Roberto Giacobazzi (Ed.)

LNCS 3148

Static Analysis

11th International Symposium, SAS 2004
Vernona, Italy, August 2004
Proceedings

Roberto Giacobazzi (Ed.)

Static Analysis

11th International Symposium, SAS 2004
Verona, Italy, August 26-28, 2004
Proceedings

Volume Editor

Roberto Giacobazzi
Università degli Studi di Verona, Dipartimento di Informatica
Strada Le Grazie 15, 37134 Verona, Italy
E-mail: roberto.giacobazzi@univr.it

Library of Congress Control Number: 2004109776

CR Subject Classification (1998): D.3.2-3, F.3.1-2, D.2.8, F.4.2, D.1

ISSN 0302-9743
ISBN 3-540-22791-1 Springer Berlin Heidelberg New York

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. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11310426 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3148

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
New York University, NY, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Preface

Static analysis is a research area aimed at developing principles and tools for verification, certification, semantics-based manipulation, and high-performance implementation of programming languages and systems. The series of Static Analysis symposia has served as the primary venue for presentation and discussion of theoretical, practical, and application advances in the area.

This volume contains the papers accepted for presentation at the 11th International Static Analysis Symposium (SAS 2004), which was held August 26–28 in Verona, Italy. In response to the call for papers, 63 contributions were submitted from 20 different countries. Following on-line discussions, the Program Committee met in Verona on May 06, and selected 23 papers, basing this choice on their scientific quality, originality, and relevance to the symposium. Each paper was reviewed by at least 3 PC members or external referees. In addition to the contributed papers, this volume includes contributions by outstanding invited speakers: a full invited paper by Thomas Henzinger (University of Califorina at Berkeley), and abstracts of the talks given by the other invited speakers, Sheila McIlraith (University of Toronto), Ehud Shapiro (Weizmann Institute) and Yannis Smaragdakis (Georgia Institute of Technology).

On the behalf of the Program Committee, the Program Chair would like to thank all the authors who submitted papers and all external referees for their careful work in the reviewing process. The Program Chair would like to thank in particular Samir Genaim, who did an invaluable, excellent job in organizing the Program Committee meeting and the structure of this volume. We would like to express our gratitude to the *Dipartimento di Informatica* and to the *Università degli Studi di Verona*, in particular to Prof. Elio Mosele (president of the university), who handled the logistical arrangements and provided financial support for organizing this event.

SAS 2004 was held concurrently with *LOPSTR 2004, International Symposium on Logic-Based Program Synthesis and Transformation*; *PEPM 2004, ACM SIGPLAN Symposium on Partial Evaluation and Program Manipulation*; and *PPDP 2004, ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. There were also several workshops in the area of programming languages. We would like to thank Sandro Etalle (LOPSTR PC Chair), Nevin Heintze and Peter Sestoft (PEPM PC Chairs), Eugenio Moggi (PPDP General Chair), Fausto Spoto (Organizing Chair), and David Warren (PPDP PC Chair) for their help in the organization aspects. Special thanks to all the members of the Organizing Committee who worked with enthusiasm in order to make this event possible and to ENDES, specifically to Anna Chiara Caputo, for the great job she did in the local organization.

Verona, June 2004 Roberto Giacobazzi

此为试读,需要完整PDF请访问: www.ertongbook.com

Organization

Program Committee

Thomas Ball	Microsoft, USA
Radhia Cousot	Ècole Polytechnique, France
Roberto Giacobazzi (Chair)	Università di Verona, Italy
Chris Hankin	Imperial College London, UK
Thomas Jensen	IRISA, France
Jens Knoop	Technische Universität Wien, Austria
Giorgio Levi	Università di Pisa, Italy
Laurent Mauborgne	École Normale Supérieure, France
Andreas Podelski	Max-Planck-Institut für Informatik, Germany
German Puebla	Technical University of Madrid, Spain
Ganesan Ramalingam	IBM, USA
Francesco Ranzato	Università di Padova, Italy
Martin Rinard	Massachusetts Institute of Technology, USA
Andrei Sabelfeld	Chalmers University of Technology, Sweden
Mary Lou Soffa	University of Pittsburgh, USA
Harald Søndergaard	University of Melbourne, Australia
Reinhard Wilhelm	Universität des Saarlandes, Germany

Steering Committee

Patrick Cousot	École Normale Supérieure, France
Gilberto Filé	Università di Padova, Italy
David Schmidt	Kansas State University, USA

Organizing Committee

Mila Dalla Preda
Samir Genaim
Isabella Mastroeni
Massimo Merro
Giovanni Scardoni
Fausto Spoto
Damiano Zanardini

Referees

Elvira Albert
M. Anton Ertl
Roberto Bagnara
Roberto Barbuti
Joerg Bauer
Michele Bugliesi
V.C. Sreedhar
Paul Caspi
Patrick Cousot
Alexandru D. Salcianu
Mila Dalla Preda
Ferruccio Damiani
Bjorn De Sutter
Bjoern Decker
Pierpaolo Degano
Nurit Dor
Manuel Fahndrich
Jérôme Feret
Gilberto Filé
Steve Fink
Bernd Finkbeiner
Cormac Flanagan
Maurizio Gabbrielli
Samir Genaim
Roberta Gori
David Grove
Daniel Hedin
Dan Hirsch
Charles Hymans
Daniel Kaestner
John Kodumal
Andreas Krall
Viktor Kuncak
Kung-Kiu Lau
Francesca Levi
Donglin Liang
Andrea Maggiolo Schettini
Isabella Mastroeni
Ken McMillan
Massimo Merro
Antoine Miné
Anders Moller
David Monniaux
Carlo Montangero
Damen Mssé
Markus Müller-Olm
Ulrich Neumerkel
Jens Palsberg
Filippo Portera
Franz Puntigam
Xavier Rival
Enric Rodríguez-Carbonell
Sabina Rossi
Salvatore Ruggieri
Andrey Rybalchenko
Rene Rydhof Hansen
Oliver Rüthing
Mooly Sagiv
Giovanni Scardoni
Dave Schmidt
Bernhard Scholz
Markus Schordan
Francesca Scozzari
Clara Segura
Helmut Seidl
Alexander Serebrenik
Vincent Simonet
Fabio Somenzi
Fausto Spoto
Zhendong Su
Francesco Tapparo
Ashish Tiwari
Thomas Wies
Sebastian Winkel
Zhe Yang
Enea Zaffanella
Damiano Zanardini
Andreas Zeller

Lecture Notes in Computer Science

For information about Vols. 1–3056

please contact your bookseller or Springer

Vol. 3172: M. Dorigo, M. Birattari, C. Blum, L. M.Gambardella, F. Mondada, T. Stützle (Eds.), Ant Colony, Optimization and Swarm Intelligence. XII, 434 pages. 2004.

Vol. 3158: I. Nikolaidis, M. Barbeau, E. Kranakis (Eds.), Ad-Hoc, Mobile, and Wireless Networks. IX, 344 pages. 2004.

Vol. 3157: C. Zhang, H. W. Guesgen, W.K. Yeap (Eds.), PRICAI 2004: Trends in Artificial Intelligence. XX, 1023 pages. 2004. (Subseries LNAI).

Vol. 3156: M. Joye, J.-J. Quisquater (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2004. XIII, 455 pages. 2004.

Vol. 3153: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), Mathematical Foundations of Computer Science 2004. XIV, 902 pages. 2004.

Vol. 3152: M. Franklin (Ed.), Advances in Cryptology – CRYPTO 2004. XI, 579 pages. 2004.

Vol. 3150: G.-Z. Yang, T. Jiang (Eds.), Medical Imaging and Virtual Reality. XII, 378 pages. 2004.

Vol. 3148: R. Giacobazzi (Ed.), Static Analysis. XI, 393 pages. 2004.

Vol. 3146: P. Érdi, A. Esposito, M. Marinaro, S. Scarpetta (Eds.), Computational Neuroscience: Cortical Dynamics. XI, 161 pages. 2004.

Vol. 3144: M. Papatriantafilou, P. Hunel (Eds.), Principles of Distributed Systems. XI, 246 pages. 2004.

Vol. 3143: W. Liu, Y. Shi, Q. Li (Eds.), Advances in Web-Based Learning – ICWL 2004. XIV, 459 pages. 2004.

Vol. 3142: J. Diaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), Automata, Languages and Programming. XIX, 1253 pages. 2004.

Vol. 3140: N. Koch, P. Fraternali, M. Wirsing (Eds.), Web Engineering. XXI, 623 pages. 2004.

Vol. 3139: F. Iida, R. Pfeifer, L. Steels, Y. Kuniyoshi (Eds.), Embodied Artificial Intelligence. IX, 331 pages. 2004. (Subseries LNAI).

Vol. 3138: A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, D.d. Ridder (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. XXII, 1168 pages. 2004.

Vol. 3136: F. Meziane, E. Métais (Eds.), Natural Language Processing and Information Systems. XII, 436 pages. 2004.

Vol. 3134: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004. XIV, 233 pages. 2004.

Vol. 3133: A.D. Pimentel, S. Vassiliadis (Eds.), Computer Systems: Architectures, Modeling, and Simulation. XIII, 562 pages. 2004.

Vol. 3131: V. Torra, Y. Narukawa (Eds.), Modeling Decisions for Artificial Intelligence. XI, 327 pages. 2004. (Subseries LNAI).

Vol. 3130: A. Syropoulos, K. Berry, Y. Haralambous, B. Hughes, S. Peter, J. Plaice (Eds.), TEX, XML, and Digital Typography. VIII, 265 pages. 2004.

Vol. 3129: Q. Li, G. Wang, L. Feng (Eds.), Advances in Web-Age Information Management. XVII, 753 pages. 2004.

Vol. 3128: D. Asonov (Ed.), Querying Databases Privately. IX, 115 pages. 2004.

Vol. 3127: K.E. Wolff, H.D. Pfeiffer, H.S. Delugach (Eds.), Conceptual Structures at Work. XI, 403 pages. 2004. (Subseries LNAI).

Vol. 3126: P. Dini, P. Lorenz, J.N.d. Souza (Eds.), Service Assurance with Partial and Intermittent Resources. XI, 312 pages. 2004.

Vol. 3125: D. Kozen (Ed.), Mathematics of Program Construction. X, 401 pages. 2004.

Vol. 3124: J.N. de Souza, P. Dini, P. Lorenz (Eds.), Telecommunications and Networking - ICT 2004. XXVI, 1390 pages. 2004.

Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), Natural Language Generation. X, 219 pages. 2004. (Subseries LNAI).

Vol. 3121: S. Nikoletseas, J.D.P. Rolim (Eds.), Algorithmic Aspects of Wireless Sensor Networks. X, 201 pages. 2004.

Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), Learning Theory. X, 648 pages. 2004. (Subseries LNAI).

Vol. 3118: K. Miesenberger, J. Klaus, W. Zagler, D. Burger (Eds.), Computer Helping People with Special Needs. XXIII, 1191 pages. 2004.

Vol. 3116: C. Rattray, S. Maharaj, C. Shankland (Eds.), Algebraic Methodology and Software Technology. XI, 569 pages. 2004.

Vol. 3114: R. Alur, D.A. Peled (Eds.), Computer Aided Verification. XII, 536 pages. 2004.

Vol. 3113: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever. X, 283 pages. 2004.

Vol. 3112: H. Williams, L. MacKinnon (Eds.), Key Technologies for Data Management. XII, 265 pages. 2004.

Vol. 3111: T. Hagerup, J. Katajainen (Eds.), Algorithm Theory - SWAT 2004. XI, 506 pages. 2004.

Vol. 3110: A. Juels (Ed.), Financial Cryptography. XI, 281 pages. 2004.

Vol. 3109: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), Combinatorial Pattern Matching. XII, 486 pages. 2004.

Vol. 3108: H. Wang, J. Pieprzyk, V. Varadharajan (Eds.), Information Security and Privacy. XII, 494 pages. 2004.

Vol. 3107: J. Bosch, C. Krueger (Eds.), Software Reuse: Methods, Techniques and Tools. XI, 339 pages. 2004.

Vol. 3106: K.-Y. Chwa, J.I. Munro (Eds.), Computing and Combinatorics. XIII, 474 pages. 2004.

Vol. 3105: S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. XVI, 304 pages. 2004.

Vol. 3104: R. Kralovic, O. Sykora (Eds.), Structural Information and Communication Complexity. X, 303 pages. 2004.

Vol. 3103: K. Deb, e. al. (Eds.), Genetic and Evolutionary Computation – GECCO 2004. XLIX, 1439 pages. 2004.

Vol. 3102: K. Deb, e. al. (Eds.), Genetic and Evolutionary Computation – GECCO 2004. L, 1445 pages. 2004.

Vol. 3101: M. Masoodian, S. Jones, B. Rogers (Eds.), Computer Human Interaction. XIV, 694 pages. 2004.

Vol. 3100: J.F. Peters, A. Skowron, J.W. Grzymała-Busse, B. Kostek, R.W. Świniarski, M.S. Szczuka (Eds.), Transactions on Rough Sets I. X, 405 pages. 2004.

Vol. 3099: J. Cortadella, W. Reisig (Eds.), Applications and Theory of Petri Nets 2004. XI, 505 pages. 2004.

Vol. 3098: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets. VIII, 849 pages. 2004.

Vol. 3097: D. Basin, M. Rusinowitch (Eds.), Automated Reasoning. XII, 493 pages. 2004. (Subseries LNAI).

Vol. 3096: G. Melnik, H. Holz (Eds.), Advances in Learning Software Organizations. X, 173 pages. 2004.

Vol. 3095: C. Bussler, D. Fensel, M.E. Orlowska, J. Yang (Eds.), Web Services, E-Business, and the Semantic Web. X, 147 pages. 2004.

Vol. 3094: A. Nürnberger, M. Detyniecki (Eds.), Adaptive Multimedia Retrieval. VIII, 229 pages. 2004.

Vol. 3093: S.K. Katsikas, S. Gritzalis, J. Lopez (Eds.), Public Key Infrastructure. XIII, 380 pages. 2004.

Vol. 3092: J. Eckstein, H. Baumeister (Eds.), Extreme Programming and Agile Processes in Software Engineering. XVI, 358 pages. 2004.

Vol. 3091: V. van Oostrom (Ed.), Rewriting Techniques and Applications. X, 313 pages. 2004.

Vol. 3089: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security. XIV, 510 pages. 2004.

Vol. 3087: D. Maltoni, A.K. Jain (Eds.), Biometric Authentication. XIII, 343 pages. 2004.

Vol. 3086: M. Odersky (Ed.), ECOOP 2004 – Object-Oriented Programming. XIII, 611 pages. 2004.

Vol. 3085: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs. X, 409 pages. 2004.

Vol. 3084: A. Persson, J. Stirna (Eds.), Advanced Information Systems Engineering. XIV, 596 pages. 2004.

Vol. 3083: W. Emmerich, A.L. Wolf (Eds.), Component Deployment. X, 249 pages. 2004.

Vol. 3080: J. Desel, B. Pernici, M. Weske (Eds.), Business Process Management. X, 307 pages. 2004.

Vol. 3079: Z. Mammeri, P. Lorenz (Eds.), High Speed Networks and Multimedia Communications. XVIII, 1103 pages. 2004.

Vol. 3078: S. Cotin, D.N. Metaxas (Eds.), Medical Simulation. XVI, 296 pages. 2004.

Vol. 3077: F. Roli, J. Kittler, T. Windeatt (Eds.), Multiple Classifier Systems. XII, 386 pages. 2004.

Vol. 3076: D. Buell (Ed.), Algorithmic Number Theory. XI, 451 pages. 2004.

Vol. 3075: W. Lenski, Logic versus Approximation. VIII, 205 pages. 2004.

Vol. 3074: B. Kuijpers, P. Revesz (Eds.), Constraint Databases and Applications. XII, 181 pages. 2004.

Vol. 3073: H. Chen, R. Moore, D.D. Zeng, J. Leavitt (Eds.), Intelligence and Security Informatics. XV, 536 pages. 2004.

Vol. 3072: D. Zhang, A.K. Jain (Eds.), Biometric Authentication. XVII, 800 pages. 2004.

Vol. 3071: A. Omicini, P. Petta, J. Pitt (Eds.), Engineering Societies in the Agents World. XIII, 409 pages. 2004. (Subseries LNAI).

Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004. XXV, 1208 pages. 2004. (Subseries LNAI).

Vol. 3068: E. André, L. Dybkjær, W. Minker, P. Heisterkamp (Eds.), Affective Dialogue Systems. XII, 324 pages. 2004. (Subseries LNAI).

Vol. 3067: M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), Programming Multi-Agent Systems. X, 221 pages. 2004. (Subseries LNAI).

Vol. 3066: S. Tsumoto, R. Słowiński, J. Komorowski, J.W. Grzymała-Busse (Eds.), Rough Sets and Current Trends in Computing. XX, 853 pages. 2004. (Subseries LNAI).

Vol. 3065: A. Lomuscio, D. Nute (Eds.), Deontic Logic in Computer Science. X, 275 pages. 2004. (Subseries LNAI).

Vol. 3064: D. Bienstock, G. Nemhauser (Eds.), Integer Programming and Combinatorial Optimization. XI, 445 pages. 2004.

Vol. 3063: A. Llamosí, A. Strohmeier (Eds.), Reliable Software Technologies - Ada-Europe 2004. XIII, 333 pages. 2004.

Vol. 3062: J.L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance. XV, 500 pages. 2004.

Vol. 3061: F.F. Ramos, H. Unger, V. Larios (Eds.), Advanced Distributed Systems. VIII, 285 pages. 2004.

Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), Advances in Artificial Intelligence. XIII, 582 pages. 2004. (Subseries LNAI).

Vol. 3059: C.C. Ribeiro, S.L. Martins (Eds.), Experimental and Efficient Algorithms. X, 586 pages. 2004.

Vol. 3058: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 233 pages. 2004.

Vol. 3057: B. Jayaraman (Ed.), Practical Aspects of Declarative Languages. VIII, 255 pages. 2004.

Table of Contents

Invited Talks

Program and System Verification

Security and Safety

Pointer Analysis

Abstract Interpretation and Algorithms

Shape Analysis

Abstract Domain and Data Structures

此为试读,需要完整PDF请访问: www.ertongbook.com

Shape Analysis and Logic

Termination Analysis

Injecting Life with Computers

Ehud Shapiro

Department of Computer Science and Applied Mathematics and
Department of Biological Chemistry
Weizmann Institute of Science, Rehovot 76100, Israel

Abstract. Although electronic computers are the only "computer species" we are accustomed to, the mathematical notion of a programmable computer has nothing to do with wires and logic gates. In fact, Alan Turing's notional computer, which marked in 1936 the birth of modern computer science and still stands at its heart, has greater similarity to natural biomolecular machines such as the ribosome and polymerases than to electronic computers. Recently, a new "computer species" made of biological molecules has emerged. These simple molecular computers inspired by the Turing machine, of which a trillion can fit into a microliter, do not compete with electronic computers in solving complex computational problems; their potential lies elsewhere. Their molecular scale and their ability to interact directly with the biochemical environment in which they operate suggest that in the future they may be the basis of a new kind of "smart drugs": molecular devices equipped with the medical knowledge to perform disease diagnosis and therapy inside the living body. They would detect and diagnose molecular disease symptoms and, when necessary, administer the requisite drug molecules to the cell, tissue or organ in which they operate. In the talk we review this new research direction and report on preliminary steps carried out in our lab towards realizing its vision.

References

1. Benenson Y., Paz-Elitzur T., Adar R., Keinan E, Livneh Z. and Shapiro E. Programmable computing machine made of biomolecules. *Nature*, 414, 430-434, 2001.
2. Benenson Y., Adar R., Paz-Elitzur T., Livneh Z., and Shapiro E. DNA molecule provides a computing machine with both data and fuel. *PNAS*, 100, 2191-2196, 2003.
3. Adar R., Benenson Y., Linshiz G., Rozner A., Tishby N. and Shapiro E. Stochastic computing with biomolecular automata. *PNAS*, in press, 2004.
4. Benenson Y., Gil B., Ben-Dor U., Adar R., and Shapiro E. An autonomous molecular computer for logical control of gene expression *Nature*, 429, 423-429, 2004. Verlag, 2002.

R. Giacobazzi (Ed.): SAS 2004, LNCS 3148, p. 1, 2004.
© Springer-Verlag Berlin Heidelberg 2004

The BLAST Query Language for Software Verification*

Dirk Beyer[1], Adam J. Chlipala[2], Thomas A. Henzinger[1,2], Ranjit Jhala[2], and Rupak Majumdar[3]

[1] EPFL, Switzerland
[2] University of California, Berkeley
[3] University of California, Los Angeles

Abstract. BLAST is an automatic verification tool for checking temporal safety properties of C programs. BLAST is based on lazy predicate abstraction driven by interpolation-based predicate discovery. In this paper, we present the BLAST specification language. The language specifies program properties at two levels of precision. At the lower level, monitor automata are used to specify temporal safety properties of program executions (traces). At the higher level, relational reachability queries over program locations are used to combine lower-level trace properties. The two-level specification language can be used to break down a verification task into several independent calls of the model-checking engine. In this way, each call to the model checker may have to analyze only part of the program, or part of the specification, and may thus succeed in a reduction of the number of predicates needed for the analysis. In addition, the two-level specification language provides a means for structuring and maintaining specifications.

1 Introduction

BLAST, the Berkeley Lazy Abstraction Software verification Tool, is a fully automatic engine for software model checking [11]. BLAST uses counterexample-guided predicate abstraction refinement to verify temporal safety properties of C programs. The tool incrementally constructs an abstract reachability tree (ART) whose nodes are labeled with program locations and truth values of predicates. If a path that violates the desired safety property is found in the ART, but is not a feasible path of the program, then new predicate information is added to the ART in order to rule out the spurious error path. The new predicate information is added on-demand and locally, following the twin paradigms of *lazy abstraction* [11] and *interpolation-based predicate discovery* [8]. The procedure stops when either a genuine error path is found, or the current ART represents a proof of program correctness [9].

In this paper we present the BLAST input language for specifying program-verification tasks. The BLAST specification language consists of two levels. On

* This research was supported in part by the NSF grants CCR-0085949, CCR-0234690, and ITR-0326577.

R. Giacobazzi (Ed.): SAS 2004, LNCS 3148, pp. 2–18, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the lower level, *observer automata* are defined to monitor the program execution and decide whether a safety property is violated. Observer automata can be infinite-state and can track the program state, including the values of program variables and type-state information associated with individual data objects. On the higher level, *relational queries* over program locations are defined which may specify both structural program properties (e.g., the existence of a syntactic path between two locations) and semantic program properties (e.g., the existence of a feasible path between two locations). The evaluation of a semantic property invokes the BLAST model-checking engine. A semantic property may also refer to an observer automaton, thus combining the two levels of specification.

Consider the following example. If we change the definition of a variable in a program, we have to review all subsequent read accesses to that variable. Using static analysis we can find all statements that use the variable, but the resulting set is often imprecise (e.g., it may include dead code) because of the path-insensitive nature of the analysis. Model checking can avoid this imprecision. In addition, using an observer automaton, we can ensure that we compute only those statements subsequent to the variable definition which (1) use the variable and (2) are not preceded by a redefinition of the variable. The two specification levels allow the natural expression of such a query: on the higher level, we specify the location-based reachability property between definition and use locations, and at the lower level, we specify the desired temporal property by a monitor automaton that watches out for redefinitions of the variable. The resulting query asks the model checker for the set of definition-use pairs of program locations that are connected by feasible paths along which no redefinitions occur.

The BLAST specification language provides a convenient user interface: it keeps specifications separate from the program code and makes the model checker easier to use for non-experts, as no manual program annotations with specification code (such as assertions) are required. On one hand it is useful to orthogonalize concerns by separating program properties from the source code, and keeping them separated during development, in order to make it easier to understand and maintain both the program and the specification [13]. On the other hand it is preferable for the programmer to specify program properties in a language that is similar to the programming language. We therefore use as much as possible C-like syntax in the specification language. The states of observer automata are defined using C type and variable declarations, and the automaton transitions are defined using C code. The query language is an imperative scripting language whose expressions specify first-order relational constraints on program locations.

The two-level specification structure provides two further benefits. First, such structured specifications are easy to read, compose, and revise. The relational query language allows the programmer to treat the program as a database of facts, which can be queried by the analysis engine. Moreover, individual parts of a composite query can be checked incrementally when the program changes, as in regression testing [10]. Second, the high-level query language can be used to break down a verification task into several independent model-checking prob-

lems, each checking a low-level trace property. Since the number of predicates in the ART is the main source of complexity for the model-checking procedure, the decomposition of a verification task into several independent subtasks, each involving only a part of the program and/or a part of the specification, can greatly contribute to the scalability of the verification process [14, 17]. A simple instance of this occurs if a specification consists of a conjunction of several properties that can be model checked independently. The relational query engine allows the compact definition of such proof-decomposition strategies.

For a more instructive example, suppose that we wish to check that there is no feasible path from a program location ℓ_0 to a program location ℓ_2, and that all syntactic paths from ℓ_0 to ℓ_2 go through location ℓ_1. Then we may decompose the verification task by guessing an intermediate predicate p_1 and checking, independently, the following two simpler properties: (1) there is no feasible path from ℓ_0 to ℓ_1 such that p_1 is false at the end of the path (at ℓ_1), and (2) there is no feasible path from ℓ_1 to ℓ_2 such that p_1 is true at the beginning of the path (at ℓ_1). Both proof obligations (1) and (2) may be much simpler to model check, with fewer predicates needed, than the original verification task. Moreover, each of the two proof obligations can be specified as a reachability query over locations together with an observer automaton that specifies the final (resp. initial) condition p_1.

The paper is organized as follows. In Section 2, we define the (lower-level) language for specifying trace properties through observer automata. In Section 3, we define the (higher-level) language for specifying location properties through relational queries. In Section 4, we give several sample specifications, and in Section 5, we briefly describe how the query processing is implemented in BLAST.

Related Work. Automata are often used to specify temporal safety properties, because they provide a convenient, succinct notation and are often easier to understand than formulas of temporal logic. For example, SLIC [2] specifications are used in the SLAM project [1] to generate C code for model checking. However, SLIC does not support type-state properties and is limited to the specification of interfaces, because it monitors only function calls and returns. Metal [7] and MOPS [4] allow more general pattern languages, but the (finite) state of the automaton must be explicitly enumerated. Temporal-logic specifications, often enriched with syntactic sugar ("patterns"), are used in Bandera [5] and Feaver [12]. Type-state verification [16] is an important concept for ensuring the reliability of software, but the generally used assumption in this field is to consider all paths of a program as feasible. Relational algebra has been applied to analyze the structure of large programs [3] and in dynamic analysis [6]. Also the decomposition of verification tasks has been recognized as a key issue and strategy-definition languages have been proposed [14, 17]. However, the use of a relational query language to group queries and decompose proof obligations in a model-checking environment seems novel.

2 Trace Properties: Observer Automata

Trace properties are expressed using *observer automata*. These provide a way to specify temporal safety properties of C programs based on syntactic pattern matching of C code. An observer automaton consists of a collection of syntactic patterns that, when matched against the current execution point of the observed program, trigger transitions in the observer. Rather than being limited to a finite number of states, the observer may have global variables of any C type, and it may track type-state information associated with the program variables. The observer transitions are also specified in C syntax; they may read program variables and both read and write observer variables.

2.1 Syntax

The definition of an observer automaton consists of a set of declarations, each defining an observer variable, a type state, an initial condition, a final condition, or an event. Figure 1 gives the grammar for specifying observer automata.

```
Observer:     DeclSeq
DeclSeq:      Declaration | DeclSeq Declaration
Declaration:  'GLOBAL' CVarDef
            | 'SHADOW' CTypeName '{' CFieldSeq '}'
            | 'INITIAL' '{' CExpression '}'
            | 'FINAL' '{' CExpression '}'
            | 'EVENT' '{'
                Temporal
                'PATTERN' '{' Pattern '}'
                Assertion
                Action
              '}'
Temporal:     'BEFORE' | 'AFTER' | empty
Pattern:      ParamCStmt | ParamCStmt 'AT' LocDesc
Assertion:    'ASSERT' '{' CExpression '}' | empty
Action:       'ACTION' '{' CStatementSeq '}' | empty
```

Fig. 1. The grammar for the observer specification language.

Observer Variables. The control state of an observer automaton consists of a global part and a per-object part. The global part of the observer state is determined by a set of typed, global observer variables. Each observer variable may have any C type, and is declared following the keyword `GLOBAL`, where the nonterminal `CVarDef` stands for any C variable declaration. For example, in the case of a specification that restricts the number of calls to a certain function, an observer variable `numCalls` of type `int` might be used to track the number of calls made: "`GLOBAL int numCalls;`".

此为试读,需要完整PDF请访问: www.ertongbook.com