

Lecture Notes in Computer Science

908

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Josyula Ramachandra Rao

Extensions of the UNITY Methodology

Compositionality, Fairness
and Probability in Parallelism



Springer

Series Editors

Gerhard Goos

Universität Karlsruhe

Vincenz-Priessnitz-Straße 3, D-76128 Karlsruhe, Germany

Juris Hartmanis

Department of Computer Science, Cornell University

4130 Upson Hall, Ithaca, NY 14853, USA

Jan van Leeuwen

Department of Computer Science, Utrecht University

Padualaan 14, 3584 CH Utrecht, The Netherlands

Author

Josyula Ramachandra Rao

IBM Thomas J. Watson Research Center

P. O. Box 704, Yorktown Heights, NY 10598, USA

CR Subject Classification (1991): D.1.3, D.2.2, D.3, F.3, G.3

ISBN 3-540-59173-7 Springer-Verlag Berlin Heidelberg New York

CIP data applied for

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1995

Printed in Germany

Typesetting: Camera-ready by author

SPIN: 10485587

06/3142-543210 - Printed on acid-free paper

Preface

The UNITY methodology marks an important milestone in research on program verification. The methodology shows how a simple programming notation and a small set of carefully engineered operators can be used to reason effectively about a wide variety of parallel programs. The goal of this treatise is to push these ideas further in order to explore and understand the limitations of this approach. We attempt to do this in three ways. First, we apply UNITY to formulate and tackle problems in parallelism such as compositionality. Second, we extend and generalize the notation and logic of UNITY in an attempt to increase its range of applicability. Finally, we develop paradigms and abstractions useful for algorithm design. We summarize our contributions below.

In designing a system of processes, it is desirable to have a guarantee that the progress made by each individual process is inherited by the system as a whole. Such a guarantee would aid in developing parallel programs in a compositional way. We use UNITY logic to formulate such a guarantee and use formal methods to derive sufficient (and yet practical) conditions for the guarantee to hold. These conditions require process interactions to obey certain commutativity conditions. Our restrictions permit compositional reasoning about progress properties of parallel programs and provide a rigorous justification for including certain syntactic features in parallel programming languages.

The nature of fairness assumed in executing a UNITY program determines the progress properties that can be proven of the program. Our second contribution is a uniform framework for the *systematic* design of proof rules for proving progress under a spectrum of fairness assumptions ranging from pure nondeterminism to strong fairness. Proofs of soundness and relative completeness of the synthesized rules follow by checking a set of simple conditions. Unlike existing work in this area, our proofs do not use ordinals.

One special notion of fairness that is being increasingly used by algorithm designers is that associated with tossing a coin. Of late, programmers have started using probabilistic transitions in designing simple and efficient algorithms for problems that may not have a deterministic solution. We generalize UNITY program to permit probabilistic transitions and develop a UNITY-like theory to design and prove the correctness of probabilistic parallel programs. We illustrate our theory with examples from random walks and mutual exclusion.

Finally, we propose a new paradigm for the design of probabilistic parallel programs called *eventual determinism*. The paradigm provides a means of com-

binning probabilistic and deterministic algorithms to take advantage of both. The proofs of such algorithms use the probabilistic generalization of UNITY. We illustrate the paradigm with examples from conflict-resolution and self-stabilization.

Our investigations and results reaffirm the promise of UNITY: we conclude that it provides a versatile medium for posing and solving many of the diverse problems of parallelism.

Acknowledgements: This book is based on my doctoral dissertation which was completed at the University of Texas at Austin in August 1992. The work reported here has been deeply influenced by discussions with several people and I would like to take this opportunity to thank some of them.

I owe an enormous debt of gratitude to my supervisor, Professor J. Misra and to Professor Edsger W. Dijkstra. I have been greatly influenced by their taste in research topics and their clarity of thought. In writing this book, I have tried to achieve their conception of simplicity and elegance while aspiring for their high standards of rigor and excellence. I have also had the privilege of improving my work through discussions with Professor C. A. R. Hoare. I will remain indebted to him for his valuable criticisms and timely words of encouragement.

Chapters 4 and 5 of this thesis represent collaborative work with two of my colleagues, Ernie Cohen and Charanjit Jutla respectively: it has been a pleasure to work with them. I would also like to thank my colleagues in the UNITY group and the Distributed Systems Discussion Group at Austin especially Mike Barnett, Ken Calvert, Ted Herman, and Dave Naumann.

I gratefully acknowledge the financial support that I received from the Office of Naval Research, the Texas Advanced Research Program, the National Science Foundation, and the University of Texas at Austin.

I have been extremely fortunate to have enjoyed the company of good friends at all stages of my life. In particular, I would like to thank Asoke Chattopadhyaya, Vipin Chaudhary, Leena and Manoj Dharwadkar, T. Krishnaprasad, Lyn and David Loewi, Linda Mohusky, Vijaya and K. Muthukumar and Bikash Sabata. Finally, I will remain indebted to my parents, my sisters, Surya and Sundari, and my wife, Sailaja, for their love and emotional support.

Yorktown Heights, New York
December 1994

Josyula R. Rao

Table of Contents

1. Prologue	3
1.1 Background and Motivation	3
1.2 Contributions of this Treatise	6
1.2.1 The Role of Commutativity in Parallel Program Design	6
1.2.2 On the Design of Proof Rules for Fair Parallel Programs	7
1.2.3 Reasoning About Probabilistic Parallel Programs	7
1.2.4 Eventual Determinism: Using Probabilistic Means to Achieve Deterministic Ends	8
1.3 Overview of this Treatise	9
2. Preliminaries	11
2.1 Notation and Terminology	11
2.2 Predicate Transformers and Their Junctionivity Properties	12
2.3 Some Useful Predicate Transformers	13
2.4 Extremal Solutions of Equations	14
2.5 Proof Format	15
3. An Introduction to UNITY	17
3.1 The Programming Notation	17
3.1.1 The Declare Section	17
3.1.2 The Always Section	18
3.1.3 The Initially Section	18
3.1.4 The Assign Section	19
3.2 Executing a UNITY Program	24
3.3 The UNITY Programming Theory	24
3.3.1 Reasoning About Safety	25
3.3.2 Reasoning About Progress	27
3.3.3 Remark on Presentation	30
3.3.4 Substitution Axiom	30
3.3.5 Program Composition in UNITY	30
4. The Role of Commutativity in Parallel Program Design	33
4.1 The Problem with Composing Progress Properties	36
4.2 Loose Coupling	37
4.3 Towards a Theory of Decoupling	38

4.3.1	The Closure of a Program	38
4.3.2	Decoupling in Terms of Progress	42
4.3.3	Decoupling in Terms of Ensures	45
4.3.4	Decoupling in Terms of Stability	46
4.3.5	A Special Case of Decoupling: Weak Decoupling	49
4.3.6	Summary	52
4.4	Existing Definitions of Commutativity	52
4.4.1	Lipton's Definition	53
4.4.2	Misra's Definition	53
4.4.3	Incomparability of Lipton and Misra Commutativity	54
4.5	Putting it All Together	54
4.6	Implications for Research in Programming Languages	57
5.	On the Design of Proof Rules for Fair Parallel Programs	59
5.1	Logics of Programs	64
5.1.1	Model of Program Execution	64
5.1.2	Temporal Logic	64
5.1.3	The μ -Calculus	67
5.2	Methodology for the Design of Proof Rules	68
5.3	From Temporal Logic to μ -Calculus	69
5.3.1	Minimal Progress	69
5.3.2	Weak Fairness	71
5.3.3	Strong Fairness	75
5.4	From μ -Calculus to UNITY-Style Proof Rules	80
5.4.1	A Predicate Transformer Approach	80
5.4.2	A Relational Approach	82
5.4.3	Constraints on <i>leads-to</i> and <i>wlt</i>	83
5.4.4	The Predicate Transformer <i>led</i>	84
5.4.5	Relating <i>leads-to</i> and <i>wlt</i>	84
5.4.6	Summary	86
5.5	Proof Rules	86
5.5.1	Defining \mathcal{E} from <i>gwp.s</i>	86
5.5.2	Minimal Progress	88
5.5.3	Weak Fairness	89
5.5.4	Strong Fairness	89
5.6	Examples	90
5.7	On a Notion of Completeness of <i>leads-to</i>	94
5.7.1	Reviewing Completeness	95
5.7.2	Constructing a Proof of Progress	96
6.	Reasoning About Probabilistic Parallel Programs	99
6.1	The Programming Model	103
6.1.1	Deterministic Statements	104
6.1.2	Probabilistic Statements	104
6.1.3	Executing a UNITY Program	104
6.1.4	Executing a Probabilistic Parallel Program	105

6.2	The Weakest Precondition	107
6.2.1	Deterministic Statements	107
6.2.2	Probabilistic Statements	107
6.3	Reasoning About Safety	109
6.4	UNITY and Progress: ensures and \mapsto	110
6.5	Deterministic Versus Probabilistic Correctness	113
6.6	The Weakest Probabilistic Precondition	115
6.7	Relating wp and wpp	117
6.8	Reasoning About Progress	118
6.8.1	The Relation upto	118
6.8.2	The Relation entails	124
6.8.3	The Relation \rightsquigarrow	129
6.8.4	Probabilistically Leads-to: $\vdash\Rightarrow$	131
6.9	An Induction Principle for Probabilistic Leads-to	134
6.10	Substitution Axiom	135
6.11	On Program Composition	135
6.11.1	Composition by Union	135
6.11.2	Conditional Properties	138
6.11.3	Superposition	138
6.12	Comments on Soundness and Completeness	138
6.13	Examples	139
7.	Eventual Determinism: Using Probabilistic Means to Achieve Deterministic Ends	149
7.1	The Symmetric Dining Philosophers Problem	151
7.1.1	Notation and Variable Declarations	151
7.1.2	The Lehmann–Rabin Algorithm	152
7.1.3	The Chandy–Misra Algorithm	153
7.1.4	The Eventually–Determinizing Algorithm	155
7.2	The Self–Stabilization Problem	161
7.2.1	Notation and Variable Declarations	161
7.2.2	A Probabilistic Algorithm	162
7.2.3	The Eventually–Determinizing Algorithm	162
8.	Epilogue	165
8.1	Conclusions	165
8.2	Topics for Future Research	166
	Bibliography	169
	Index	175