Int J Softw Tools Technol Transfer (2013) 15:287–289
DOI 10.1007/s10009-013-0280-3

Model checking and abstract interpretation as building blocks of advanced program analysis techniques

Selected papers from TACAS 2009

Stefan Kowalewski · Anna Philippou · Jörg Brauer

Published online: 27 June 2013

1 Introduction

Formal analysis of software systems can look back on a close to 50 years history and, yet, significant progress is being made today by either new approaches or innovative combinations of existing techniques. In the 1960s, Floyd [5] and Hoare [7] provided the emerging field of computer science with a fundamental understanding of (partial and total) program correctness. Given a precondition P of a program φ, establishing a correctness argument amounts to deriving the strongest description Q of φ subject to P. A correctness proof in Floyd–Hoare logic is then represented as a tuple:

$$\{P\}\ \varphi\ \{Q\}$$

Ideally, Q somehow describes the desired correctness argument of φ if the precondition P holds on input to the program. This simple concept has had an enormous impact on our understanding of programming; it is not only the basis for a vast number of publications in the field of program verification, but also forms part of the syllabus for undergraduate courses in computer science.

However, the general notion of Floyd–Hoare logic has left computer scientists with a question whose impact cannot be overestimated in the development of our field: *How can we effectively and efficiently compute Q?* Many outstanding contributions to the field of verification have been motivated by the desire to find an answer to this simple yet fundamental question, leading to a widespread collection of techniques that were invented and successively refined over several decades. An enumeration of these techniques should of course list terms such as (explicit-state, symbolic and bounded) model checking [1,2,4,12], abstract interpretation [3], automated theorem proving, testing, symbolic execution [9] and many more.

The longstanding efforts in studying these techniques and applying them to verification problems have then led to some conclusions:

– Each approach exhibits strengths in certain situations.
– Yet, neither technique perfectly solves all verification problems in practice.

To illustrate these conclusions by means of an example, let us consider model checking of software systems: explicit-state model checking exhibits its strengths if programs consisting of concurrent components are analyzed; symbolic model checking is strong if large amounts of data in a program can be efficiently represented by logical descriptions; bounded model checking is considered an effective approach for bug-hunting rather than proving the absence of errors. A natural conclusion from these observations is that attempts to solve each and every verification or analysis problem using a single technique is likely to fail.

Although this conclusion may lead to some frustration, it provides the potential for a more general approach to verification: modern verification tools are formed from *building blocks* of verification techniques, thereby combining different approaches so as to profit from their individual strengths.

S. Kowalewski (✉)
RWTH Aachen University, 52056 Aachen, Germany
e-mail: kowalewski@cs.rwth-aachen.de;
kowalewski@embedded.rwth-aachen.de

A. Philippou
University of Cyprus, 1678 Nicosia, Cyprus
e-mail: annap@cs.ucy.ac.cy

J. Brauer
Verified Systems International GmbH, Am Fallturm 1,
28359 Bremen, Germany
e-mail: brauer@verified.de

An example of a widely accepted approach, which has been implemented in many model checkers, is to use a relatively cheap static analysis to derive equivalence-preserving abstractions that ease the model checking process [13]. This combination of static analysis and model checking achieves synergies since it preserves the strengths of model checking whilst benefiting from the tractability of static analyses.

The conference *Tools and Algorithms for the Construction and Analysis of Systems* (*TACAS*) has always considered itself a forum for researchers from different disciplines within the verification community, not limiting the accepted publications to a certain field. The conference thus supports the convergence of seemingly orthogonal techniques into practical verification frameworks that solve the complex verification problems posed by modern (software and hardware) systems.

2 Contributions to this issue

This special issue contains extended versions of outstanding papers that were selected from TACAS 2009. The selection also highlights the impact of combinations of techniques on the effectiveness of verification.

The paper *From Tests To Proofs* [6] studies how invariant generation based on constraint solving can benefit from both abstract interpretation and dynamic execution of programs. It thus provides a framework in which the scalability of automatic invariant generation is strengthened by the application of other computationally less expensive techniques. The approach derives additional linear constraints from sets of reachable states, either explicit or symbolic, which then lead to a significant simplification of the invariant generation by the constraint solver. Extensive experiments show that the approach not only speeds up the invariant computation, but also allows handling examples not treatable by other state-of-the-art tools, and that the additional strengthening does not introduce a significant running time overhead in cases for which static constraint solving is already quite well performing.

Similar in spirit to [6], *Falsification of LTL Safety Properties in Hybrid Systems* [11] combines automata-based model checking for hybrid systems with motion planning. Both components exchange information on the fly so as to find trajectories through systems that falsify the specification. The authors first extend approaches using tree-search-based motion planning frameworks to falsify LTL safety properties in hybrid systems by using the automaton $\mathcal{A}$ for the negated LTL property as an external monitor. Then, this technique is combined with model checking using an abstraction of the hybrid system to compute sequences of propositional assignments representing discrete witnesses of the violation of the LTL safety property. The approach is illustrated by a robot navigation benchmark. Experiments show that the combination of motion planning and model checking has significant performance advantages over applying only tree search motion planning with $\mathcal{A}$ as a monitor.

In *Static Analysis for Concurrent Programs with Applications to Data Race Detection* [8], the authors combine different techniques to analyze concurrent multi-threaded programs with locks and rendezvous. The basic model is a graph structure over the global control state space of a program, representing atomically executable sequences of instructions called transactions. This transaction graph is generated automatically and then incrementally refined using different techniques, most notably abstract interpretation for deriving program invariants. Further, model checking is applied to derive concrete counterexample traces from the final abstraction of the concurrent program. The approach is illustrated by application to the detection of data race bugs. Experimental evaluation for a set of Linux device drivers demonstrates the advantage of generating small sliced models by the approach, which lead to successful analysis even of comparably large drivers.

Finally, the paper *Selected Dynamic Issues in Software Model Checking* [10] explains how a model checker for `.NET` programs benefits from the combination of model checking with static analysis and the addition of variations of well-known programming idioms such as exception handling and garbage collection. Additionally, a memory reduction scheme is introduced to decrease the amount of stored metadata generated during verification by dynamic partial order reduction. The approach is evaluated and compared to the Java PathFinder model checker by application to a Java benchmark suite. The results show considerable performance improvements.

A commonality of these fascinating papers is that, while focusing on very specific verification goals, they combine different techniques to overcome the efficiency problems associated with program verification: they achieve synergies by exploiting the advantages of different verification techniques and thereby advance the state of the art in verification.

References

1. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Adv. Comput. **58**, 117–148 (2003)
2. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science, pp. 428–439 (1990)
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 238–252. Los Angeles, California (1977)

4. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
5. Floyd, R.W.: Assigning meanings to programs. In: Proceedings of Symposium on Applied Mathematics, vol. 19, pp. 19–31 (1967)
6. Gupta, A., Majumdar, R., Rybalchenko, A.: From Tests to Proofs. Int. J. Softw. Tools. Technol. Transfer. (this volume). doi:10.1007/s10009-012-0267-5
7. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM. **12**(10), 576–580 (1969)
8. Kahlon, V., Sankaranarayanan, S., Gupta, A.: Static Analysis for concurrent programs with applications to data race detection. Int. J. Softw. Tools. Technol. Transfer. (this volume). doi:10.1007/s10009-013-0274-1
9. King, J.C.: Symbolic execution and program testing. Commun. ACM. **19**(7), 385–394 (1976)
10. Nguyen, V.Y., Ruys, T.C.: Selected dynamic issues in software model checking. Int. J. Softw. Tools. Technol. Transfer. (this volume). doi:10.1007/s10009-012-0261-y
11. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Falsification of LTL safety Properties in hybrid systems. Int. J. Softw. Tools. Technol. Transfer. (this volume). doi:10.1007/s10009-012-0233-2
12. Queille, J.-P., Sifakis, J.: Specification and verification of concurrent systems. In: CESAR. Symposium on Programming, pp. 337–351 (1982)
13. Yorav, K., Grumberg, O.: Static analysis for state-space reductions preserving temporal logics. Formal Methods Syst. Design. **25**(1), 67–96 (2004)