

Patrice Godefroid (Ed.)

LNCS 3639

Model Checking Software

12th International SPIN Workshop
San Francisco, CA, USA, August 2005
Proceedings



TP 311.5-53
8757
2005

Patrice Godefroid (Ed.)

Model Checking Software

12th International SPIN Workshop
San Francisco, CA, USA, August 22-24, 2005
Proceedings



E200502110



Springer

Volume Editor

Patrice Godefroid
Bell Laboratories, Lucent Technologies
2701 Lucent Lane, Lisle, IL 60532, USA
E-mail: god@bell-labs.com

Library of Congress Control Number: 2005930636

CR Subject Classification (1998): F.3, D.2.4, D.3.1, D.2

ISSN 0302-9743
ISBN-10 3-540-28195-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28195-5 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 2005

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11537328 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 12th International SPIN Workshop on Model Checking of Software, held in San Francisco, USA, on August 22–24, 2005. SPIN 2005 is a forum for practitioners and researchers interested in model-checking based techniques for the validation and analysis of communication protocols and software systems. The workshop focuses on topics including theoretical and algorithmic foundations and tools for software model checking, model derivation from code and code derivation from models, techniques for dealing with large and infinite state spaces, and applications. The workshop aims to foster interactions and exchanges of ideas with all related areas in software engineering. It has traditionally drawn contributions from both academia and industry.

The SPIN workshop series started 10 years ago, in 1995. Since then, SPIN workshops have been held on an annual basis at Montréal (1995), New Brunswick (1996), Enschede (1997), Paris (1998), Trento (1999), Toulouse (1999), Stanford (2000), Toronto (2001), Grenoble (2002), Portland (2003) and Barcelona (2004). All but the first SPIN workshop were organized as satellite events of larger conferences, in particular of CAV (1996), TACAS (1997), FORTE/PSTV (1998), FLOC (1999), the World Congress on Formal Methods (1999), FMOODS (2000), ICSE (2001, 2003) and ETAPS (2002, 2004). This year, SPIN was held as a satellite event of CONCUR 2005. The co-location of SPIN workshops with conferences has proven to be very successful and has helped to disseminate SPIN model checking technology to wider audiences. Since 1999, the proceedings of the SPIN workshops have appeared in Springer's Lecture Notes in Computer Science series.

The history of successful SPIN workshops is evidence for the maturing of software model-checking technology. While in earlier years the focus of the workshop series was algorithms and tool development around the SPIN model-checker, its scope was widened several years ago to include other software model-checking techniques, tools and applications.

This year, we received 45 regular paper submissions out of which 15 were selected. In addition, the SPIN 2005 program contained 4 tool presentation papers selected from 6 submissions. All submissions went through a rigorous reviewing process, where each paper received a minimum of 3 referee reviews.

In addition to the refereed papers, three invited talks were given, by David Wagner (UC Berkeley) on *Pushdown Model Checking for Security*, Dawson Engler (Stanford University) on *Static Analysis Versus Model Checking for Bug Finding*, and Rajeev Alur (University of Pennsylvania) on *The Benefits of Exposing Calls and Returns* (invited talk of CONCUR/2005, and joint with SPIN/2005). Dawson Engler also contributed an original paper entitled *Execution Generated Test Cases: How to Make Systems Code Crash Itself* (co-authored

with Cristian Cadar), which can be found in these proceedings. The program also included three invited tutorials, by Gerard J. Holzmann (NASA JPL) and Theo C. Ruys (University of Twente) on *Effective Bug Hunting with SPIN and MODEX*, by Thomas A. Henzinger (EPFL), Ranjit Jhala (UCSD) and Rupak Majumdar (UCLA) on *The BLAST Software Verification System*, and by Willem Visser (NASA Ames) on *Model Checking Programs with Java PathFinder*.

I would like to thank the Program Committee members, as well as all the external reviewers who assisted them in their work. My thanks also go to the Steering Committee members and last year's organizers, Susanne Graf and Laurent Mounier, for their helpful advice. I would also like to thank the invited speakers and invited tutorial speakers, the authors of submitted papers, and the workshop participants. Special thanks go to Springer for providing us with the possibility of using a Conference Online Service free of charge and to the METAFramework team for their support. Last but not least, I would like to thank the organizers of CONCUR 2005, in particular Luca de Alfaro, for inviting us to hold SPIN 2005 as a satellite event and for their support and flexibility in accommodating the particular needs of the SPIN workshop.

June 2005

Patrice Godefroid

Organization

Program Committee

George Avrunin (U. Mass. Amherst, USA)
Dennis Dams (Bell Labs, USA)
Stefan Edelkamp (U. Dortmund, Germany)
Cormac Flanagan (UC Santa Cruz, USA)
Jaco Geldenhuys (Tampere U., Finland)
Patrice Godefroid (Bell Labs, USA; Chair)
Susanne Graf (Verimag, France)
Gerard Holzmann (NASA JPL, USA)
Sarfraz Khurshid (UT Austin, USA)
Stefan Leue (U. Konstanz, Germany)
Rupak Majumdar (UCLA, USA)
Laurent Mounier (Verimag, France)
Shaz Qadeer (Microsoft, USA)
Theo Ruys (U. Twente, Netherlands)
Willem Visser (NASA Ames, USA)
Pierre Wolper (U. Liège, Belgium)

Advisory Committee

Gerard Holzmann (NASA JPL, USA; Chair)
Amir Pnueli (Weizmann Inst., Israel)

Steering Committee

Thomas Ball (Microsoft, USA)
Susanne Graf (Verimag, France)
Stefan Leue (U. Konstanz, Germany)
Moshe Vardi (Rice U., USA)
Pierre Wolper (U. Liège, Belgium; Chair)

Additional Reviewers

Husain Aljazzar	Pieter de Villiers	Ranjit Jhala
Nina Amla	Paul Grisham	Rajeev Joshi
Dragan Boshnachki	Henri Hansen	Alberto Lluch-Lafuente
Marius Bozga	Klaus Havelund	Oded Maler
Richard Chang	Shahid Jabbar	Tilman Mehler

VIII Organization

Kedar Namjoshi
Iulian Ober
Corina Pasareanu
Michael Perin

Nicolas Rouquette
Hassen Saïdi
Danhua Shao
Natalia Sidorova

Evghenia Stegantova
Brink Van der Merwe
Wei Wei

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

Lecture Notes in Computer Science

For information about Vols. 1–3518

please contact your bookseller or Springer

- Vol. 3654: S. Jajodia, D. Wijesekera (Eds.), *Data and Applications Security XIX*. X, 353 pages. 2005.
- Vol. 3639: P. Godefroid (Ed.), *Model Checking Software*. XI, 289 pages. 2005.
- Vol. 3638: A. Butz, B. Fisher, A. Krüger, P. Olivier (Eds.), *Smart Graphics*. XI, 269 pages. 2005.
- Vol. 3634: L. Ong (Ed.), *Computer Science Logic*. XI, 567 pages. 2005.
- Vol. 3633: C. Bauzer Medeiros, M. Egenhofer, E. Bertino (Eds.), *Advances in Spatial and Temporal Databases*. XIII, 433 pages. 2005.
- Vol. 3632: R. Nieuwenhuis (Ed.), *Automated Deduction – CADE-20*. XIII, 459 pages. 2005. (Subseries LNAI).
- Vol. 3626: B. Ganter, G. Stumm, R. Wille (Eds.), *Formal Concept Analysis*. X, 349 pages. 2005. (Subseries LNAI).
- Vol. 3623: M. Liśkiewicz, R. Reischuk (Eds.), *Fundamentals of Computation Theory*. XV, 576 pages. 2005.
- Vol. 3621: V. Shoup (Ed.), *Advances in Cryptology – CRYPTO 2005*. XI, 568 pages. 2005.
- Vol. 3619: X. Lu, W. Zhao (Eds.), *Networking and Mobile Computing*. XXIV, 1299 pages. 2005.
- Vol. 3615: B. Ludäscher, L. Raschid (Eds.), *Data Integration in the Life Sciences*. XII, 344 pages. 2005. (Subseries LNBI).
- Vol. 3608: F. Dehne, A. López-Ortiz, J.-R. Sack (Eds.), *Algorithms and Data Structures*. XIV, 446 pages. 2005.
- Vol. 3607: J.-D. Zucker, L. Saitta (Eds.), *Abstraction, Reformulation and Approximation*. XII, 376 pages. 2005. (Subseries LNAI).
- Vol. 3606: V. Malyshev (Ed.), *Parallel Computing Technologies*. XII, 470 pages. 2005.
- Vol. 3602: R. Eigenmann, Z. Li, S.P. Midkiff (Eds.), *Languages and Compilers for High Performance Computing*. IX, 486 pages. 2005.
- Vol. 3598: H. Murakami, H. Nakashima, H. Tokuda, M. Yasumura, *Ubiquitous Computing Systems*. XIII, 275 pages. 2005.
- Vol. 3597: S. Shimojo, S. Ichii, T.W. Ling, K.-H. Song (Eds.), *Web and Communication Technologies and Internet-Related Social Issues – HSI 2005*. XIX, 368 pages. 2005.
- Vol. 3596: F. Dau, M.-L. Mugnier, G. Stumm (Eds.), *Conceptual Structures: Common Semantics for Sharing Knowledge*. XI, 467 pages. 2005. (Subseries LNAI).
- Vol. 3595: L. Wang (Ed.), *Computing and Combinatorics*. XVI, 995 pages. 2005.
- Vol. 3594: J.C. Setubal, S. Verjovski-Almeida (Eds.), *Advances in Bioinformatics and Computational Biology*. XIV, 258 pages. 2005. (Subseries LNBI).
- Vol. 3592: S. Katsikas, J. Lopez, G. Pernul (Eds.), *Trust and Privacy in Digital Business*. XII, 332 pages. 2005.
- Vol. 3587: P. Perner, A. Imiya (Eds.), *Machine Learning and Data Mining in Pattern Recognition*. XVII, 695 pages. 2005. (Subseries LNAI).
- Vol. 3586: A.P. Black (Ed.), *ECOOP 2005 – Object-Oriented Programming*. XVII, 631 pages. 2005.
- Vol. 3584: X. Li, S. Wang, Z.Y. Dong (Eds.), *Advanced Data Mining and Applications*. XIX, 835 pages. 2005. (Subseries LNAI).
- Vol. 3583: R.W. H. Lau, Q. Li, R. Cheung, W. Liu (Eds.), *Advances in Web-Based Learning – ICWL 2005*. XIV, 420 pages. 2005.
- Vol. 3582: J. Fitzgerald, I.J. Hayes, A. Tarlecki (Eds.), *FM 2005: Formal Methods*. XIV, 558 pages. 2005.
- Vol. 3581: S. Miksch, J. Hunter, E. Keravnou (Eds.), *Artificial Intelligence in Medicine*. XVII, 547 pages. 2005. (Subseries LNAI).
- Vol. 3580: L. Caires, G.F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), *Automata, Languages and Programming*. XXV, 1477 pages. 2005.
- Vol. 3579: D. Lowe, M. Gaedke (Eds.), *Web Engineering*. XXII, 633 pages. 2005.
- Vol. 3578: M. Gallagher, J. Hogan, F. Maire (Eds.), *Intelligent Data Engineering and Automated Learning – IDEAL 2005*. XVI, 599 pages. 2005.
- Vol. 3577: R. Falcone, S. Barber, J. Sabater-Mir, M.P. Singh (Eds.), *Trusting Agents for Trusting Electronic Societies*. VIII, 235 pages. 2005. (Subseries LNAI).
- Vol. 3576: K. Etessami, S.K. Rajamani (Eds.), *Computer Aided Verification*. XV, 564 pages. 2005.
- Vol. 3575: S. Wermter, G. Palm, M. Elshaw (Eds.), *Biomimetic Neural Learning for Intelligent Robots*. IX, 383 pages. 2005. (Subseries LNAI).
- Vol. 3574: C. Boyd, J.M. González Nieto (Eds.), *Information Security and Privacy*. XIII, 586 pages. 2005.
- Vol. 3573: S. Etalle (Ed.), *Logic Based Program Synthesis and Transformation*. VIII, 279 pages. 2005.
- Vol. 3572: C. De Felice, A. Restivo (Eds.), *Developments in Language Theory*. XI, 409 pages. 2005.
- Vol. 3571: L. Godo (Ed.), *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. XVI, 1028 pages. 2005. (Subseries LNAI).
- Vol. 3570: A. S. Patrick, M. Yung (Eds.), *Financial Cryptography and Data Security*. XII, 376 pages. 2005.
- Vol. 3569: F. Bacchus, T. Walsh (Eds.), *Theory and Applications of Satisfiability Testing*. XII, 492 pages. 2005.

- Vol. 3568: W.-K. Leow, M.S. Lew, T.-S. Chua, W.-Y. Ma, L. Chaisorn, E.M. Bakker (Eds.), *Image and Video Retrieval*. XVII, 672 pages. 2005.
- Vol. 3567: M. Jackson, D. Nelson, S. Stirk (Eds.), *Database: Enterprise, Skills and Innovation*. XII, 185 pages. 2005.
- Vol. 3566: J.-P. Banâtre, P. Fradet, J.-L. Giavitto, O. Michel (Eds.), *Unconventional Programming Paradigms*. XI, 367 pages. 2005.
- Vol. 3565: G.E. Christensen, M. Sonka (Eds.), *Information Processing in Medical Imaging*. XXI, 777 pages. 2005.
- Vol. 3564: N. Eisinger, J. Małuszynski (Eds.), *Reasoning Web*. IX, 319 pages. 2005.
- Vol. 3562: J. Mira, J.R. Álvarez (Eds.), *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Part II. XXIV, 636 pages. 2005.
- Vol. 3561: J. Mira, J.R. Álvarez (Eds.), *Mechanisms, Symbols, and Models Underlying Cognition*, Part I. XXIV, 532 pages. 2005.
- Vol. 3560: V.K. Prasanna, S. Iyengar, P.G. Spirakis, M. Welsh (Eds.), *Distributed Computing in Sensor Systems*. XV, 423 pages. 2005.
- Vol. 3559: P. Auer, R. Meir (Eds.), *Learning Theory*. XI, 692 pages. 2005. (Subseries LNAI).
- Vol. 3558: V. Torra, Y. Narukawa, S. Miyamoto (Eds.), *Modeling Decisions for Artificial Intelligence*. XII, 470 pages. 2005. (Subseries LNAI).
- Vol. 3557: H. Gilbert, H. Handschuh (Eds.), *Fast Software Encryption*. XI, 443 pages. 2005.
- Vol. 3556: H. Baumeister, M. Marchesi, M. Holcombe (Eds.), *Extreme Programming and Agile Processes in Software Engineering*. XIV, 332 pages. 2005.
- Vol. 3555: T. Vardanega, A.J. Wellings (Eds.), *Reliable Software Technology – Ada-Europe 2005*. XV, 273 pages. 2005.
- Vol. 3554: A. Dey, B. Kokinov, D. Leake, R. Turner (Eds.), *Modeling and Using Context*. XIV, 572 pages. 2005. (Subseries LNAI).
- Vol. 3553: T.D. Hämäläinen, A.D. Pimentel, J. Takala, S. Vassiliadis (Eds.), *Embedded Computer Systems: Architectures, Modeling, and Simulation*. XV, 476 pages. 2005.
- Vol. 3552: H. de Meer, N. Bhatti (Eds.), *Quality of Service – IWQoS 2005*. XVIII, 400 pages. 2005.
- Vol. 3551: T. Härder, W. Lehner (Eds.), *Data Management in a Connected World*. XIX, 371 pages. 2005.
- Vol. 3548: K. Julisch, C. Kruegel (Eds.), *Intrusion and Malware Detection and Vulnerability Assessment*. X, 241 pages. 2005.
- Vol. 3547: F. Bomarius, S. Komi-Sirviö (Eds.), *Product Focused Software Process Improvement*. XIII, 588 pages. 2005.
- Vol. 3546: T. Kanade, A. Jain, N.K. Ratha (Eds.), *Audio- and Video-Based Biometric Person Authentication*. XX, 1134 pages. 2005.
- Vol. 3544: T. Higashino (Ed.), *Principles of Distributed Systems*. XII, 460 pages. 2005.
- Vol. 3543: L. Kutvonen, N. Alonistioti (Eds.), *Distributed Applications and Interoperable Systems*. XI, 235 pages. 2005.
- Vol. 3542: H.H. Hoos, D.G. Mitchell (Eds.), *Theory and Applications of Satisfiability Testing*. XIII, 393 pages. 2005.
- Vol. 3541: N.C. Oza, R. Polikar, J. Kittler, F. Roli (Eds.), *Multiple Classifier Systems*. XII, 430 pages. 2005.
- Vol. 3540: H. Kalviainen, J. Parkkinen, A. Kaarna (Eds.), *Image Analysis*. XXII, 1270 pages. 2005.
- Vol. 3539: K. Morik, J.-F. Boulicaut, A. Siebes (Eds.), *Local Pattern Detection*. XI, 233 pages. 2005. (Subseries LNAI).
- Vol. 3538: L. Ardissono, P. Brna, A. Mitrovic (Eds.), *User Modeling 2005*. XVI, 533 pages. 2005. (Subseries LNAI).
- Vol. 3537: A. Apostolico, M. Crochemore, K. Park (Eds.), *Combinatorial Pattern Matching*. XI, 444 pages. 2005.
- Vol. 3536: G. Ciardo, P. Darondeau (Eds.), *Applications and Theory of Petri Nets 2005*. XI, 470 pages. 2005.
- Vol. 3535: M. Steffen, G. Zavattaro (Eds.), *Formal Methods for Open Object-Based Distributed Systems*. X, 323 pages. 2005.
- Vol. 3534: S. Spaccapietra, E. Zimányi (Eds.), *Journal on Data Semantics III*. XI, 213 pages. 2005.
- Vol. 3533: M. Ali, F. Esposito (Eds.), *Innovations in Applied Artificial Intelligence*. XX, 858 pages. 2005. (Subseries LNAI).
- Vol. 3532: A. Gómez-Pérez, J. Euzenat (Eds.), *The Semantic Web: Research and Applications*. XV, 728 pages. 2005.
- Vol. 3531: J. Ioannidis, A. Keromytis, M. Yung (Eds.), *Applied Cryptography and Network Security*. XI, 530 pages. 2005.
- Vol. 3530: A. Prinz, R. Reed, J. Reed (Eds.), *SDL 2005: Model Driven*. XI, 361 pages. 2005.
- Vol. 3528: P.S. Szczepaniak, J. Kacprzyk, A. Niewiadomski (Eds.), *Advances in Web Intelligence*. XVII, 513 pages. 2005. (Subseries LNAI).
- Vol. 3527: R. Morrison, F. Oquendo (Eds.), *Software Architecture*. XII, 263 pages. 2005.
- Vol. 3526: S. B. Cooper, B. Löwe, L. Torenvliet (Eds.), *New Computational Paradigms*. XVII, 574 pages. 2005.
- Vol. 3525: A.E. Abdallah, C.B. Jones, J.W. Sanders (Eds.), *Communicating Sequential Processes*. XIV, 321 pages. 2005.
- Vol. 3524: R. Barták, M. Milano (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. XI, 320 pages. 2005.
- Vol. 3523: J.S. Marques, N. Pérez de la Blanca, P. Pina (Eds.), *Pattern Recognition and Image Analysis*, Part II. XXVI, 733 pages. 2005.
- Vol. 3522: J.S. Marques, N. Pérez de la Blanca, P. Pina (Eds.), *Pattern Recognition and Image Analysis*, Part I. XXVI, 703 pages. 2005.
- Vol. 3521: N. Megiddo, Y. Xu, B. Zhu (Eds.), *Algorithmic Applications in Management*. XIII, 484 pages. 2005.
- Vol. 3520: O. Pastor, J. Falcão e Cunha (Eds.), *Advanced Information Systems Engineering*. XVI, 584 pages. 2005.
- Vol. 3519: H. Li, P. J. Olver, G. Sommer (Eds.), *Computer Algebra and Geometric Algebra with Applications*. IX, 449 pages. 2005.

Table of Contents

Invited Talks/Papers

Pushdown Model Checking for Security <i>David Wagner</i>	1
Execution Generated Test Cases: How to Make Systems Code Crash Itself <i>Cristian Cadar, Dawson Engler</i>	2

Invited Tutorials

Effective Bug Hunting with Spin and Modex <i>Gerard J. Holzmann, Theo C. Ruys</i>	24
The BLAST Software Verification System <i>Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar</i>	25
Model Checking Programs with Java PathFinder <i>Willem Visser, Peter Mehlitz</i>	27

State Representation and Abstraction

An Incremental Heap Canonicalization Algorithm <i>Madanlal Musuvathi, David L. Dill</i>	28
Memory Efficient State Space Storage in Explicit Software Model Checking <i>Sami Evangelista, Jean-François Pradat-Peyre</i>	43
Counterexample-Based Refinement for a Boundedness Test for CFSM Languages <i>Stefan Leue, Wei Wei</i>	58

Dealing with Concurrency

Symbolic Model Checking for Asynchronous Boolean Programs <i>Byron Cook, Daniel Kroening, Natasha Sharygina</i>	75
--	----

Improving Spin's Partial-Order Reduction for Breadth-First Search <i>Dragan Bošnački, Gerard J. Holzmann</i>	91
Sound Transaction-Based Reduction Without Cycle Detection <i>Vladimir Levin, Robert Palmer, Shaz Qadeer, Sriram K. Rajamani</i>	106
<hr/> Dealing with Complex Data <hr/>	
Repairing Structurally Complex Data <i>Sarfraz Khurshid, Iván García, Yuk Lai Suen</i>	123
Crafting a Promela Front-End with Abstract Data Types to Mitigate the Sensitivity of (Compositional) Analysis to Implementation Choices <i>Yung-Pin Cheng</i>	139
Behavioural Models for Hierarchical Components <i>Tomás Barros, Ludovic Henrio, Eric Madelaine</i>	154
<hr/> Checking Temporal Properties <hr/>	
On-the-Fly Emptiness Checks for Generalized Büchi Automata <i>Jean-Michel Couvreur, Alexandre Duret-Lutz, Denis Poitrenaud</i>	169
Stuttering Congruence for χ <i>Bas Luttik, Nikola Trčka</i>	185
Verifying Pattern-Generated LTL Formulas: A Case Study <i>Salamah Salamah, Ann Gates, Steve Roach, Oscar Mondragon</i>	200
<hr/> Checking Security and Real-Time Properties <hr/>	
Generic Verification of Security Protocols <i>Abdul Sahid Khan, Madhavan Mukund, S.P. Suresh</i>	221
Using SPIN and Eclipse for Optimized High-Level Modeling and Analysis of Computer Network Attack Models <i>Gerrit Rothmaier, Tobias Kneiphoff, Heiko Krumm</i>	236
Model Checking Machine Code with the GNU Debugger <i>Eric Mercer, Michael Jones</i>	251

Tool Papers

ETCH: An Enhanced Type Checking Tool for Promela <i>Alastair F. Donaldson, Simon J. Gay</i>	266
Enhanced Probabilistic Verification with 3Spin and 3Murphi <i>Peter C. Dillinger, Panagiotis Manolios</i>	272
SPLAT: A Tool for Model-Checking and Dynamically-Enforcing Abstractions <i>Anil Madhavapeddy, David Scott, Richard Sharp</i>	277
Learning-Based Assume-Guarantee Verification (Tool Paper) <i>Dimitra Giannakopoulou, Corina S. Păsăreanu</i>	282
Author Index	289

Pushdown Model Checking for Security

David Wagner

U.C. Berkeley

daw@cs.berkeley.edu

Abstract. One of the key challenges for computer security is the problem of software security: how to build software that is free of implementation vulnerabilities. In this talk, I will present experience with pushdown model checking for software security. First, I will survey simple methods for pushdown model checking, and I will introduce MOPS, a tool for pushdown model checking of C programs. Then, I will show many security properties of interest may be encoded as temporal safety properties that are well-suited to analysis with model checking. I will report on our experience applying MOPS to tens of millions of lines of C code. Finally, I will discuss some possible directions for future research.

Execution Generated Test Cases: How to Make Systems Code Crash Itself

Cristian Cadar and Dawson Engler*

Computer Systems Laboratory,
Stanford University,
Stanford, CA 94305, U.S.A

Abstract. This paper presents a technique that uses code to automatically generate its own test cases at run-time by using a combination of symbolic and concrete (i.e., regular) execution. The input values to a program (or software component) provide the standard interface of any testing framework with the program it is testing, and generating input values that will explore all the “interesting” behavior in the tested program remains an important open problem in software testing research. Our approach works by turning the problem on its head: we lazily generate, from within the program itself, the input values to the program (and values derived from input values) as needed. We applied the technique to real code and found numerous corner-case errors ranging from simple memory overflows and infinite loops to subtle issues in the interpretation of language standards.

1 Introduction

Systems code is difficult to test comprehensively. Externally, systems interfaces tend towards the baroque, with many different possible behaviors based on tricky combinations of inputs. Internally, their implementations tend towards heavily entangling nests of conditionals that are difficult to enumerate, much less exhaust with test cases. Both features conspire to make comprehensive, manual testing an enormous undertaking, so enormous that empirically, many systems code test suites consist only of a handful of simple cases or, perhaps even more commonly, none at all.

Random testing can augment manual testing to some degree. A good example is the fuzz [3, 4] tool, which automatically generates random inputs, which is enough to find errors in many applications. Random testing has the charm that it requires no manual work, other than interfacing the generator to the tested code. However, random test generation by itself has several severe drawbacks. First, blind generation of values means that it misses errors triggered by narrow ranges of inputs. A trivial example: if a function only has an error if its 32-bit integer argument is equal to “12345678” then random will most likely have to generate billions of test cases before it hits this specific case. Second, and

* This paper is a shortened version of [1], which was in simultaneous submission with similar but independent work by Patrice Godefroid et al [2]. Our thanks to Patrice for graciously accepting this version as an invited paper.

similarly, random testing has difficulty hitting errors that depend on several different inputs being within specific (even wide) ranges of values. Third, the ability of random testing to effectively generate random noise is also its curse. It is very poor at generating input that has structure, and as a result will miss errors that require some amount of correct structure in input before they can be hit. A clear example would be using random test generation to find bugs in a language parser. It will find cases where the parser cannot handle garbage inputs. However, because of the extreme improbability of random generation constructing inputs that look anything like legal programs it will miss almost all errors cases where the parser mishandles them.

Of course, random can be augmented with some amount of guidance to more intelligently generate inputs, though this comes at the cost of manual intervention. A typical example would be writing a tool to take a manually-written language grammar and use it to randomly generate legal and illegal programs that are fed to the tested program. Another would be having a specification or model of what a function's external behavior is and generate test cases using this model to try to hit "interesting" combinations. However, all such hybrid approaches require manual labor and, more importantly, a willingness of implementors to provide this labor at all. The reluctance of systems builders to write specifications, grammars, models of what their code does, or even assertions is well known. As a result, very few real systems have used such approaches.

This paper's first contribution is the observation that code can be used to *automatically* generate its own potentially highly complex test cases. At a high level, the basic idea is simple. Rather than running the code on manually-constructed concrete input, we instead run it on symbolic input that is initially allowed to be "anything." As the code observes this input, these observations tell us what legal values (or ranges of values) the input could be. Each time the code makes a decision based on an observation we conceptually fork the execution, adding on one branch the constraint that the input satisfies the observation, and on the other that it does not. We can then generate test cases by solving these constraints for concrete values. We call such tests *execution generated testing* (EGT).

This process is most easily seen by example. Consider the following contrived routine `bad_abs` that incorrectly implements absolute value:

```
0:  int bad_abs(int x) {  
1:      if(x < 0)  
2:          return -x;  
3:      if(x == 12345678)  
4:          return -x;  
5:      return x;  
6:  }
```

As mentioned before, even such a simple error will probably take billions of random-generated test cases to hit. In contrast, finding it with execution generated testing it is straightforward. Symbolic execution would proceed as follows:

1. Initial state: set x to the symbolic value of “anything.” In this case, before any observations at all, it can be any value between INT_MIN and INT_MAX . Thus we have the constraints $x \geq INT_MIN \wedge x \leq INT_MAX$.
2. Begin running the code.
3. At the first conditional (line 1) fork the execution, setting x to the symbolic constraint $x < 0$ on the true path, and to $x \geq 0$ on the false path.
4. At the return (line 2) solve the constraints on x for a concrete value (such as $x == -1$). This value is later used as a test input to `bad_abs`.
5. At the second conditional (line 3) fork the execution, setting x to the constraints $x \equiv 12345678 \wedge x \geq 0$ on the true path and $x \neq 12345678 \wedge x \geq 0$ on the false path.
6. At the second return (line 4) solve the symbolic constraints $x \equiv 12345678 \wedge x \geq 0$. The value 12345678 is our second test case.
7. Finally, at line 5, solve x ’s constraints for a concrete value (e.g., $x = 1$). This value is used as our third, final case.

We can then test the code on the three generated values for x . Of course, this sketch leaves many open questions — when to generate concrete values, how to handle system calls, how to tell what is correct, etc. The rest of the paper discusses these issues in more detail.

There are a couple of ways to look at the approach. From one point of view, implementation code has a “grammar” of the legal inputs it accepts and acts on, or rejects. EGT is an automatic method to extract this grammar (and the concrete sentences it accepts and rejects) from the implementation rather than from a hand-written specification. From another viewpoint, it can be seen as a way to turn code “inside out” so that instead of consuming inputs becomes a generator of them. Finally, and perhaps only half-vacuously, it can be viewed as a crude analogue of the Heisenberg effect in the sense that unlike observations perturbing experiments from a set of potential states into a variety of concrete ones, observations in this case perturb a set of possible inputs into a set of increasingly concrete ones. The more precise the observation the more definitively it perturbs the input. The most precise observation, an equality comparison, fixes the input to a specific concrete value. The least precise, an inequality, simply disallows a single value but leaves all others as possibilities.

This paper has three main contributions:

1. A simple conceptual approach to automatically generate test cases by running code on symbolic inputs.
2. A working prototype EGT system.
3. Experimental results showing that the approach is effective on real code.

The paper is organized as follows. Section 2 gives an overview of the method. Section 3 discusses concrete implementation issues. The next four sections give four case studies of applying the approach to systems code. Finally, Section 7 discusses related work and Section 8 concludes.