

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

129

Brent T. Hailpern

Verifying
Concurrent Processes
Using Temporal Logic



Springer-Verlag
Berlin Heidelberg New York 1982

Editorial Board

W. Brauer P. Brinch Hansen D. Gries C. Moler G. Seegmüller
J. Stoer N. Wirth

Author

Brent T. Hailpern
IBM Thomas J. Watson Research Center
P.O.Box 218, Yorktown Heights, NY 10598, USA

CR Subject Classifications (1981): 5.21 5.24

ISBN 3-540-11205-7 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-11205-7 Springer-Verlag New York Heidelberg Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to "Verwertungsgesellschaft Wort", Munich.

© by Springer-Verlag Berlin Heidelberg 1982
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

Abstract

Concurrent processes can exhibit extremely complicated behavior, and neither informal reasoning nor testing is reliable enough to establish their correctness. In this thesis, we develop a new technique for the verification of parallel programs. The technique is stated in terms of axioms and inference rules, and it is used to prove safety and liveness properties of parallel programs. Safety properties are assertions that must be satisfied by the system state at all times; they are analogous to partial correctness. Liveness properties refer to events that will occur in the future, such as program termination or the eventual receipt of a message. In addition to the formal proof rules, we present several heuristics to aid in the preparation of correctness proofs.

We model a parallel program as a set of interacting modules (processes and monitors), and we exploit this modularity in the verification process. First we prove properties of the low-level modules directly from their code. We then combine the specifications of the low-level modules to prove properties of higher-level modules, without again referring to the code. Eventually, we prove properties of the entire program.

We discuss the application of this verification technique to two classes of parallel programs: network protocols and resource allocators. Most previous approaches to verifying network protocols have been based upon reachability arguments for finite-state models of the protocols. Only protocols of limited complexity can be verified using the finite-state model, because of the combinatorial explosion of the state space as the complexity of the protocol increases. In contrast, our approach allows us to abstract information from the details of the implementation, so that the proof need not grow unmanageably as the protocol size increases.

The discussion of resource allocation centers around Hoare's structured paging system, which is a complex hierarchical program. With this example, we demonstrate that many of the techniques used in program verification can be used for specification as well.

The thesis also describes a number of tools that have been useful in proving concurrent programs. Two of the most important are history variables and temporal logic. We employ history variables to record the interaction between the modules that constitute a program. Temporal logic serves as a convenient notation for stating and proving liveness properties.

Acknowledgments

Years ago, my parents taught me to love learning. I thank them for that lesson and for their love; for without either one, this thesis would never have been written.

Many people at Stanford contributed in one way or another to this thesis. John Hennessy and Gio Wiederhold, as members of my reading committee, provided me with many useful comments and suggestions. John Gilbert, Jim Boyce, and Shel Finkelstein helped me understand the beauty of formal logic in general and temporal logic in particular. Robert Tarjan guided me through my first year at Stanford with his wisdom and his inexhaustable common sense. To these people and to the faculty, staff, and students of the Stanford Computer Science Department, I express my gratitude.

I want give special thanks to two of my dearest friends: David Wall and Richard Pattis. They provided constant professional and emotional support during my entire graduate career. The two of them were always there when I needed advice, someone to listen to me complain, a sounding board, criticism, or companionship. They contributed greatly to this thesis by reading various drafts and providing numerous comments.

There are no words to express my gratitude to Susan Owicki, my advisor. She introduced me to program verification and to temporal logic. Her ideas, comments, advice, and suggestions form an integral part of this thesis. It has been a great honor to have known her and to have worked with her.

I dedicate this thesis to my wife, Susan. Her love, support, and understanding gave me the strength to write this thesis. I am in her debt for the many hours I spent working when we could have been together, and I look forward to spending the rest of my life discharging that debt.

My research was supported by a number of agencies, and I gratefully acknowledge their generosity. My graduate education was funded primarily by fellowships from the National Science Foundation and the Fannie and John Hertz Foundation. I received additional support from teaching assistantships in the Stanford Computer Science Department and Computer System Laboratory and from research assistantships with the S-1 project and my advisor. (The S-1 project is supported at Lawrence Livermore Laboratory of the University of California by the Department of the Navy via ONR Order No. N00014-78-F0023. Research with my advisor was supported by the Joint Services Electronics Project, under contract N-00014-75-C-0601. JSEP also funded my travel expenses in connection with this research.)

Table of Contents

Abstract	
Acknowledgments	
Table of Contents	
List of Figures	
Chapter 1. Introduction	1
Chapter 2. Programming Environment	3
2.1. Pascal-like Constructs	3
2.2. Processes and Monitors	5
Chapter 3. Verification	15
3.1. Floyd	16
3.2. Hoare	18
3.3. Manna and Pnueli	20
3.4. Owicki and Gries	22
3.5. Modularity	23
Chapter 4. Temporal Logic	27
4.1. Introduction	27
4.2. Syntax	29
4.3. Axioms	30
4.4. Semantics	31
4.5. History of Temporal Logic	33
Chapter 5. Techniques	36
5.1. Basic Tools	36
5.1.1. Safety Inference Rules—Sequential VALET	37
5.1.2. Invariants	37
5.1.3. Auxiliary Variables	39
5.1.4. Safety Properties of Parallel VALET	44
5.1.5. Commitments and Liveness Inference Rules	45
5.1.6. Monitor and Process Specifications	49
5.2. Heuristics	51

Chapter 6. Network Protocols	70
6.1. Introduction	70
6.2. Verification	72
6.3. The Communication Medium	74
6.4. The Alternating Bit Protocol	77
6.5. Stenning's Data Transfer Protocol (Simplified Version)	96
6.6. Stenning's Data Transfer Protocol (Full Version)	108
6.7. Brinch Hansen's Network	118
6.7.1. Overview	120
6.7.2. Safety: Communication Medium	123
6.7.3. Safety: Node Components	125
6.7.4. Safety: Node	133
6.7.5. Safety: The Relationship Between the Nodes	138
6.7.6. Safety: System	142
6.7.7. Liveness: Communication Medium	146
6.7.8. Liveness: Node Components	147
6.7.9. Liveness: Node	151
6.7.10. Liveness: No Blocking	153
6.7.11. Liveness: System	154
Chapter 7. Resource Allocation	157
7.1. Introduction	157
7.2. Hoare's Structured Paging System	158
7.2.1. Overview	159
7.2.2. Resource Allocators: MFree and DFree	163
7.2.3. Main Store	165
7.2.4. Drum Module	166
7.2.5. Virtual Memory Specifications	170
7.2.6. Virtual Memory Implementation	172
7.2.7. Virtual Page Implementation	175
Chapter 8. Conclusion	183
Appendix A. Temporal Logic (Derived Theorems)	185
Appendix B. Hoare's Structured Paging System: The Program	194
References	203

List of Figures

Chapter 2. Programming Environment

2.1-1	Pascal-like Constructs in VALET	4
2.1-2	Array Initialization	6
2.2-1	A Buffer System	8
2.2-2	Module Procedures	9
2.2-3	Module Procedures	10
2.2-4	Shared Stack Monitor	12
2.2-5	One Hundred Shared Stacks	14

Chapter 3. Verification

3.1-1	Flowchart of Program to Compute $\sum_{j=1}^n a_j$	17
3.2-1	Division by Repeated Subtraction	19
3.2-2	Proof of Division Algorithm	19

Chapter 5. Techniques

5.1-1	Safety Axiom and Inference Rules for Sequential VALET	38
5.1-2	Shared Stack Monitor	42
5.1-3	Revised Push and Pop Procedures	43
5.1-4	Live-assertions for Sequential VALET	46
5.2-1	Unbounded Buffer	53
5.2-2	Unbounded Buffer with History Variables	54
5.2-3	Bounded Buffer System	58
5.2-4	Bounded Buffer	59
5.2-5	Semaphores	63
5.2-6	Semaphores with Auxiliary Variables	64
5.2-7	Proof of Invariant 2	66
5.2-8	Proof of Invariant 3	67

Chapter 6. Network Protocols

6.2-1	The Alternating Bit Protocol	73
6.4-1	System Diagram for the Alternating Bit Protocol	78
6.4-2	Alternating Bit Protocol: Process A	79
6.4-3	Alternating Bit Protocol: Process B	80
6.4-4	Proof of A1	84
6.4-5	Parity and Corruption	88
6.4-6	Proof of A8	91

6.5-1	Stenning's Data Transfer Protocol	97
6.5-2	Stenning's Data Transfer Protocol: Transmitter	98
6.5-3	Stenning's Data Transfer Protocol: Receiver	99
6.6-1	Stenning's Data Transfer Protocol: Transmitter	110
6.6-2	Stenning's Data Transfer Protocol: Receiver	111
6.7-1	Brinch Hansen's Network	119
6.7-2	Node	122
6.7-3	Medium	124
6.7-4	Reader	126
6.7-5	Writer	126
6.7-6	Buf Monitor	128
6.7-7	Inputs Monitor	130
6.7-8	Outputs Monitor	131
6.7-9	Table of Submodule Invariants	135
6.7-10	Table of Node Invariants	143
Chapter 7. Resource Allocation		
7.2-1	Hoare's Structured Paging System	161
7.2-2	The Drum Module	167
7.2-3	The Virtual Memory Module	173
7.2-4	The Virtual Page Module	176