Lecture Notes in Computer Science 8931

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

Editorial Board

David Hutchison, UK
Josef Kittler, UK
John C. Mitchell, USA
Bernhard Steffen, Germany
Demetri Terzopoulos, USA
Gerhard Weikum, Germany
Takeo Kanade, USA
Jon M. Kleinberg, USA
Friedemann Mattern, Switzerland
Moni Naor, Israel
C. Pandu Rangan, India
Doug Tygar, USA

Advanced Research in Computing and Software Science
Subline of Lectures Notes in Computer Science

Subline Series Editors

Giorgio Ausiello, *University of Rome 'La Sapienza', Italy*
Vladimiro Sassone, *University of Southampton, UK*

Subline Advisory Board

Susanne Albers, *TU Munich, Germany*
Benjamin C. Pierce, *University of Pennsylvania, USA*
Bernhard Steffen, *University of Dortmund, Germany*
Deng Xiaotie, *City University of Hong Kong*
Jeannette M. Wing, *Microsoft Research, Redmond, WA, USA*

Deepak D'Souza Akash Lal
Kim Guldstrand Larsen (Eds.)

Verification, Model Checking, and Abstract Interpretation

16th International Conference, VMCAI 2015
Mumbai, India, January 12-14, 2015
Proceedings

Springer

Volume Editors

Deepak D'Souza
Indian Institute of Science
Department of Computer Science and Automation
Bangalore, India
E-mail: deepakd@csa.iisc.ernet.in

Akash Lal
Microsoft Research India
Bangalore, India
E-mail: akashl@microsoft.com

Kim Guldstrand Larsen
Aalborg University
Department of Computer Science
Aalborg East, Denmark
E-mail: kgl@cs.aau.dk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-662-46080-1 e-ISBN 978-3-662-46081-8
DOI 10.1007/978-3-662-46081-8
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014958456

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2015
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in ist current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at the 16th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2015), held during January 12–14, 2015, in Mumbai, India.

This edition of the conference attracted 53 competitive submissions from 22 countries across the world. Each submission was reviewed by at least three Program Committee members. The Committee decided to accept 24 papers. The program also included four invited talks.

We would like to thank our invited speakers Supratik Chakraborty, Rustan Leino, Antoine Miné, and Jean-François Raskin for readily agreeing to share their insights with us through their talks and articles contributed to the conference. We would like to thank all the Program Committee members and reviewers for their diligent reviews that helped maintain the high standards of VMCAI. Like many other conferences, we are indebted to EasyChair for providing us with an excellent conference management system. We are grateful to Alfred Hofmann and Anna Kramer of Springer for their close cooperation in publishing these proceedings.

Finally, we would to thank ACM SIGPLAN-SIGACT and the local organizers, especially Sriram Rajamani and Paritosh Pandya, for the excellent infrastructural support to VMCAI.

November 2014

Deepak D'Souza
Akash Lal
Kim Guldstrand Larsen

Organization

Program Committee

Erika Abraham	RWTH Aachen University, Germany
S. Akshay	IIT Bombay, India
Aws Albarghouthi	University of Toronto, Canada
Patricia Bouyer	ENS Cachan, France
Alessandro Cimatti	Fondazione Bruno Kessler, Trento, Italy
Agostino Cortesi	Università Ca' Foscari di Venezia, Italy
Deepak D'Souza	Indian Institute of Science, Bangalore, India
Thomas Dillig	University of Texas, Austin, USA
Constantin Enea	Université Paris, Diderot, France
William Harris	University of Wisconsin-Madison, USA
Ranjit Jhala	University of California, San Diego, USA
Barbara Jobstmann	Verimag, Grenoble, France
Daniel Kroening	University of Oxford, UK
Akash Lal	Microsoft Research, Bangalore, India
Kim Larsen	Aalborg University, Denmark
Francesco Logozzo	Microsoft Research, Redmond, USA
Ken Mcmillan	Cadence Berkeley Labs, USA
David Monniaux	Verimag, Grenoble, France
David Parker	University of Birmingham, UK
Matthew Parkinson	Microsoft Research, Cambridge, UK
Ruzica Piskac	Yale University, USA
Andreas Podelski	University of Freiburg, Germany
Pavithra Prabhakar	IMDEA Software Institute
Xiaokang Qiu	Massachusetts Institute of Technology, USA
Sriram Sankaranarayanan	University of Colorado, Boulder, USA
Thomas Wies	New York University, USA

Additional Reviewers

Bogomolov, Sergiy	D'Silva, Vijay
Bollig, Benedikt	Dang, Thao
Calzavara, Stefano	Delahaye, Benoît
Chakraborty, Supratik	Deshmukh, Jyotirmoy
Chen, Xin	Dodds, Mike
Christ, Juergen	Dragoi, Cezara
Corzilius, Florian	Emmi, Michael
Coughlin, Devin	Ferrara, Pietro

Ganty, Pierre
Gonnord, Laure
Griesmayer, Andreas
Griggio, Alberto
Guha, Shibashis
Hamza, Jad
Harris, William
Hoenicke, Jochen
Irfan, Ahmed
Joshi, Saurabh
Kapinski, James
Kinder, Johannes
Kirchner, Florent
Koskinen, Eric
Kremer, Gereon
Kuncak, Viktor
Maiza, Claire
Mastroeni, Isabella
Micheli, Andrea
Misailovic, Sasa
Mover, Sergio
Murano, Aniello
Niksic, Filip
Norman, Gethin
Parlato, Gennaro
Potet, Marie-Laure
Ramalingam, Ganesan
S., Krishna
Sampath, Prahaladavaradan
Schupp, Stefan
Sharma, Subodh
Song, Fu
Sousa, Marcelo
Srivathsan, B.
Stückrath, Jan
Summers, Alexander J.
Suter, Philippe
Tautschnig, Michael
Tiezzi, Francesco
Trivedi, Ashutosh
Zufferey, Damien

Abstracts

AstréeA: A Static Analyzer for Large Embedded Multi-Task Software

Antoine Miné

CNRS & École Normale Supérieure
45, rue d'Ulm
75005 Paris, France
mine@di.ens.fr

Embedded critical systems, such as planes and cars, cannot be easily fixed during missions and any error can have catastrophic consequences. It is thus primordial to ensure the correctness of their controlling software before they are deployed. At the very least, critical embedded software must be exempt from runtime errors, including ill-defined operations according to the specification of the language (such as arithmetic or memory overflows) as well as failure of programmer-inserted assertions. Sound and approximate static analysis can help, by providing tools able to analyze the large codes found in the industry in a fully automated way and without missing any real error. Sound and scalable static analyzers are sometimes thought to be too imprecise and report too many false alarms to be of any use in the context of verification. This claim was disproved when, a decade ago [2], the Astrée static analyzer [1] successfully analyzed the runtime errors in several Airbus control flight software, with few or no false alarm. This result could be achieved by employing abstract interpretation [4], a principled framework to define and compose modular sound-by-construction and parametric abstractions, but also by adopting a design-by-refinement development strategy. Starting from an efficient and easy to design, but rather coarse, fully flow- and context-sensitive interval analyzer, we integrated more complex abstractions (carefully chosen from the literature, such as octagons [10], adapted from it, such as trace partitioning [9], or specifically invented for our needs, such as digital filter domains [6]) to remove large sets of related false alarms, until we reached our precision target.

In this presentation, we discuss our *on-going* efforts towards a similar goal: the efficient and precise sound verification of the absence of run-time errors, but targeting another, more complex class of software: *shared-memory concurrent embedded C software*. Such software are already present in critical systems and will likely become the norm with the generalization of multi-core processors in embedded systems, leading to new challenging demands in verification. Our analyzer is named AstréeA [5], in reference to Astrée on which it takes its inspiration and on the code base of which it elaborates. AstréeA's specialization target is a family of several embedded avionic codes, each featuring a small fixed set of a dozen threads, more than 1.5 Mlines of C code, implicit communication through the shared memory, and running under a real-time OS based on the ARINC 653 specification.

One major challenge is that a concurrent program execution does not follow a fixed sequential order, but one of many interleavings of executions from different tasks chosen by the scheduler. A sound analysis must consider all possible interleavings in order to cover every corner case and race condition. As it is impractical to build a fully flow-sensitive analysis by enumerating explicitly all interleavings, we took inspiration from *thread-modular* methods: we analyze each thread individually, in an environment consisting of (an abstraction of) the effect of the other threads. This is a form of *rely-guarantee* reasoning [8], but in a fully automatic static analysis settings formalized as abstract interpretation. Contrary to Jones' seminal rely-guarantee proof method or its more recent incarnations [7], our method does not require manual annotations: thread interferences are automatically inferred by the analysis (including which variables are actually shared and their possible values). Following the classic methodology of abstract interpretation [4, 3], a thread-modular static analysis is now viewed as a computable abstraction of a *complete* concrete thread-modular semantics. This permits a fine control between precision and efficiency, and opens the way to analysis specialization: any given safety property of a given program can be theoretically inferred given the right abstract domain.

Following the design-by-refinement principle of Astrée, our first prototype AstréeA [11] used a very coarse but efficient flow-insensitive and non-relational notion of thread interference: it gathered independently for each variable and each thread an interval abstraction of the values the thread can store into the variable along its execution, and injected these values as non-deterministic writes into other threads. This abstraction allowed us to scale up to our target applications, in efficiency (a few tens of hours of computation) if not in precision (a few tens of thousands alarms).

This presentation will describe our subsequent work in improving the precision of AstréeA by specialization on our target applications, and the interesting abstractions we developed along the way. For instance, we developed new interference abstractions enabling a limited but controllable (for efficiency) degree of relationality and flow-sensitivity [12]. We also designed abstractions able to exploit our knowledge of the real-time scheduler used in the analysis target: *i.e.*, it schedules tasks on a single core and obeys a strict priority scheme.[1] The resulting analysis is less general, but more precise on our target applications, which was deemed necessary as the correctness of the applications relies on these hypotheses on the scheduler.[2] Finally, not all false alarms are caused by our abstraction of concurrency; we also developed numeric and memory domains to handle more

[1] The scheduler remains fully preemptive: a low-priority thread may be interrupted at any point by a higher-priority thread whose request to an external resource has just been granted, resulting in a large number of possible thread interleavings.

[2] It is important not to confuse here specialization with restriction: the scheduler abstraction is optional and can be omitted to achieve a more general, but less specialized analysis.

precisely some programming patterns which we did not encounter in our previous experience with Astrée and for which no stock abstract domain was available.

The end-result is a more precise analyzer on our target applications, with currently around a thousand alarms. We stress that AstréeA is a work in progress and that its results, although they are not yet as impressive as those of Astrée, are likely to improve through further specialization. We also believe that, thanks to the intrinsic modularity of the abstract interpretation framework, the analysis performed by AstréeA can be adapted to other settings (other families of applications, other schedulers, other concurrency models) by developing new abstractions, while the abstractions we designed along the journey may also be of use in similar or different static analyses.

References

1. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The Astrée static analyzer, `http://www.astree.ens.fr`
2. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: PLDI 2013, pp. 196–207. ACM (2003)
3. Cousot, P., Cousot, R.: Invariance proof methods and analysis techniques for parallel programs. In: Automatic Program Construction Techniques, ch. 12, pp. 243–271. Macmillan, New York (1984)
4. Cousot, P., Cousot, R.: Abstract interpretation frameworks. Journal of Logic and Computation 2(4), 511–547 (1992)
5. Cousot, P., Cousot, R., Feret, J., Miné, A., Rival, X.: The AstréeA static analyzer, `http://www.astreea.ens.fr`
6. Feret, J.: Static analysis of digital filters. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 33–48. Springer, Heidelberg (2004)
7. Flanagan, C., Freund, S.N., Qadeer, S.: Thread-Modular Verification for Shared-Memory Programs. In: Le Métayer, D. (ed.) ESOP 2002. LNCS, vol. 2305, pp. 262–277. Springer, Heidelberg (2002)
8. Jones, C.B.: Tentative steps toward a development method for interfering programs. ACM TOPLAS 5, 596–619 (1983)
9. Mauborgne, L., Rival, X.: Trace Partitioning in Abstract Interpretation Based Static Analyzers. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 5–20. Springer, Heidelberg (2005)
10. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation 19(1), 31–100 (2006)
11. Miné, A.: Static analysis of run-time errors in embedded critical parallel C programs. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 398–418. Springer, Heidelberg (2011)
12. Miné, A.: Relational thread-modular static value analysis by abstract interpretation. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 39–58. Springer, Heidelberg (2014)

Word-Level Quantifier Elimination

Supratik Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Bombay

A first order theory $\mathcal{T}$ is said to admit *quantifier elimination* if every $\mathcal{T}$-formula of the form $Qx.\,\varphi(x, y_1, \ldots y_n)$, where $\varphi(x, y_1, \ldots y_n)$ is quantifier-free and Q is either $\exists$ or $\forall$, is $\mathcal{T}$-equivalent to a quantifier-free formula of the form $\widehat{\varphi}(y_1, \ldots y_n)$. The process of systematically deriving $\widehat{\varphi}(y_1, \ldots y_n)$ from $\varphi(x, y_1, \ldots y_n)$ is called "quantifier elimination".

Quantifier elimination is an important operation in several verification, synthesis and analysis tasks. When reasoning about hardware and software with finite precision arithmetic, the theory $\mathcal{T}$ of relevance is that of fixed-width bit-vectors (or words). Since each variable in this theory has a fixed finite domain, the theory is easily seen to admit quantifier elimination via expansion of quantified variables. This naive approach, however, does not translate to a practical algorithm for eliminating quantifiers, since the domain of a variable is exponential in its bit-width. Therefore, the formula resulting from expansion of quantified variables blows up exponentially, rendering the naive approach infeasible in practice. Approaches based on bit-blasting $\varphi(x, y_1, \ldots y_n)$, followed by quantifier elimination techniques for quantified propositional formulas are not very useful either, since the result obtained via such approaches have no word-level structure at all. This makes it difficult to apply further word-level reasoning on the formula resulting from quantifier elimination. It is therefore important to develop *word-level quantifier elimination* algorithms that avoid variable expansion and bit-blasting as much as possible, and instead reason directly at the level of bit-vectors (or words).

The importance of word-level quantifier elimination in several application domains has spurred a lot of interesting work in this area in the recent past. This talk surveys these techniques, and discusses in more depth some quantifier elimination algorithms for linear word-level constraints, developed in our research group. Since the output of each word-level quantifier elimination technique is a word-level formula, it is indeed possible to harness the power of multiple techniques in a co-operative manner to quantify a set of variables. The talk concludes with an overview of sub-problems that still remain to be addressed satisfactorily in our quest for word-level quantifier elimination techniques for real-world verification, synthesis and analysis problems.

Early Verification

K. Rustan and M. Leino

Microsoft Research, Redmond, WA, USA
leino@microsoft.com

Abstract. Technology that accurately models, analyzes, and verifies software has come a long way since its conception several decades ago. One mode of using such technology is to look for defects in software that has already left the hands of developers. Another mode is to integrate the technology into the process of software authoring (see, for example, [2,1,5,3,4]). The advantage of this mode is that it lends analytical power to the developer's thinking. To be used in this way, the technology must be packaged in a way that is understandable, unobtrusive, and responsive. In this talk, I showcase an integrated development environment that supports reasoning and verification, trying to provide an aid to the developer earlier during the software development process.

References

[1] Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
[2] Bourdoncle F.: Abstract debugging of higher-order imperative languages. In: Cartwright, R. (ed.) Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation (PLDI), pp. 46–55. ACM (June 1993)
[3] Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16 2010. LNCS, vol. 6355, pp. 348–370. Springer, Heidelberg (2010)
[4] Logozzo, F., Barnett, M., Fähndrich, M., Cousot, P., Cousot, R.: A semantic integrated development environment. In: Leavens, G.T. (ed.) Conference on Systems, Programming, and Applications: Software for Humanity, SPLASH 2012. ACM (October 2012)
[5] Reif, W.: The KIV system: Systematic construction of verified software. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 753–757. Springer, Heidelberg (1992)

Table of Contents