Int J Softw Tools Technol Transfer (2017) 19:511–515
DOI 10.1007/s10009-017-0460-7

INTRODUCTION

Advances in verification presented in TACAS'13

Nir Piterman[1]

Published online: 9 June 2017

Abstract Computers are becoming increasingly ubiquitous in all aspects of our life. It is becoming more and more important to ensure that the software (and hardware) that drives them performs as expected. Verification is one approach to improve quality of software and hardware. Verification attempts to formally prove that programs or systems fulfill desired properties and lack undesirable properties. This is a thriving area of research, and much resources are invested in extending it both in academia and in industry. In this special issue, we introduce four papers on verification selected from the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13).

Keywords Verification · Model checking · Static analysis

1 Introduction

This special issue of the journal *Software Tools for Technology Transfer* (STTT) contains revised and extended versions of four papers selected out of 42 papers presented at the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13) [37]. The peer-reviewed papers collected in this special issue have been invited by the guest editors among the top papers presented at TACAS'13 based on their relevance to STTT. They all report on advances in verification that relate to domains that, in our opinion, are central to make further progress in verification.

As computers and the software that drives them become more important and more prevalent there is a growing need in ensuring that they work correctly. Verification is the field of research that aims to produce tools and techniques for proving correctness of programs and finding bugs that could hinder their performance. An active research community and much demand in industry for better and more scalable verification techniques are driving a thriving research field. In spite of much progress that was done in this field there is much scope for further research. With every success, our appetite for increasing the scope of verification increases and we try to tackle more general and bigger problems. Correctness through the life cycle of a system, scalability of verification, the challenges imposed by concurrent and distributed systems, and the importance of identification of areas where verification can be applied more efficiently are some of the domains in which interesting research is ongoing.

The selected papers cover these four domains. Our discussion of these papers is organized as follows. Section 2 discusses the verification of a system through its life cycle. Section 3 discusses compositional verification. Section 4 discusses verification of concurrent and distributed systems. Section 5 discusses the applications of model checking to recursive programs that have integer variables. Finally, Sect. 6 concludes the paper.

2 Model checking through code life cycle

One of the major challenges in the software life cycle is maintenance. This includes the fixing of bugs but also the addition of features or simply restructuring of code to facilitate fixes and improvements. However, many of the techniques and tools that are developed for analysis and verification of software is targeted at new software. One can apply these techniques for existing software but the involved effort does not take advantage of previous verification efforts. Indeed, with frequent (and small) changes to the software, it is

✉ Nir Piterman
nir.piterman@gmail.com

[1] University of Leicester, Leicester, UK

prohibitively expensive to redo the verification effort from scratch. The formal methods community is putting more efforts into the usage of its tools and techniques during the life cycle of a system.

In [11,39] the authors introduce the concept of *substitutivity*, where a part of a software system replaces a different part and the task of verification is to check that all required behaviors of the previous component are still available and that no new behaviors that lead to the violation of specification are added.

This idea is extended encompass the concept of regression testing in [25]. Regression testing is well known for both software and hardware testing. A large set of tests is collected that checks many features of the design. Changes to the design are validated by running the regression and ensuring that the results match those of the previous version. Godlin and Strichman suggest that two related programs can be checked for equivalence utilizing the similarity between the two programs to facilitate the proof of equivalence. This idea is also very appealing in the absence of formal specification as the previous version of the program can be used as defacto specification. One of the problems with this approach is that complete equivalence is too restricting. The authors of [25] consider various ways to allow for equivalence modulo some beneficial changes. In [4] the authors consider ways to improve efficiency of regression verification by distinguishing between behaviors that are impacted by the change and those that are not impacted by the change between the two versions.

A similar usage of the similarity of the program is applied in the context of existing verification efforts in [30]. Here, a subset of the assertions checked over the original program is considered and the goal of the verification task is to find scenarios according to which the original program and the new program differ regarding specific assertions. The assertion checking can be applied compositionally by considering functions or parts of the program and does not need to take into account the entire program.

Using the results of previous verification efforts has been applied also in the context of symbolic execution of programs [23]. The authors consider *whitebox fuzzing* [24], where symbolic executions of a program are used to drive dynamic tests for the program. In order to reuse part of the effort that went into the symbolic execution and its analysis, the authors suggest to keep summaries of the symbolic executions. It is then easier to check whether previous summaries are still applicable to the new version of the program rather than performing symbolic execution from scratch. From existing program summaries, it is easy to produce new test cases saving significant resources.

This is the topic of the paper *Flexible SAT-based Framework for Incremental Bounded Upgrade Checking* by Fedyukovich et al. [20], which extends the TACAS'13 conference paper [19]. They operate in the context of bounded model checking of software programs. They assume that a program that has been successfully verified in the past has been changed and needs a new verification effort. In order to apply this technique, the initial verification effort needs to be extended by further analysis. This further analysis extracts summaries of the behavior of analyzed functions. These summaries can then be used for checking the new version of the program. For functions that have not changed or changed in a way that maintains their behavior the existing summaries can be proven by considering relatively small parts of the program. It is important to note that summaries are artifacts that relate to the global correctness argument and hence abstract the full behavior of the functions. Thus, it is conceivable that even changed functions maintain the same correctness criteria. But even functions that do change can rely on summaries for the functions that they are calling and there is much hope that (if the new program is indeed correct) the new analysis effort will not have to go all the way to the root of the program. The authors report on their technique and its implementation in a tool called EVOLCHECK. They also included several case studies that show the benefit of applying this technique.

3 Compositional verification

One of the most challenging aspects of verification is the size of the systems that are being handled. One of the approaches to tackle this problem is to suggest compositional techniques that allow to reason about parts of the system but deduce properties of the entire system. Generally stated, the idea of *assume-guarantee*, is to consider a part P_i of a larger system and prove that $\langle a_i \rangle M \langle g_i \rangle$, i.e., that under the assumption a_i the part P_i guarantees the guarantee g_i. Then, relationships between the assumptions of the different parts and the guarantees of other parts need to be established so that when the parts interact all assumptions hold and, as consequence, all guarantees hold as well. The advantage is that most reasoning is done at the level of the parts of the system, which are smaller and easier to handle. The interest in such techniques started very early in the history of verification. Notably, Owicki and Gries suggested a framework for compositional verification of concurrent programs [35]. Their work inspired the extension of the assume-guarantee approach to, e.g., temporal-logic reasoning [38].

One of the main obstacles to application of the compositional approach to verification is the complexity of assumptions and (local) guarantees that together are able to imply the global correctness goal. Much research has been dedicated to different approaches that try to automatically divine the different parts of the proof. For example, learning algorithms were suggested to learn general assumptions about the different parts of the system [12,13,36]. The idea is

to have a learning algorithm create a candidate for an assumption and use failures of the proof to give the learner ideas on how to refine its candidate. A related approach replaces the learning by abstraction and abstraction refinement [8]. An abstraction of one part is used as a guarantee of this part hoping that it will be strong enough to discharge the guarantee of other parts. If this fails, the counter example is used as either a real counter example or to refine the abstraction. From a different field, Calcagno et al. perform compositional verification of heap manipulating software programs using *Separation Logic* [10]. Starting from Hoare triples describing the preconditions and postconditions of existing functions they use abduction to infer extra conditions on the structure of the heap that are required to discharge the postconditions. These inferred preconditions then need to be discharged by themselves by reasoning about other parts of the code.

It is in this scope that the paper *Synthesis of Circular Compositional Program Proofs via Abduction* by Dillig et al. [18], which extends the TACAS'13 conference paper [31], suggests a way to synthesize the parts of a compositional proof of correctness. The authors suggest several proof rules that allow for circular dependencies between the different assumptions and guarantees but are still sound in their conclusions. The general framework partitions the global proof to a sequence of subgoals that need to be discharged. A distinct advantage of their approach is that they allow for the combination of multiple approaches for the discharge of different subgoals. What's more, the failure to prove certain subgoals can lead to their technique suggesting the partition of the subgoal to further sub-subgoals, which are then attempted using the same approach. This gives rise to an overall lazy approach to compositional verification whereby the approach starts from general goals and finds by itself the structure of the proof that will prove these goals.

4 Concurrency and distributed systems

Concurrent and distributed systems have always interested the verification community. However, over the past few years, two major processes have led to concurrent and distributed systems to increase considerably in prevalence and in importance. First, multi-core processors are replacing the increase in processor speed as the main tool to increase the power of processors. This arises from the physical implications of heat dissipation in processors and making increases in speed physically impossible. Second, the emergence and prevalence of communication caused an explosion in the need and supply of programs that interact and collaborate to fulfill a common goal. However, concurrent and distributed systems pose even further challenges to verification. Such programs have all the features that make verification hard (such as variables ranging over infinite domains, usage of heap, recursion) but in addition include additional features of concurrency, communication, and synchronization. For example, when trying to verify a data structure that is intended to be used in a concurrent environment one has to cope with the usual difficulties of verification. Namely, the data structure is unbounded and uses the heap to store data. Furthermore, the data itself come from infinite domains. In addition to these difficulties, in the concurrent setting, verification has to reason about a potentially unbounded number of threads accessing the data structure. Thus, the unboundedness of checking arises in three different axes. The breadth and wealth of work done on verification of such systems makes it impossible to describe here. We choose to highlight some work on linearizability that is related to the work that appears in this special issue.

The correctness criterion suggested to verify concurrent data structures is that of *linearizability* [27]. Some recent approaches to checking linearizability include the following. Being able to prove that a concurrent data structure is linearizable is a major challenge that called much attention to it. For example, in [3] data structures that are based on a concurrent implementation of a singly linked list are verified by maintaining a connection between the concurrent and a sequential implementation of the same data structure. In [32], the authors use refinement between the concurrent and the sequential data structures to verify linearizability without specifying explicit points where the concurrent data structure can be identified with the sequential. Vafeiadis shows how to combine rely-guarantee with separation logic to compare the linearization points of a concurrent data structure with its sequential version [41,42]. Another attempt to remove the need to identify the linearization points is presented in [34] and replace them by verifying local invariants on the threads that imply a global invariant on the data structure.

In the paper *An Integrated Specification and Verification Technique for Highly Concurrent Data Structures* by Abdulla et al. [2], which extends the TACAS'13 conference paper [1], the authors consider the topic of verifying data structures that are used in a concurrent environment. They combine several approaches in order to enable automatic verification. They use a novel type of automata and apply the automata-theoretic approach to verification using them. These automata are particularly suitable for capturing the correctness criteria of such data structures. Furthermore, they use static analysis to capture the state of the heap and follow reachability between pointers on the heap. Data abstraction is used in order to handle the infiniteness of the data domain by restricting the number of times that a value appears in the data structure to one. For the unbounded number of threads, they show that considering two threads is sufficient for checking the correctness of these properties. The approach is implemented and in addition to verifying (automatically) data structures that have not been verified in the past they show that for cases that have been in the scope of previous tools the new

techniques affords some acceleration of the effort involved in verification. The TACAS'13 paper [1] already inspired further work. For example, in [9] the approach is extended to more expressive properties and handles more data structures.

5 Underapproximating integer programs

Over the years, the verification community dedicated much attention to programs whose variables are restricted to integer variables. Such programs appear in many domains and the removal of floating point variables has proven to be very beneficial. The approach led to many successful verification efforts. Increasing the scope of verification and allowing to verify large code bases effectively.

Software model checking has made a huge progress in the past 10 years alongside the major progress in satisfiability (SAT) and satisfiability modulo theories (SMT) solving techniques. Several tools for checking whether errors are reachable concentrate on programs that manipulate integer variables and concentrate on the control flow in such programs. For example, tools such as Impact [33] and SeaHorn [26] harness the strength of SMT solvers with the theory of integers (and linear inequalities in particular) to offer scalable solutions for reachability analysis of software (cf. [7] for a presentation of additional related tools). Such tools have also been harnessed for model-checking efforts that consider temporal-logic model checking [14–17]. One of the important aspects of software model checking has been the combination of over- and underapproximation [5,6] as can be seen in, e.g., [28], where it is applied to the analysis of recursive programs just like the paper included in this special issue.

In the paper *Underapproximation of Procedure Summaries for Integer Programs* by Ganty et al. [22], which extends the TACAS'13 conference paper [21], the authors analyze recursive programs that use integer variables. They compute underapproximations of program behavior by bounding recursion and analyzing the resulting programs using existing tools for underapproximations of non-recursive programs. They show how to generalize the results for the potentially unbounded usage of recursion and under what conditions on the recursive program the result is complete. They also present an experimental evaluation of the technique.

6 Conclusions

We have discussed some recent advances in verification and presented papers selected from those presented in TACAS 2013. These papers highlight areas of research that we believe are important for continuing success of verification. We have discussed the following topics. How to consider verification throughout the life cycle of a system and how verification techniques can be modified in order to support that. The issue of compositional verification and its relation to improving the capacity of verification. How to apply verification to concurrent and distributed systems. The importance of restricted domains for application of verification such as integer programs.

Acknowledgements We are grateful to all the authors for their contributions, to the reviewers and program committee of TACAS 2013 in their help for selecting the conference program and the papers for this issue, especially to the referees who reviewed the extended version of the papers that appear in this special issue.

References

1. Abdulla, P.A., Haziza, F., Holík, L., Jonsson, B., Rezine, A.: An integrated specification and verification technique for highly concurrent data structures. In Piterman and Smolka [37], pp. 324–338
2. Abdulla, P.A., Haziza, F., Holík, L., Jonsson, B., Rezine, A.: An integrated specification and verification technique for highly concurrent data structures. Softw. Tools Technol. Transf. (2017, in this issue)
3. Amit, D., Rinetzky, N., Reps, T.W., Sagiv, M., Yahav, E.: Comparison under abstraction for verifying linearizability. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3–7, 2007, Proceedings, Volume 4590 of Lecture Notes in Computer Science, pp. 477–490. Springer, Berlin (2007)
4. Backes, J.D., Person, S., Rungta, N., Tkachuk, O.: Regression verification using impact summaries. In: Bartocci, E., Ramakrishnan, C.R. (eds.) Model Checking Software—20th International Symposium, SPIN 2013, Stony Brook, NY, USA, July 8-9, 2013. Proceedings, Volume 7976 of Lecture Notes in Computer Science, pp. 99–116. Springer, Berlin (2013)
5. Ball, T., Rajamani, S.K.: The SLAM toolkit. In: Berry, G., Comon, H., Finkel, A. (eds.) Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18–22, 2001, Proceedings, Volume 2102 of Lecture Notes in Computer Science, pp. 260–264. Springer, Berlin (2001)
6. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker blast. STTT **9**(5–6), 505–525 (2007)
7. Bjørner, N., Gurfinkel, A., McMillan, K.L., Rybalchenko, A.: Horn clause solvers for program verification. In: Beklemishev, L.D., Blass, A., Dershowitz, N., Finkbeiner, B., Schulte, W. (eds.) Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday, Volume 9300 of Lecture Notes in Computer Science, pp. 24–51. Springer, Berlin (2015)
8. Bobaru, M.G., Pasareanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: Gupta, A., Malik, S. (eds.) Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7–14, 2008, Proceedings, Volume 5123 of Lecture Notes in Computer Science, pp. 135–148. Springer, Berlin (2008)
9. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Tractable refinement checking for concurrent objects. In: Rajamani, S.K., Walker, D.

(eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15–17, 2015, pp. 651–662. ACM, New York (2015)
10. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. J. ACM **58**(6), 26:1–26:66 (2011)
11. Chaki, S., Sharygina, N., Sinha, N.: Verification of evolving software. In: 3rd Workshop on Specification and Verification of Component-Based Systems (2004)
12. Chaki, S., Strichman, O.: Three optimizations for assume-guarantee reasoning with L*. Formal Methods Syst. Des. **32**(3), 267–284 (2008)
13. Chen, Y.-F., Clarke, E.M., Farzan, A., Tsai, M.-H., Tsay, Y.-K., Wang, B.-Y.: Automated assume-guarantee reasoning through implicit learning. In: Touili et al. [40], pp. 511–526
14. Cook, B., Khlaaf, H., Piterman, N.: On automation of CTL* verification for infinite-state systems. In: Kroening and Pasareanu [29], pp. 13–29
15. Cook, B., Koskinen, E.: Reasoning about nondeterminism in programs. In: Boehm, H.-J., Flanagan, C. (eds.) ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16–19, 2013, pp. 219–230. ACM, New York (2013)
16. Cook, B., Podelski, A., Rybalchenko, A.: Proving thread termination. In: Ferrante, J., McKinley, K.S. (eds.) Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10–13, 2007, pp. 320–330. ACM, New York (2007)
17. Cook, B., Podelski, A., Rybalchenko, A.: Proving program termination. Commun. ACM **54**(5), 88–98 (2011)
18. Dillig, I., Dillig, T., Li, B., McMillan, K.L., Sagiv, M.: Synthesis of circular compositional program proofs via abduction. Softw. Tools Technol. Transf. (2017, in this issue)
19. Fedyukovich, G., Sery, O., Sharygina, N.: evolcheck: Incremental upgrade checker for C. In: Piterman and Smolka [37], pp. 292–307
20. Fedyukovich, G., Sery, O., Sharygina, N.: Flexible sat-based framework for incremental bounded upgrade checking. Softw. Tools Technol. Transf. (2017, in this issue)
21. Ganty, P., Iosif, R., Konecný, F.: Underapproximation of procedure summaries for integer programs. In: Piterman and Smolka [37], pp. 245–259
22. Ganty, P., Iosif, R., Konecný, F.: Underapproximation of procedure summaries for integer programs. Softw. Tools Technol. Transf. (2017, in this issue)
23. Godefroid, P., Lahiri, S.K., Rubio-González, C.: Statically validating must summaries for incremental compositional dynamic test generation. In: Yahav, E. (ed.) Static Analysis—18th International Symposium, SAS 2011, Venice, Italy, September 14–16, 2011. Proceedings, Volume 6887 of Lecture Notes in Computer Science, pp. 112–128. Springer, Berlin (2011)
24. Godefroid, P., Levin, M.Y., Molnar, D.A.: Automated whitebox fuzz testing. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February–13th February 2008. The Internet Society (2008)
25. Godlin, B., Strichman, O.: Regression verification. Proceedings of the 46th Design Automation Conference, DAC 2009, San Francisco, CA, USA, July 26–31, 2009, pp. 466–471. ACM, New York (2009)
26. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The seahorn verification framework. In: Kroening and Pasareanu [29], pp. 343–361
27. Herlihy, M., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990)
28. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Formal Methods Syst. Des. **48**(3), 175–205 (2016)
29. Kroening, D., Pasareanu, C.S. (eds.): Computer Aided Verification–27th International Conference, CAV 2015, San Francisco, CA, USA, July 18–24, 2015, Proceedings, Part I, Volume 9206 of Lecture Notes in Computer Science. Springer, Berlin (2015)
30. Lahiri, S.K., McMillan, K.L., Sharma, R., Hawblitzel, C.: Differential assertion checking. In: Meyer, B., Baresi, L., Mezini, M. (eds.) Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18–26, 2013, pp. 345–355. ACM, New York (2013)
31. Li, B., Dillig, I., Dillig, T., McMillan, K.L., Sagiv, M.: Synthesis of circular compositional program proofs via abduction. In: Piterman and Smolka [37], pp. 370–384
32. Liu, Y., Chen, W., Liu, Y.A., Sun, J.: Model checking linearizability via refinement. In: Cavalcanti, A., Dams, D. (eds.) FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2–6, 2009. Proceedings, Volume 5850 of Lecture Notes in Computer Science, pp. 321–337. Springer, Berlin (2009)
33. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings, Volume 4144 of Lecture Notes in Computer Science, pp. 123–136. Springer, Berlin (2006)
34. O'Hearn, P.W., Rinetzky, N., Vechev, M.T., Yahav, E., Yorsh, G.: Verifying linearizability with hindsight. In: Richa, A.W., Guerraoui, R. (eds.) Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25–28, 2010, pp. 85–94. ACM, New York (2010)
35. Owicki, S.S., Gries, D.: Verifying properties of parallel programs: an axiomatic approach. Commun. ACM **19**(5), 279–285 (1976)
36. Pasareanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. Formal Methods Syst. Des. **32**(3), 175–205 (2008)
37. Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems—19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16–24, 2013. Proceedings, volume 7795 of Lecture Notes in Computer Science. Springer, Berlin (2013)
38. Pnueli, A.: In Transition from Global to Modular Temporal Reasoning About Programs. Springer, Berlin (1985)
39. Sharygina, N., Chaki, S., Clarke, E.M., Sinha, N.: Dynamic component substitutability analysis. In: Fitzgerald, J.S., Hayes, I.J., Andrzej, T. (eds.) FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Newcastle, UK, July 18–22, 2005, Proceedings, Volume 3582 of Lecture Notes in Computer Science, pp. 512–528. Springer, Berlin (2005)
40. Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15–19, 2010. Proceedings, Volume 6174 of Lecture Notes in Computer Science. Springer, Berlin (2010)
41. Vafeiadis, V.: Shape-value abstraction for verifying linearizability. In: Jones, N.D., Müller-Olm, M. (eds.) Verification, Model Checking, and Abstract Interpretation, 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18–20, 2009. Proceedings, Volume 5403 of Lecture Notes in Computer Science, pp. 335–348. Springer, Berlin (2009)
42. Vafeiadis, V.: Automatically proving linearizability. In: Touili et al. [40], pp. 450–464