.3 Causality in circuit evaluation 106
6.4 A simple sequential circuit 107
6.5 Circuit with data, first example 108
6.6 Circuit with data, second example 109
6.7 Circuit with data, third example 109
6.8 Incorrect circuits 112
6.9 The selection circuit of a simple example 115
6.10 The interface of the generated circuits 115
6.11 The global translation context 116
6.12 Surface and depth circuits for `pause` 117
6.13 Surface circuit for `nothing` 117
6.14 Surface circuit for "`loop` p `end`" 118
6.15 Depth circuit for "`loop` p `end`" 118

6.16 Surface circuit for the two-way sequence "p ; q" 119
6.17 Depth circuit for the binary sequence "p ; q" 119
6.18 Surface circuit for "p_1 || ... || p_n" 120
6.19 Depth circuit for "p_1 || ... || p_n" 120
6.20 The circuit-level parallel synchronizer 121
6.21 Surface circuit for "`emit S`" 121
6.22 Surface and depth circuits for "`signal` S `in` p `end`" 122
6.23 Surface circuit for the signal and data tests 122
6.24 Depth circuit for the test statements 123
6.25 Signal test circuit (`present` statement) 123
6.26 Surface circuit for "`suspend` p `when` *expr*" 124
6.27 Depth circuit for "`suspend` p `when` *expr*" 124
6.28 Surface circuit for "`exit T`(i)" 124
6.29 Surface circuit for "`trap T in` p `end`" 125
6.30 Depth circuit for "`trap T in` p `end`" 125
6.31 Surface circuit for the variable assignments 126
6.32 Surface circuit for variable declarations 127
6.33 Depth circuit for variable declarations 128
6.34 Variable test circuit (`if` statement) 128

7.1 Esterel implementation flavors 136
7.2 The flow of the INRIA compiler 138
7.3 Esterel fragment and its Synopsys translation into C 140
7.4 Concurrent control-flow graph, Synopsys style 141
7.5 A simple Esterel program modeling a shared resource 142
7.6 Event graph and C code generated by the Saxo-RT compiler 143

8.1 The `MainExample` Esterel program 147
8.2 The selection tree of `MainExample` 148
8.3 Selection tree and selection flags for a small example 148
8.4 Simple GRC flowgraph 149
8.5 Signal dependency representations 150
8.6 The flowgraph of `MainExample` 153
8.7 The Esterel to GRC translation interface 158
8.8 The translation of `pause` 159
8.9 The translation of `nothing` and `emit` S 160
8.10 The translation of `loop` q `end` 160
8.11 The translation of q ; r 161
8.12 The translation of q || r 162
8.13 The translation of `signal` S `in` q `end` 162
8.14 The translation of the test primitives 163
8.15 The translation of `suspend` q `when` *expr* 164
8.16 The translation of `exit` $T(i)$ 164
8.17 The translation of `trap` T `in` q `end` 165

8.18 The translation of variable assignment 165
8.19 The translation of variable declarations 166
8.20 The global translation context 166
8.21 Circuit counterpart for the **Test** node 167
8.22 Circuit counterpart for the **Switch** node 168
8.23 Tagged tree for `MainExample` 173
8.24 Simplified graph for `MainExample` 177

9.1 A GRC-level cycle resolved at the circuit level 181
9.2 Simplified translation patterns for `present` 182
9.3 Example of synchronizer refinement 184
9.4 Bit allocation for our first small example 186
9.5 State encoding examples for our first small example 187
9.6 Bit allocation for `MainExample` 188
9.7 State encoding examples for `MainExample` 188
9.8 Bit allocation for the third small example 189
9.9 State encoding examples for the third small example 189
9.10 Bit allocation for `pause ; pause` 191
9.11 Possible states for `pause ; pause` and their encoding . . . 191
9.12 Flowgraph transformation due to state encoding 192
9.13 A partition of the selection tree of `MainExample` 193
9.14 Code generated from the flowgraph of Figure 8.24 194
9.15 The GRC of the cyclic example 196
9.16 The SCC at GRC and circuit level 196
9.17 Globally acyclic graph for our cyclic example 197

10.1 An Esterel model of a shared resource 205
10.2 The $\mathrm{GRC}_{\mathrm{CEC}}$ for the program in Figure 10.1 206
10.3 The control-flow graph from Figure 10.2 divided into blocks 209
10.4 The code CEC generates for part of the graph in Figure 10.3 210
10.5 Cluster code and the linked list pointers 213
10.6 The clustering algorithm 214
10.7 A program dependence graph requiring interleaving 217
10.8 The Main procedure 219
10.9 Successor Priority Assignment 220
10.10 The Scheduling Procedure 221
10.11 The Restructure procedure 221
10.12 The DuplicationSet function. 222
10.13 The DuplicateNode procedure 223
10.14 The ConnectPredecessors procedure 223
10.15 The restructured PDG from Figure 10.7 226
10.16 A complex example 227
10.17 The reconstructed PDG from Figure 10.7 229

10.18 The PDG of Figure 10.17 after guard variable fusion 231
10.19 The successor ordering procedure 232

A.1 Test sub-circuit for the expression "`C and not (A or B or C)`" 237
A.2 Tagged selection tree for the expansion of `pre` 239
A.3 The life cycle of a task 249

Tables

2.1 The primitives of Pure Esterel 22

8.1 Parallel branch status redundancy statistics 172

9.1 State encoding results on some typical examples 190
9.2 Code speed 202
9.3 Code size 202

10.1 Experimental results for the dynamic approach 230
10.2 Statistics for the examples 230
10.3 Experimental Results for the PDG-based approach 233