

A. Lingas R. Karlsson S. Carlsson (Eds.)

001-700

# Automata, Languages and Programming

20th International Colloquium, ICALP 93  
Lund, Sweden, July 5-9, 1993  
Proceedings



Springer-Verlag

Berlin Heidelberg New York  
London Paris Tokyo  
Hong Kong Barcelona  
Budapest

## Series Editors

Gerhard Goos  
 Universität Karlsruhe  
 Postfach 69 80  
 Vincenz-Priessnitz-Straße 1  
 D-76131 Karlsruhe, FRG

Juris Hartmanis  
 Cornell University  
 Department of Computer Science  
 4130 Upson Hall  
 Ithaca, NY 14853, USA

## Volume Editors

Andrzej Lingas  
 Rolf Karlsson  
 Department of Computer Science, Lund University  
 Box 118, S-22100 Lund, Sweden

Svante Carlsson  
 Department of Computer Science, University of Luleå  
 S-95187 Luleå, Sweden

CR Subject Classification (1991): F, D.1, E.1, E.3, G.2, I.3.5

ISBN 3-540-56939-1 Springer-Verlag Berlin Heidelberg New York

ISBN 0-387-56939-1 Springer-Verlag New York Berlin Heidelberg

5308

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 1993  
 Printed in Germany

Typesetting: Camera ready by author  
 Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.  
 45/3140-543210 - Printed on acid-free paper

## FOREWORD

The International Colloquium on Automata, Languages and Programming (ICALP) is an annual conference series sponsored by the European Association for Theoretical Computer Science (EATCS). It is intended to cover all important areas of theoretical computer science, such as: computability, automata, formal languages, term rewriting, analysis of algorithms, computational geometry, computational complexity, symbolic and algebraic computation, cryptography, data types and data structures, theory of data bases and knowledge bases, semantics of programming languages, program specification, transformation and verification, foundations of logic programming, theory of logical design and layout, parallel and distributed computation, theory of concurrency, and theory of robotics.

ICALP 93 was held at Lund University, Sweden, from July 5 to July 9, 1993.

Previous colloquia were held in Wien (1992), Madrid (1991), Warwick (1990), Stresa (1989), Tampere (1988), Karlsruhe (1987), Rennes (1986), Nafplion (1985), Antwerp (1984), Barcelona (1983), Aarhus (1982), Haifa (1981), Amsterdam (1980), Graz (1979), Udine (1978), Turku (1977), Edinburgh (1976), Saarbrücken (1974) and Paris (1972). ICALP 94 will be held in Jerusalem from July 11 to July 15, 1994.

The number of papers submitted was 151. Each submitted paper was sent to at least four Programme Committee members, who were often assisted by their referees. The Programme Committee meeting took place at Lund University on the 5th and 6th of February 1993 (the names of those participating in the meeting are underlined below). This volume contains the 51 papers selected at the meeting plus the five invited papers.

We would like to thank all the Programme Committee members and the referees who assisted them in their work. The list of referees is as complete as we can achieve and we apologize for any omissions or errors.

The members of the Organizing Committee, the members of the algorithm group at our department sometimes assisted by their families, our departmental secretaries, and many other members of the department deserve our gratitude for their contributions throughout the preparations.

We also gratefully acknowledge support from Swedish Natural Science Research Council, Apple Computer AB, the Department of Computer Science at Lund University, Lund Institute of Technology, Lund University, and the city of Lund.

Svante Carlsson, Rolf Karlsson, and Andrzej Lingas  
*April 1993, Lund University*

### Invited Lecturers

S. Abiteboul, INRIA Rocquencourt;  
 M. Blum, Berkeley;  
 D. Dolev, IBM San José and Jerusalem;  
 L. Hemachandra, Rochester;  
 I. Simon, Sao Paulo.

### Programme Committee

S. Abiteboul, Paris  
J. Diaz, Barcelona  
R. Freivalds, Riga  
F. Gécseg, Szeged  
 G. Gonnet, Zurich  
 Y. Gurevich, Ann Arbor  
 D. Harel, Rehovot  
T. Harju, Turku  
 I. M. Havel, Prague  
J. Håstad, Stockholm  
J.-P. Jouannaud, Paris  
 D. Kirkpatrick, Vancouver  
H.-J. Kreowski, Bremen  
W. Kuich, Vienna  
G. Levi, Pisa  
A. Lingas, Lund (chairman)  
T. Maibaum, London  
 A. Mazurkiewicz, Warsaw  
M. Nielsen, Aarhus  
 M. H. Overmars, Utrecht  
W. Thomas, Kiel  
 U. Vishkin, Tel-Aviv and Maryland  
 P. Wolper, Liege

### Organizing Committee

Arne Andersson, Lund;  
 Svante Carlsson, Luleå (co-chairman);  
 Rolf Karlsson, Lund (co-chairman);  
 Andrzej Lingas, Lund;  
 Ola Petersson, Växjö.

## List of Referees

Abrahamson K.	Fülöp Z.	Lazard D.	Sales T.
Alur R.	Gabarró J.	Leduc G.	Salomaa A.
Anderson R.	Gallo G.	Leroy X.	Sannella D.
Andersson A.	Garrido O.	Levcopoulos C.	Sassone V.
Anstee R.P.	Gavaldá R.	Liestman A.	Schicber B.
Arnborg S.	Giacobazzi R.	Luccio F.	Schmid U.
Arnold A.	Godefroid P.	Maggs B.M.	Schmidt E.M.
Baaz M.	Goldmann M.	Magnusson B.	Sécébold P.
Balcázar J.L.	Goldreich O.	Manorioci D.	Seibert S.
Bang-Jensen J.	Goltz U.	Manovssakis Y.	Seidl H.
Bartha M.	Graedel E.	Martini S.	Serna M.
Bcauquier D.	Grandjean E.	Masini A.	Skyum S.
Berman P.	Grosse-Rhode M.	Matias Y.	Staiger L.
Bernot G.	Habel A.	Mattsson C.	Steffen B.
Berry G.	Hajnal P.	Milo T.	Steinby M.
Best E.	Hankin C.L.	Miltersen P.B.	Stevenne J.-M.
Bhattacharia B.	Hansen M.R.	Moggi E.	Steyaert J.-M.
Boasson L.	Higham L.	Montanari U.	Stirling C.
Bodlaender H.L.	Hodkinson I.M.	Moreau L.	Storlind R.
Bonatti P.	Honkala J.	Moscowitz Y.	Stout Q.
Breazu-Tannen V.	Hortmann M.	Mossakowski T.	Taylor Paul
Carlsson S.	Horváth G.	Nagarajan R.	Tel G.
Casas R.	Inverardi P.	Nickl F.	Thurimella R.
Cassaigne J.	JaJa J.	Niemi V.	Tison
Choffrut C.	Jancar P.	Nilsson B.	Torán J.
Comon	Jantzen M.	Nilsson S.	Treinen R.
Compton K.	Jebclean T.	Olderog E.-R.	Tsantilas T.
Courcelle B.	Jenner B.	Pacholski L.	Turán G.
Craig T.S.	Jennings E.	Padawitz P.	Upfal E.
Crochemore N.	Juras M.	Paz A.	Valkema E.
Csakany B.	Kameda T.	Petersson O.	Varricchio S.
Csirik J.	Karhumäki J.	Pin J.E.	Vianu V.
Culberson J.	Kari J.	Pippenger N.	Vickers S.J.
Damgaard I.B.	Kari L.	Plump D.	Wagner A.
Delorme	Karlsson R.	Poigné A.	Weber A.
Dessmark A.	Kasif S.	Potthoff A.	Wiehagen R.
Diaconescu	Khuller S.	Prodinger H.	Wilke T.
Diekert V.	Klarlund N.	Qian Z.	Willem J.
Dovier A.	Klop J.W.	Rajasekaran S.	Winkel G.
Drewes F.	Kluge W.	Raman R.	Wojciechowski S.
Droste M.	Knoop J.	Raz D.	Woo Ryu K.
Emerson E.A.	Koubek V.	Regnier M.	Yesha Y.
Engberg U.H.	Krivánek M.	Renvall A.	Young N.
Enjalbert P.	Kudlek M.	Reynolds M.A.	Zielonka W.
Farvardin N.	La Poutre J.A.	Richter M.M.	
Frandsen G.	Lagergren J.	Ruthing O.	
Fuchs N.E.	Landau G.M.	Sahinalp S.	



# TABLE OF CONTENTS

## Session 1: Programs and Data Structures

Program Result Checking: A New Approach to Making Programs More Reliable (Invited)	
<i>M. Blum</i> .....	1
Dynamic Interpolation Search in $o(\log \log n)$ Time	
<i>A. Andersson and C. Mattsson</i> .....	15
Searching Among Intervals and Compact Routing Tables	
<i>G.N. Frederickson</i> .....	28

## Session 2: Approximation Complexity

The Approximation of Maximum Subgraph Problems	
<i>C. Lund and M. Yannakakis</i> .....	40
Polynomially Bounded Minimization Problems Which are Hard to Approximate	
<i>V. Kann</i> .....	52
Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees, with Applications to Matching and Set Cover	
<i>N. Garg, V. V. Vazirani, and M. Yannakakis</i> .....	64
The Complexity of Approximating PSPACE-Complete Problems for Hierarchical Specifications	
<i>M.V. Marathe, H.B. Hunt III, and S.S. Ravi</i> .....	76

## Session 3: Graph Algorithms

Problems on Pairs of Trees and the Four Colour Problem of Planar Graphs	
<i>A. Czumaj and A. Gibbons</i> .....	88
Constructing Competitive Tours from Local Information	
<i>B. Kalyanasundaram and K.R. Pruhs</i> .....	102
Treewidth and Pathwidth of Permutation Graphs	
<i>H. Bodlaender, T. Kloks, and D. Kratsch</i> .....	114
A Theory of Even Functionals and Their Algorithmic Applications	
<i>J.W. Jaromczyk and G. Świątek</i> .....	126

## Session 4: Algorithm Analysis and Computational Geometry

Exact Asymptotics of Divide-and-Conquer Recurrences	
<i>P. Flajolet and M. Golin</i> .....	137
Optimal Bounds for the Change-Making Problem	
<i>D. Kozen and S. Zaks</i> .....	150

The Complexity of $N$ -body Simulation <i>J.H. Reif and S.R. Tate</i> .....	162
A Simple Method for Resolving Degeneracies in Delaunay Triangulations <i>M.B. Dillencourt and W.D. Smith</i> .....	177
<b>Session 5: Complexity</b>	
Fault-Tolerance and Complexity (Invited) <i>L.A. Hemachandra</i> .....	189
Reversal-Space Trade-offs for Simultaneous Resource-Bounded Nondeterministic Turing Machines <i>H. Yamamoto</i> .....	203
On the Computational Power of Discrete Hopfield Nets <i>P. Orponen</i> .....	215
<b>Session 6: Probabilistic Complexity and Cryptography</b>	
On Randomized Versus Deterministic Computation <i>M. Karpinski and R. Verbeek</i> .....	227
Lower Bounds for One-Way Probabilistic Communication Complexity <i>F. Ablayev</i> .....	241
Maintaining Discrete Probability Distributions Optimally <i>T. Hagerup, K. Mehlhorn, and I. Munro</i> .....	253
Secure and Efficient Off-Line Digital Money <i>M. Franklin and M. Yung</i> .....	265
<b>Session 7: Computability, Formal Languages and Automata</b>	
Computational Depth and Reducibility <i>D.W. Juedes, J.I. Lathrop, and J.H. Lutz</i> .....	277
Learnability: Admissible, Co-Finite, and Hypersimple Languages <i>G. Baliga and J. Case</i> .....	289
Inclusion is Undecidable for Pattern Languages <i>T. Jiang, A. Salomaa, K. Salomaa, and S. Yu</i> .....	301
New Decidability Results Concerning Two-Way Counter Machines and Applications <i>O.H. Ibarra, T. Jiang, N. Tran, and H. Wang</i> .....	313
<b>Session 8: Logic, Formal Languages and Automata</b>	
Cobham's Theorem Seen Through Büchi's Theorem <i>C. Michaux and R. Villemaire</i> .....	325



Logical Definability on Infinite Traces <i>W. Ebinger and A. Muscholl</i> .....	335
Algebras for Classifying Regular Tree Languages and an Application to Frontier Testability <i>T. Wilke</i> .....	347
Finite Automata as Characterizations of Minor Closed Tree Families <i>A. Gupta</i> .....	359
<b>Session 9: Parallel and Distributed Algorithms I</b>	
On Distributed Algorithms in a Broadcast Domain (Invited) <i>D. Dolev and D. Malki</i> .....	371
Sparse Networks Supporting Efficient Reliable Broadcasting <i>B.S. Chlebus, K. Diks, and A. Pelc</i> .....	388
<b>Session 10: Parallel and Distributed Algorithms II</b>	
Strongly Adaptive Token Distribution <i>F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka</i> .....	398
Fast Parallel Computation of Characteristic Polynomials by Leverrier's Power Sum Method Adapted to Fields of Finite Characteristic <i>A. Schönhage</i> .....	410
Fast Parallel Constraint Satisfaction <i>L.M. Kirousis</i> .....	418
<b>Session 11: Algebraic Aspects of Formal Languages and Automata I</b>	
The Product of Rational Languages (Invited) <i>I. Simon</i> .....	430
On Regular Compatibility of Semi-Commutations <i>E. Ochmanski and P.-A. Wacrenier</i> .....	445
Algebraic Aspects of B-Regular Series <i>P. Dumas</i> .....	457
<b>Session 12: Algebraic Aspects of Formal Languages and Automata II</b>	
Products of Finite State Machines with Full Coverage <i>D.M. Cohen and M.L. Fredman</i> .....	469
An Effective Version of Stallings' Theorem in the Case of Context-Free Groups <i>G. Sénizergues</i> .....	478
On the Power of Periodic Iteration of Morphisms <i>A. Lepistö</i> .....	496
If a DOL Language is k-Power Free then It is Circular <i>F. Mignosi and P. Séébold</i> .....	507

**Session 13: Concurrency**

Deciding True Concurrency Equivalences on Finite Safe Nets <i>L. Jategaonkar and A.R. Meyer</i> .....	519
Timed Testing of Concurrent Systems <i>W. Vogler</i> .....	532
The Fork Calculus <i>K. Havelund and K. G. Larsen</i> .....	544
Extended Transition Systems for Parametric Bisimulation <i>P. Inverardi, C. Priami, and D. Yankelevich</i> .....	558

**Session 14: Temporal Logic**

Temporal Logic and Categories of Petri Nets <i>C. Brown and D. Gurr</i> .....	570
Decidability of a Partial Order Based Temporal Logic <i>K. Lodaya, R. Ramanujam, and P.S. Thiagarajan</i> .....	582
Local Model Checking for Context-Free Processes <i>H. Hunger and B. Steffen</i> .....	593

**Session 15: Theory of Programming I**

Computing on Structures (Invited) <i>S. Abiteboul and V. Vianu</i> .....	606
A Partial Solution for D-Unification Based on a Reduction to AC1-Unification <i>E. Contejean</i> .....	621
Efficient Analysis of Concurrent Constraint Logic Programs <i>M. Codish, M. Fulaschi, K. Marriott, and W. Winsborough</i> .....	633

**Session 16: Theory of Programming II**

A Confluent Reduction for the Extensional Typed $\lambda$ -Calculus with Pairs, Sums, Recursion and Terminal Object <i>R. Di Cosmo and D. Kesner</i> .....	645
Modularity of Termination and Confluence in Combinations of Rewrite Systems with $\lambda_w$ <i>F. Barbanera and M. Fernández</i> .....	657
From Domains to Automata with Concurrency <i>F. Bracho and M. Droste</i> .....	669
What Is a Universal Higher-Order Programming Language? <i>R. Kanneganti and R. Cartwright</i> .....	682
Author Index .....	696

# Program Result Checking: A New Approach to Making Programs More Reliable

Manuel Blum<sup>1</sup>

Computer Science Division  
University of California at Berkeley  
94720

**Abstract.** Program result checking is concerned with designing programs to check their work. For example, after solving an equation for  $x$ , a result-checking program would substitute  $x$  back into the equation to make sure that the answer obtained is correct. There are many ways to check results, but there has been no theory to say what constitutes a good check. It is not a good check, for example, to redo a computation without change a second time. Such recomputation may uncover an intermittent hardware fault, but it will not uncover a software fault, and the discovery and elimination of software faults is the principal goal of this work. This talk discusses the concept of result checking, gives several examples, and outlines the basic theory.

## 1. Introduction

This talk restricts attention to program result checkers for a certain clean class of computational problems. These problems are completely described by specifying what constitutes correct input/output and acceptable running time, i.e. by describing

1. what is an allowable *input* (to any program for the problem),
2. for each such input, what is an acceptable (i.e. correct) *output*, and
3. what is an (easily computable) upper bound on the running time of a (reasonably fast) program for the problem.

For each (such) computational problem  $\pi$ , discussion will center on two classes of programs. The first will be a (possibly faulty) program for solving problem  $\pi$ . The second will be a (possibly faulty) checker for  $\pi$ . The latter *result checker* for computational problem  $\pi$  is an efficient program for checking the correctness of *any given program* (supposedly) for problem  $\pi$  on *any particular input* to the program. Notice that a checker does *not* check that the given program for  $\pi$  is correct on *all* inputs, but only that it is correct on the *given* input. This is one reason why it is generally easier to *check* a program (on a given input) than it is to *verify* a program (*prove* the program correct on all inputs).

---

<sup>1</sup> This work was supported in part by NSF grant CCR92-01092, in part by IBM Research Division, T.J. Watson Research Center, and in part by ICSI, the International Computer Science Institute, Berkeley, CA.

A preliminary version of this paper appeared in Blum and Raghavan [BR].

Author's e-mail address: blum@cs.berkeley.edu

More precisely, a *result checker* for problem  $\pi$  is an algorithm that is supplied with:

1. a (possibly faulty) program for solving computational problem  $\pi$ , given as a black box, which the checker may run on inputs of its choice; and
2. a particular input for this program.

The checker must determine either that the output is correct or the program is faulty. (It may be that *both* are true!) To this end, the checker is designed to do efficient computations that are permitted to include calling the given program on various allowable inputs of the checker's own choosing. If the checker outputs **CORRECT**, then this implies that the checker has verified that the output of the given program on the given input is correct; if it outputs **FAULTY**, then this implies that the checker has verified the existence of a fault in the given program. The latter fault may manifest itself when the program is run on an input different from the given input.

Notice above that when a faulty program gives a correct output despite its fault, the checker is permitted to output either **CORRECT** or **FAULTY**. In particular, the checker is permitted to output **FAULTY** even when the given program outputs a correct answer on the given input, provided the program is indeed faulty. While this may at first appear strange, it is exactly what is wanted: for example, a trivial program that gives the same answer on all inputs may be perfectly correct on the given input for *no good reason*! The checker that discovers the program is junk has good reason to declare it faulty.

## 2. Program Result Checkers

Let  $\pi$  denote a (computational) decision and/or search problem. For  $x$  an input to  $\pi$ , let  $\pi(x)$  denote the output of  $\pi$ . Let  $P$  be a program (supposedly for  $\pi$ ) that halts on all instances of  $\pi$ . We say that such a program  $P$ , viewed as a program for  $\pi$ , has a *fault* (or *bug*) if for some instance  $x$  of  $\pi$ ,  $P(x) \neq \pi(x)$ .

Define an (efficient) *program result checker*  $C_\pi$  for problem  $\pi$  as follows:  $C_\pi^P(I; k)$  is any probabilistic (expected-poly-time) oracle Turing machine that satisfies the following conditions, for any program  $P$  (supposedly for  $\pi$ ) that halts on all instances of  $\pi$ , for any instance  $I$  of  $\pi$ , and for any positive integer  $k$  (the so-called "security parameter") presented in unary:

1. If  $P$  has no bugs, i.e.,  $P(x) = \pi(x)$  for all instances  $x$  of  $\pi$ , then with probability<sup>2</sup>  $\geq 1 - 2^{-k}$ ,  $C_\pi^P(I; k) = \text{CORRECT}$  (i.e.,  $P(I)$  is correct).
2. If  $P(I) \neq \pi(I)$ , then with probability<sup>2</sup>  $\geq 1 - 2^{-k}$ ,  $C_\pi^P(I; k) = \text{BUGGY}$  (i.e.,  $P$  has a fault).

In the above, it is assumed that any program  $P$  for problem  $\pi$  halts on all instances of  $\pi$ . This is done in order to help focus on the problem at hand. In general, however, programs do not always halt, and the definition of a "bug" must be

<sup>2</sup> This probability is computed over the sample space of all finite sequences of coin flips that  $C$  could have tossed. In most examples, including all those in this paper, program  $P$  definitely (rather than probably) has a bug when  $C_\pi^P(I; k) = \text{BUGGY}$ .