

TOWARDS VERIFIED SYSTEMS

EDITED BY
Jonathan Bowen

TOWARDS VERIFIED SYSTEMS

EDITED BY
Jonathan Bowen

safe**mos**

This page deliberately left blank for publisher's use

This page deliberately left blank for publisher's use

This page deliberately left blank for publisher's use

This page deliberately left blank for publisher's use

Contents

Foreword	xvii
Preface	xix
Contact Addresses	xxiii
I Introduction	1
1 Safety-Critical Systems and Formal Methods	3
1.1 A Brief Historical Perspective	3
1.2 Safety-critical Computer Systems	5
1.2.1 Dependable computer systems	6
1.2.2 Formal methods	7
1.2.3 The cost of software safety	9
1.3 Industrial-scale Examples of Use	11
1.3.1 Aviation	12
1.3.2 Railway systems	13
1.3.3 Nuclear power plants	13
1.3.4 Medical systems	14
1.3.5 Ammunition control	16
1.3.6 Embedded microprocessors	17
1.4 Areas of Application of Formal Methods	18
1.4.1 Requirements capture	19
1.4.2 Design	19
1.4.3 Compilation	20
1.4.4 Programmable hardware	21
1.4.5 Documentation	21
1.4.6 Human-computer interface	21
1.4.7 Object-oriented methods	22
1.4.8 Artificial intelligence	22
1.4.9 Static analysis	22
1.5 Safety Standards	22
1.5.1 Formal methods in standards	23
1.5.2 Education, certification and legislation	27
1.6 Discussion	29

1.6.1	Formal methods research	30
1.6.2	Formal methods technology	31
1.6.3	Education and accreditation	31
1.6.4	Standards	32
2	Overview of the Project	35
2.1	The SAFEMOS Project	35
2.1.1	The SAFEMOS tower	37
2.2	System Modelling	38
2.2.1	Timed transition systems	38
2.3	Software Development and Compilation	38
2.3.1	State transition assertions	38
2.3.2	A real-time programming language	39
2.3.3	Program compilation	39
2.4	Hardware Design and Compilation	41
2.4.1	Microprocessor design methods	41
2.4.2	Hardware compilation	41
2.5	Other SAFEMOS Project Work	42
2.5.1	Symbolic execution	42
2.5.2	Assembler verification	44
2.5.3	Machine semantics using Z	45
2.6	Related Work	46
2.7	Conclusion	46
II	Tools and Models	47
3	The HOL Logic and System	49
3.1	Introduction	49
3.2	The HOL Logic	50
3.2.1	Types	50
3.2.2	Terms	51
3.2.3	Standard notions	52
3.2.4	Sequents	53
3.2.5	Semantics	54
3.2.6	Deductive systems	56
3.2.7	Theories	58
3.2.8	Built-in theories and notations	61
3.2.9	Consistency	62
3.2.10	Extensions of theories	62
3.3	The HOL System	67
3.3.1	The history of HOL	68
3.3.2	Overview of the theorem-proving infrastructure	68
3.3.3	Getting and using HOL	70
4	Timed Transition Systems	71

4.1	Introduction to TTSs and HOL	71
4.2	Example: A Traffic Light Controller	73
4.2.1	System description	73
4.2.2	System requirements	75
4.3	A Real-Time Temporal Logic	75
4.3.1	Variables, expressions and equality	77
4.3.2	Boolean operators	77
4.3.3	Next	77
4.3.4	Previous	77
4.3.5	Eventually	77
4.3.6	Always	77
4.3.7	Unless	78
4.3.8	Example	78
4.4	Timed Transition Systems	78
4.4.1	Timed transitions	78
4.4.2	Computations	79
4.4.3	Requirements of computations	80
4.5	Timed Transition Diagrams	80
4.5.1	TTD representation	80
4.5.2	Semantics of TTDs	81
4.5.3	Example	82
4.6	Verification	83
4.6.1	Proof rules	83
4.6.2	Single step rules	84
4.6.3	TTD rules	85
4.6.4	An example proof	87
4.6.5	Other examples	89
4.7	Discussion	90

III Software 91

5	State Transition Assertions: A Case Study 93
5.1	Introduction 93
5.2	An Example: <code>Mult</code> 94
5.2.1	Overview 94
5.2.2	Informal specification of <code>Mult</code> 95
5.2.3	<code>MultProg</code> : an implementation of <code>Mult</code> 96
5.3	A More Detailed Specification of <code>Mult</code> 97
5.4	Determining a Machine from a Program 97
5.5	State Transition Assertions 100
5.5.1	Holding states 101
5.6	Formal Specification of <code>Mult</code> 103
5.7	Correctness of <code>MultProg</code> 104
5.8	Generating Atomic STAs 104

5.9	Laws for Combining STAs	107
5.9.1	The consequence rule	107
5.9.2	The sequencing rule	107
5.9.3	Cases rules	108
5.9.4	The wait loop rule	108
5.9.5	The while rule	110
5.10	Conclusions	112
6	A Real-time Programming Language	115
6.1	The SAFE Programming Language	115
6.1.1	Processes	115
6.1.2	Expressions	116
6.2	Interval Model	117
6.2.1	Types	117
6.2.2	Variables, registers and ports	117
6.2.3	Interval operators	118
6.2.4	Ordering of processes	118
6.3	Interval Semantics	119
6.3.1	Expressions	119
6.3.2	Boolean operators	120
6.3.3	Interval length	120
6.3.4	Stability and assignment	121
6.3.5	Sequence	122
6.3.6	Conditional	122
6.3.7	While loop	122
6.3.8	Local variables	123
6.4	SAFE Semantics	123
6.4.1	Expressions	124
6.4.2	Processes	124
6.5	Laws	125
6.5.1	Ordering	125
6.5.2	Boolean operators	126
6.5.3	Sequence	126
6.5.4	Interval length	127
6.5.5	Assignment and stability	127
6.5.6	Conditional	128
6.5.7	While loops	129
6.5.8	Local variables	130
6.6	Conclusion	130
7	Program Compilation	131
7.1	Machine Language Syntax	131
7.2	Machine Language Semantics	132
7.2.1	Instruction semantics	132
7.2.2	Program semantics	133

7.3	Compiler Specification	133
7.3.1	Symbol table	134
7.3.2	Expressions	134
7.3.3	Processes	136
7.3.4	Compilation in HOL	137
7.4	Correctness of Compilation	139
7.5	Proof of Correctness of Compilation	140
7.5.1	Skip	141
7.5.2	Assignment	142
7.5.3	While Loop	144
7.6	Conclusion	146
IV Hardware		147
8	A Framework for Microprocessor Design	149
8.1	Introduction	149
8.1.1	Hierarchy of computation models	150
8.1.2	Generic arguments	150
8.1.3	Verification template	152
8.1.4	Incremental models	152
8.2	Machine Specification Framework	153
8.2.1	Computational model	153
8.2.2	Definition of relational interpreter	154
8.2.3	Implementation satisfying an abstract machine transition	155
8.2.4	Specification correctness	156
8.2.5	Use of an incremental model	157
8.3	Microcoded Machine Example	157
8.4	Incremental Model of Control Memory	161
8.4.1	Overview of model	161
8.4.2	Behaviour relation in HOL	162
8.4.3	Incremental correctness theorem	162
8.4.4	Microcode ROM	163
8.4.5	Microcoded machine result	163
8.4.6	Independent verification of segments	165
8.5	Summary	165
9	Designing a Processor	167
9.1	Instruction Set and Machine Architecture	167
9.2	Top Level Specification	170
9.2.1	Architectural specification	170
9.2.2	Processor state	173
9.2.3	Instruction specifications	173
9.2.4	Transition system model of machine	175
9.3	Microcoded Implementation	177

9.3.1	Microcode machine	177
9.3.2	Verification of microcoded machine	186
9.4	Low-level Implementation	188
9.4.1	Design style and methods	188
9.4.2	Transformational design	189
9.5	Conclusions	192
10	Hardware Compilation	193
10.1	Introduction	193
10.1.1	Background	194
10.1.2	Previous work and research experience	194
10.1.3	Outline	196
10.2	A Language of Communicating Processes	196
10.2.1	Syntax	196
10.2.2	Algebraic laws	197
10.2.3	Timing delays	198
10.3	Normal Form Implementation	199
10.3.1	Normal form definition	199
10.4	Reduction to Normal Form	201
10.4.1	Assignment	201
10.4.2	Output	201
10.4.3	Input	202
10.4.4	Sequence	202
10.4.5	Conditional	202
10.4.6	Iteration	203
10.5	Example Proof	203
10.6	Rapid Prototype Compiler	205
10.7	Mapping Normal Form into Hardware	205
10.8	Conclusions	206
V	Technology Transfer	209
11	Transfer into Industrial Design	211
11.1	Historical Background	211
11.2	Benefits from Formal Methods	213
11.3	Technology Transfer Problems	214
11.3.1	Modes of use	215
11.3.2	Cost considerations	216
11.3.3	Industrial-scale usage	216
11.4	Requirements for Transfer of Formal Methods	217
11.5	Methods for Transferring Formal Methods	218
11.6	Technology Transfer from the SAFEMOS Project	220
	Appendices: Related Work	223

A	System Verification and the CLI Stack	225
A.1	Introduction	225
A.2	Our Philosophy of Systems Verification	227
A.3	Verifying Systems	228
A.3.1	Defining finite state machines	229
A.3.2	Interpreter equivalence theorems	230
A.3.3	Stacking machines	233
A.4	The CLI Stack and Kit	234
A.4.1	FM9001	235
A.4.2	Piton	236
A.4.3	μ -Gypsy	240
A.4.4	Kit	241
A.4.5	Implementing an applicative language	243
A.4.6	Building verified applications	244
A.4.7	Some statistics	244
A.5	Extending the Stack	245
A.5.1	Revising system components	245
A.5.2	Realizing the stack	246
A.6	Future Verified Systems	246
A.7	Conclusions	247
B	The ProCoS Project: Provably Correct Systems	249
B.1	Introduction	249
B.2	History and Experience	252
B.3	Requirements Engineering and Duration Calculus	253
B.3.1	Model	253
B.3.2	Duration calculus	254
B.3.3	A design	255
B.3.4	Component specifications	257
B.3.5	Further considerations	257
B.4	Program Specification and Development	258
B.4.1	Semantics of communicating systems	258
B.4.2	Specification language	258
B.4.3	Programming language	260
B.4.4	Transformations	260
B.5	Compiler Correctness	261
B.5.1	Compiler development	261
B.6	Base Systems	262
B.6.1	Kernel development	263
B.7	Conclusion	264
	Acknowledgements	267
	Bibliography	269

List of Figures

4.1	Hierarchy of models	72
4.2	Environment	74
4.3	Traffic light controller	74
5.1	An implementation of Mult	96
5.2	Intermediate commands and machine instructions for MultProg	105
7.1	Compilation in HOL	138
7.2	Correctness of compilation	139
8.1	Arithmetic logic unit: (a) 8-bit (b) Generic	151
8.2	Semantics of the POP instruction	153
8.3	Relating implementation to abstract machine transition	155
8.4	Microprocessor implementation overview	159
8.5	Microcode instruction flow	164
9.1	Microcode machine architecture	178
9.2	Microcode machine architecture block decomposition	179
9.3	Instruction fetch unit specification	180
9.4	Instruction select and micro-scheduler units	180
9.5	Extracts from data-path specification	184
9.6	Portion of microcode ROM	184
9.7	Processor core for microcoded machine	185
9.8	Micro-machine specification	186
A.1	Equivalence of machines	231
A.2	Composing equivalence theorems	233
A.3	FM9001 specification levels	237
A.4	Sample Piton state	239
B.1	Timing diagram for <i>Leak</i>	254
B.2	Specification of gas burner control program in SL	259

List of Tables

1.1	Cost of saving a life	10
1.2	Applications of formal methods to safety-critical systems	12
1.3	Cost-effectiveness of approaches compared by Rolls-Royce and Associates .	15
1.4	Comparison of some Hewlett-Packard project metrics	15
1.5	Summary of software-related standards and guidelines	24
9.1	Abstract type manipulation functions	172
B.1	ProCoS tower of work areas	253

To err is human but to really foul things up requires a computer.

Farmers' Almanac for 1978, *Capsules of Wisdom* (1977)

Foreword

In basic science, fundamental discoveries are made by intense concentration on a single issue, and by rigorous control of all extraneous variation. By contrast, in practical engineering new products are designed and new markets opened up by successful integration of the discoveries of many diverse branches of basic science. This requires careful specification of interfaces, which should be tolerant to variation in environmental parameters, and cost-effective for a range of applications.

In a new scientific discipline, or one which has expanded too fast for its own good, it is a slow process to establish a consensus on what is the appropriate subdivision of the subject into its branches, and what are the appropriate methods of research within each branch. Exploration of the structure of the discipline and elucidation of the interfaces between its branches are necessary conditions of progress; and, of course, mathematical concepts, calculations and proofs play the same central role as they have in all well-established scientific disciplines.

In engineering methodology, two directions of interfacing can be distinguished:

1. Horizontal integration between components of a complex product, perhaps implemented in differing materials or technologies;
2. Vertical integration between levels of abstraction in the design process, ranging from requirements through specifications, designs, and ultimate implementation.

The scientific study of both kinds of interface can help not only to clarify the subject matter and structure of a scientific discipline; it can also help the engineer to improve product reliability and reduce time to market by avoiding the most insidious and most expensive kinds of error, those that lurk in the interfaces between components and between phases of the design. The benefits are even greater if the engineering calculations can be carried out or at least checked with the assistance of a computer.

That is the philosophical background to the **safemos** project, whose results are reported in this book, and of several related projects in other leading centres of research. They concentrate on what are recognized as issues central to computing science, including requirements, specifications, designs, programs, compilers, machines architectures, and logic design of hardware. Many of these interfaces are well understood; and here the project has aimed at an increase in rigour of formalization, preparing the ground for reliable mechanical support.

The **safemos** project concentrates on the most urgent problems of ensuring the reliability of designs and programs for embedded systems working in real-time; it is not

aimed at any particular product, but it has clarified the principles of reliable design and implementation.

These principles, we hope, will be just as effective in the timely and reliable implementation of more general systems, where safety is not such a critical issue. But above all, the principles enlarge our basic scientific understanding of computing science, in a way that illuminates the structure of the whole subject and its methods of research.

C.A.R. Hoare

Preface

As the complexity of embedded computer-controlled systems increases, the present industrial practice for their development gives cause for concern, especially for safety-critical applications where human lives are at stake. The use of software in such systems has increased enormously in the last decade. Formal methods, based on firm mathematical foundations, provide one means to help with reducing the risk of introducing errors during specification and development. There is currently much interest in both academic and industrial circles concerning the issues involved, but the techniques still need further investigation and promulgation to make their widespread use a reality.

This book presents some results of research into techniques to aid the formal verification of mixed hardware/software systems. Aspects of system specification and verification from requirements down to the underlying hardware are addressed, with particular regard to real-time issues. The work presented is largely based around the Occam programming language and Transputer microprocessor paradigm. The HOL theorem prover, based on higher order logic, has mainly been used in the application of machine-checked proofs.

The book describes research work undertaken on the collaborative UK DTI/SERC-funded Information Engineering Directorate **safemos** project. The partners were Inmos Ltd, Cambridge SRI, the Oxford University Computing Laboratory and the University of Cambridge Computer Laboratory, who investigated the problems of formally verifying embedded systems. The most important results of the project are presented in the form of a series of interrelated chapters by project members and associated personnel. In addition, overviews of two other ventures with similar objectives are included as appendices.

The material in this book is intended for computing science researchers and advanced industrial practitioners interested in the application of formal methods to real-time safety-critical systems at all levels of abstraction from requirements to hardware. In addition, Chapters 1 and 11 contain material of a more general nature which may be of interest to managers in charge of projects applying formal methods, especially for safety-critical systems, and others who are considering their use.

In Part I of the book, Chapter 1 provides an introduction to the setting to which the rest of the book is intended to contribute, with particular regard to safety-critical systems, where correctness is of paramount importance. Standards are likely to provide a major motivating force for the use of formal methods in the development of such systems, and a selection of these are surveyed. Chapter 2 continues by giving an overview of the work undertaken on the **safemos** project, with a little more detail devoted to areas not covered in subsequent chapters.

Part II provides an introduction to the main theorem proving tool used on the **safemos**

project (HOL) in Chapter 3, together with an example of how it may be used in modelling real-time systems in Chapter 4. Chapter 3 is included to give the reader not acquainted with the HOL mechanical theorem proving system a knowledge of its capabilities that will aid the reading of the rest of the book from Chapter 4 to 9.

Chapter 4 considers the mechanization of *timed transitions systems* (TTS) in HOL to allow modelling and reasoning about real-time systems. A traffic light controller example is used to present the principles involved. A mechanical proof environment could be further developed along the principles presented here to allow the specification and verification of real-time systems at a rather higher level of abstraction than considered in Part III. Embedding of requirements and design specifications, and techniques for demonstrating that a design meets its requirements using TTS proof rules are discussed.

Part III presents the use of HOL for developing and compiling software. Chapter 5 presents a complete self-contained case study of the verification of a small example program. The technique described is intended to be applied when the highest level of integrity is required. The timing aspects are modelled at the level of the machine clock cycle for the compiled object code. This is the only way to ensure completely accurate reasoning about the timing properties of the program. Of course this limits the size of code that can be handled tractably, but it is envisaged that small sections of safety-critical code could be verified in this manner to give the highest degree of confidence. The process is mechanized in HOL to help avoid human error and make it usable for non-trivial examples.

In the past, safety-critical software has often been developed using assembler programs due to the unreliability of high-level languages, and their unpredictable timing properties. On the **safemos** project, a small real-time Occam-like language and its compilation to a Transputer-like instruction set have been developed and mechanized in HOL. The language and its *interval temporal logic* semantics are presented in Chapter 6. Its compilation and the verification of this process are presented in Chapter 7. It is intended that the development of more reliable compilation for real-time programming along these lines will enable higher-level programming techniques to be used for safety-critical systems with more confidence in the future.

Correct software must be run on correct hardware for overall system correctness. Therefore the formal development of both aspects of a software/hardware system is important. Part IV presents aspects of verifying hardware designs. Chapter 8 discusses techniques to design microprocessors in a generic manner. Chapter 9 presents the development of a simple (but realistic) Transputer-like processor. The verified hardware described could be used to run programs compiled by the technique previously presented in Chapter 7.

An interesting recent development is the possibility of compiling hardware in a similar manner to that which software is routinely compiled today. Chapter 10 gives a more speculative presentation of how hardware for safety-critical systems could be developed in the future. These techniques are still an active area of research at an early stage of development and there is potential for considerable progress. For example, there is growing interest in the area of hardware/software co-design, which typically involves the intervention of a design engineer to determine suitable tradeoffs between the use of software and hardware.

Finally in Part V, aspects of technology transfer from formal methods academic research to industrial application are addressed. For formal methods to be accepted, their use

must be integrated into current best industrial practice. It is too risky and expensive to completely replace existing methods. Chapter 11 discusses some of the issues involved and considers the future prospects for methods such as those investigated by the **safemos** project.

Two appendices present related work with similar aims to **safemos**, although using different techniques. At Computational Logic, Inc. (CLI) in the US, the verification of a number of related software and hardware levels has been undertaken using the Boyer-Moore theorem prover. Appendix A presents this inspiring example, and also some of their more recent work. In Europe, the collaborative ESPRIT **ProCoS** project has investigated formal techniques from requirements down to machine code and how these relate to each other. Appendix B gives an overview of the achievements of the first phase of this research project. These efforts are still ongoing and further progress and results are expected.

A large bibliography is included at the end of the book for those interested in particular areas of the **safemos** project, and related work by other researchers in the field of software/hardware system verification. A number of relevant standards and other publicly available documents are also included.

J.P. Bowen

Contact Addresses

Editor

Jonathan Bowen

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building
Parks Road
OXFORD OX1 3QD
England

Email: Jonathan.Bowen@comlab.ox.ac.uk

URL: <http://www.comlab.ox.ac.uk/oucl/people/jonathan.bowen.html>

Contributors

Juanito Camilleri

Department of Computer Studies
University of Malta
University Heights
Msida
Malta G.C.

Email: juan%panther@carla.dist.unige.it

Rachel Cardell-Oliver

Department of Computer Science
University of Essex
Colchester
ESSEX CO4 3SQ
England

Email: cardr@essex.ac.uk

Mike Gordon

University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
CAMBRIDGE CB2 3QG
England
Email: mjcg@cl.cam.ac.uk

Roger Hale

SRI International
Cambridge Research Centre (CRC)
Suite 23
Millers Yard
Mill Lane
CAMBRIDGE CB2 1RQ
England
Email: rwsh@cam.sri.com

John Herbert

SRI International
Cambridge Research Centre (CRC)
Suite 23
Millers Yard
Mill Lane
CAMBRIDGE CB2 1RQ
England
Email: jmjh@cam.sri.com

Prof. C.A.R. Hoare

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building
Parks Road
OXFORD OX1 3QD
England
Email: Tony.Hoare@comlab.ox.ac.uk

He Jifeng

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building
Parks Road
OXFORD OX1 3QD
England
Email: Jifeng.He@comlab.ox.ac.uk

Prof. Hans Langmaack

Christian-Albrechts Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Preußerstraße 1–9
D-24105 Kiel
Germany
Email: hl@informatik.uni-kiel.d400.de

Ian Page

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building
Parks Road
OXFORD OX1 3QD
England
Email: Ian.Page@comlab.ox.ac.uk

Paritosh Pandya

Tata Institute of Fundamental Research
Computer Science Group
TIFR, Homi Bhabha Road
Colaba
BOMBAY 400 005
India
Email: pandya@tcs.tifr.res.in

Andrew M. Pitts

University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
CAMBRIDGE CB2 3QG
England
Email: Andrew.Pitts@cl.cam.ac.uk

Anders P. Ravn

Department of Computer Science
Technical University of Denmark
Building 344Ø
DK-2800 Lyngby
Denmark
Email: apr@id.dth.dk

David Shepherd

INMOS Limited
1000 Park Avenue
Aztec Way
Almondsbury
Bristol
AVON BS12 4SQ
England
Email: des@inmos.co.uk

Victoria Stavridou

Department of Computer Science
Royal Holloway and Bedford New College
University of London
Egham Hill
Egham
SURREY TW20 0EX
England
Email: victoria@dcs.rhbnc.ac.uk

Bill Young

Computational Logic, Inc.
1717 West Sixth Street
Suite 290
Austin
Texas 78703-4776
U.S.A.
Email: young@CLI.com