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

Cco1-700

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

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

TABLE OF CONTENTS

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

Manuel Blum[1]

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, ie. by describing
1. what is an allowable *input* (to any program for the problem),
2. for each such input, what is an acceptable (ie. 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 π, discussion will center on two classes of programs. The first will be a (possibly faulty) program for solving problem π. The second will be a (possibly faulty) checker for π. The latter *result checker* for computational problem π is an efficient program for checking the correctness of *any given program* (supposedly) for problem π on *any particular input* to the program. Notice that a checker does *not* check that the given program for π 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).

[1] 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 π is an algorithm that is supplied with:
1. a (possibly faulty) program for solving computational problem π, 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 checkers 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 π denote a (computational) decision and/or search problem. For x an input to π, let $\pi(x)$ denote the output of π. Let P be a program (supposedly for π) that halts on all instances of π. We say that such a program P, viewed as a program for π, has a *fault* (or *bug*) if for some instance x of π, $P(x) \neq \pi(x)$.

Define an *(efficient) program result checker* C_π for problem π 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 π) that halts on all instances of π, for any instance I of π, 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 π, then with probability[2] $\geq 1 - 2^{-k}$, $C_\pi^P(I;k) = CORRECT$ (i.e., $P(I)$ is correct).
2. If $P(I) \neq \pi(I)$, then with probability[2] $\geq 1 - 2^{-k}$, $C_\pi^P(I;k) = BUGGY$ (i.e., P has a fault).

In the above, it is assumed that any program P for problem π halts on all instances of π. 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

2 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 $C_\pi^P(I;k) = BUGGY$.