Lecture Notes in Computer Science **10747**

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

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

More information about this series at http://www.springer.com/series/7407

Isil Dillig · Jens Palsberg (Eds.)

Verification, Model Checking, and Abstract Interpretation

19th International Conference, VMCAI 2018 Los Angeles, CA, USA, January 7–9, 2018 Proceedings

Springer

Editors
Isil Dillig
University of Texas
Austin, TX
USA

Jens Palsberg
University of California
Los Angeles, CA
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-73720-1 ISBN 978-3-319-73721-8 (eBook)
https://doi.org/10.1007/978-3-319-73721-8

Library of Congress Control Number: 2017963752

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains the papers presented at VMCAI 2018: the International Conference on Verification, Model Checking, and Abstract Interpretation held during January 7–9, 2018, in Los Angeles, co-located with POPL 2018 (the annual ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages). Previous meetings were held in Port Jefferson (1997), Pisa (1998), Venice (2002), New York (2003), Venice (2004), Paris (2005), Charleston (2006), Nice (2007), San Francisco (2008), Savannah (2009), Madrid (2010), Austin (2011), Philadelphia (2012), Rome (2013), San Diego (2014), Mumbai (2015), St. Petersburg, Florida (2016), and Paris (2017).

VMCAI provides a forum for researchers from the communities of verification, model checking, and abstract interpretation to present their research and aims to facilitate interaction, cross-fertilization, and advancement of hybrid methods that combine these and related areas. VMCAI topics include: program verification, model checking, abstract interpretation, program synthesis, static analysis, type systems, deductive methods, decision procedures, theorem proving, program certification, debugging techniques, program transformation, optimization, hybrid and cyber-physical systems.

This year the conference received 43 submissions, of which 24 were selected for publication in the proceedings. Each submission was reviewed by at least three Program Committee members, and the main selection criteria were quality, relevance, and originality. In addition to the presentations of the 24 selected papers, the conference also featured an invited tutorial by Mayur Naik (University of Pennsylvania) as well as three invited keynote talks by Ken McMillan (Microsoft Research), Azadeh Farzan (University of Toronto), and Ranjit Jhala (University of California San Diego). We warmly thank them for their participation and contributions.

We would like to thank the members of the Program Committee and the external reviewers for their excellent work. We also thank the members of the Steering Committee, and in particular Lenore Zuck and Andreas Podelski, for their helpful advice, assistance, and support. We thank Annabel Satin for her help in coordinating the events co-located with POPL 2018, and we thank the POPL 2018 Organizing Committee for providing all the logistics for organizing VMCAI. We are also indebted to EasyChair for providing an excellent conference management system.

Finally, we would like to thank our sponsors, Amazon Web Services and Facebook, for their valuable contributions to VMCAI 2018.

November 2017

Isil Dillig
Jens Palsberg

Organization

Program Committee

Jade Alglave	University College London, UK
Roderick Bloem	Graz University of Technology, Austria
Wei-Ngan Chin	National University of Singapore, Singapore
Maria Christakis	MPI-SWS, Germany
Patrick Cousot	New York University, USA
Isil Dillig	The University of Texas at Austin, USA
Laure Gonnord	University of Lyon/Laboratoire d'Informatique du Parallélisme, France
Eric Koskinen	Yale University, USA
Laura Kovacs	Vienna University of Technology, Austria
Paddy Krishnan	Oracle, Australia
Ondrej Lhotak	University of Waterloo, Canada
Ruben Martins	Carnegie Mellon University, USA
Ken McMillan	Microsoft, USA
Daniel Neider	Max Planck Institute for Software Systems, Germany
Jens Palsberg	University of California, Los Angeles, USA
Corina Pasareanu	CMU/NASA Ames Research Center, USA
Andreas Podelski	University of Freiburg, Germany
Xiaokang Qiu	Purdue University, USA
Noam Rinetzky	Tel Aviv University, Israel
Philipp Ruemmer	Uppsala University, Sweden
Roopsha Samanta	Purdue University, USA
Rahul Sharma	Microsoft, India
Ana Sokolova	University of Salzburg, Austria
Tachio Terauchi	Waseda University, Japan
Thomas Wahl	Northeastern University, USA
Thomas Wies	New York University, USA
Charles Zhang	The Hong Kong University of Science and Technology, SAR China

Additional Reviewers

Abdullah, Syed Md Jakaria
An, Shengwei
Antonopoulos, Timos
Bartocci, Ezio
Biere, Armin
Cai, Zhuohong
Costea, Andreea
Darais, David
Ebrahimi, Masoud
Esterle, Lukas
Forget, Julien
Gorogiannis, Nikos

Gotlieb, Arnaud
Gu, Yijia
Guatto, Adrien
Hoenicke, Jochen
Humenberger, Andreas
Jansen, Nils
Kiefer, Stefan
Krishna, Siddharth
Lazic, Marijana
Liu, Peizun
Lozes, Etienne
Mottola, Luca
Moy, Matthieu
Niksic, Filip
Pavlinovic, Zvonimir
Poncelet, Clement
Roeck, Franz
Rusu, Vlad
Schäf, Martin
Stuckey, Peter
Suda, Martin
Udupa, Abhishek
Villard, Jules
Wang, Xinyu

Abstracts of Invited Talks

Rethinking Compositionality for Concurrent Program Proofs

Azadeh Farzan

University of Toronto

Abstract. Classical approaches to reasoning about concurrency are based on reductions to sequential reasoning. Typical tactics are to reason about the global behaviour of the system (commonly employed in model checking) or to reason about the behaviour of each thread independently (such as in Owicki-Gries or Rely/Guarantee). We will discuss a new foundation for reasoning about multi-threaded programs, which breaks from this mold. In the new approach, proof ingredients extracted from a few distinct program behaviours are used as building blocks to a program proof that is free to follow the program control structure when appropriate and break away from it when necessary. Our algorithmic solution to the automated construction of these proofs leverages the power of sequential reasoning similar to the classical techniques, but the sequential reasoning lines need not be drawn at the thread boundaries.

Keywords: Proofs · Concurrency · Compositionality

Reasoning About Functions

Ranjit Jhala

University of California, San Diego

Abstract. SMT solvers' ability to reason about equality, arithmetic, strings, sets and maps, have transformed program analysis and model checking. However, SMT crucially relies on queries being restricted to the above theories which preclude the specification and verification of specification and verification of properties of higher-order, user-defined functions. In this talk, we will describe some recent progress towards removing this restriction by presenting two algorithms for SMT-based reasoning about functions.

The first algorithm, FUSION, enables *abstract* reasoning about functions. FUSION generalizes the first-order notions of strongest post-conditions and summaries to the higher-order setting to automatically synthesize the most precise representation of functions expressible in the SMT logic. Consequently, FUSION yields a relatively complete algorithm for verifying specifications over SMT-decidable theories. While this suffices to verify classical (1-safety) specifications, e.g. array-bounds checking, it does not apply to general (k-safety) specifications over user defined functions, *e.g.* that certain functions are commutative or associative.

The second algorithm, PLE *Proof by Logical Evaluation* (PLE), enables *concrete* reasoning about functions, by showing how to mimic computation within SMT-logics. The key idea is to represent functions in a guarded form and repeatedly unfold function calls under enabled guards. We formalize a notion of an equational proof and show that PLE is complete, *i.e.*is guaranteed to find an equational proof if one exists. Furthermore, we show that PLE corresponds to a universal (or must) abstraction of the concrete semantics of the user-defined functions, and hence, terminates, yielding a precise and predictable means of automatically reasoning about user-defined functions.

Joint work with Benjamin Cosman, Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan Scott, Ryan Newton and Philip Wadler.

How to Stay Decidable

Kenneth McMillan

Microsoft Research

Abstract. Automated provers can substantially increase productivity in the formal verification of complex systems. However, the unpredictability of automated provers presents a major hurdle to usability of these tools. This problem is particularly acute in case of provers that handle undecidable logics, for example, first-order logic with quantifiers.

On the other hand, there is a long history of work on *decidable* logics or fragments of logics. Generally speaking, decision procedures for these logics perform more predictably and fail more transparently than provers for undecidable logics. In particular, in the case of a false proof goal, they usually can provide a concrete counter-model to help diagnose the problem. The problem that remains little studied is how to apply these logics in practice in the proof of large systems. That is, how do we effectively decompose the proof of the system into lemmas couched in decidable fragments, and is the human effort required to do this repaid by more reliable automation?

To answer these questions, we must address the fact that combinations of decidable theories are generally not decidable, and that useful decidable fragments are generally not closed under conjunction. This requires us to practice separation of concerns. For example, it is important to express the implementation of a protocol in a language that captures the protocol's logic without mixing in low-level details such as data structures. Moreover, modularity is an important tool for avoiding undecidability. For example, we can use a high-level protocol model to prove global properties, which are then used as lemmas in proving correctness of the protocol implementation. This allows us to separate invariants which, if combined, would take us outside the decidable realm. In particular, this strategy allows us to produce verification conditions that are decidable because they use function symbols in a stratified way.

Preliminary experience indicates that it is possible to produce verified implementations of distributed protocols in this way with reduced proof complexity and greater reliability of proof automation, without sacrificing execution performance.

Maximum Satisfiability in Program Analysis: Applications and Techniques (Invited Tutorial)

Mayur Naik[1], Xujie Si[1], Xin Zhang[1], and Radu Grigore[2]

[1] University of Pennsylvania
[2] University of Kent

Abstract. A central challenge in program analysis concerns balancing different competing tradeoffs. To address this challenge, we propose an approach based on the Maximum Satisfiability (MaxSAT) problem, an optimization extension of the Boolean Satisfiability (SAT) problem. We demonstrate the approach on three diverse applications that advance the state-of-the-art in balancing tradeoffs in program analysis. Enabling these applications on real-world programs necessitates solving large MaxSAT instances comprising over 10^{30} clauses in a sound and optimal manner. We propose a general framework that scales to such instances by iteratively expanding a subset of clauses while providing soundness and optimality guarantees. We also present new techniques to instantiate and optimize the framework.

Keywords: Maximum satisfiability · Program analysis

Contents