A Broader View on Verification: From Static to Runtime and Back (Track Summary)

Wolfgang Ahrendt[1], Marieke Huisman[2(✉)], Giles Reger[3], and Kristin Yvonne Rozier[4]

[1] Chalmers University of Technology, Gothenburg, Sweden
[2] University of Twente, Enschede, The Netherlands
m.huisman@utwente.nl
[3] University of Manchester, Manchester, UK
[4] Iowa State University, Ames, IA, USA

Abstract. When seeking to verify a computational system one can either view the system as a *static* description of possible behaviours or a *dynamic* collection of observed or actual behaviours. Historically, there have been clear differences between the two approaches in terms of their level of completeness, the associated costs, the kinds of specifications considered, how and when they are applied, and so on. Recently there has been a concentrated interest in the combination of static and runtime (dynamic) techniques and this track (taking place as part of ISoLA 2018) aims to explore this combination further.

1 Motivation and Goals

Traditionally, program verification has been *static* in nature, e.g. it has sought to verify that a program, given as a piece of code in a programming language, satisfies a specification for all possible inputs to that program. This approach can be traced back to the seminal work of Floyd and Hoare, which has grown into the mature field of *Deduction Software Verification* [2,15]. Another successful static approach is that of *(Software) Model Checking* [12] where the state space of the program is searched looking for bad states (and trying to establish their absence). Many other static analysis techniques make use of the source code of a program to extract information or establish properties of interest. Conversely, the fields of *Runtime Verification* [5] and *Runtime Assertion Checking* [11,13] abstract a program as a set of behaviours observed at *runtime*, which significantly restricts the coverage of verification whilst improving efficiency by focusing on "real" behaviours.

Recently, there have been multiple proposals to combine static with runtime verification techniques [1,17,18]. Results from static verification can be exploited to reduce the overhead of runtime verification [3,4,7], while results from runtime verification can be is used to fill "holes" left by static verification, where static results are too costly, or impossible, to establish. Static results can moreover

T. Margaria and B. Steffen (Eds.): ISoLA 2018, LNCS 11245, pp. 3–7, 2018.
https://doi.org/10.1007/978-3-030-03421-4_1

increase confidence over what runtime verification can achieve alone. And runtime analysis can learn information that can be used later for static analysis (e.g., mining invariants, and detecting data structures) [21]. Finally, it has been observed that the two approaches (static and runtime) tend to focus on different kinds of specifications with static techniques often focussing on simpler state-based properties and runtime techniques focussing on richer temporal properties. This leads to a simple benefit from combination – a wider range of properties can be checked. Significant progress has been made in these directions, but it is also evident that the communities, targets, methods, and discussions in static and runtime verification research are still too unrelated. There is even less of a connection between early design-time verification techniques (e.g., model checking), the intermediate (e.g., code analysis), and execution-time (e.g., runtime monitoring) verification techniques in the system development cycle.

Within this track, we investigate what can be achieved in joint efforts of the communities of verification techniques for different design stages, including static and dynamic analyses, model checking, deductive verification, and runtime verification. This track is a follow-up of the track *Static and Runtime Verification: Competitors or Friends?*, organised during ISoLA 2016 [18]. That track investigated the different ways in which static and runtime verification could be combined, addressed questions such as which application areas could benefit from combinations of static and dynamic analysis, which program properties can be established using which technique, what artefacts can be carried forward through different verification technologies, what is a good balance of guarantees and user efforts in various application areas, and how we can integrate the various techniques into the software, and hybrid/cyber-physical system development process. We believe that the track in 2016 was a very good catalyst for the connection of the different communities, and the formation of common agendas. At the same time, it is clear that a one-time event would not be enough to fully achieve and sustain the set goals.

The current track *A Broader View on Verification: From Static to Runtime and Back*, at ISoLA 2018, continues and further develops the cross-community endeavour for an integrated interplay of static and runtime techniques, with the overall aim of building systems that are evidently well-functioning. The track features eight contributions, by 24 authors in total, on three different topics: application areas; classes of properties; and practical issues of combining static and runtime techniques. We would like to thank all the authors for their contributions to this track, in the form of the papers written, and in the form of on-site discussions. We would also like to thank everyone who visited this track at ISoLA, showed interest in the overall topic and the individual talks, and participated in the discussions.

Finally, we would like to express our deep gratitude to the ISoLA organisers, in particular Tiziana Margaria and Bernhard Steffen, for working so hard to provide such a wonderful platform for our and other tracks, enabling lively and creative interaction between individuals and communities, helping us all to not

forget the bigger picture of working for the development of systems that people can rely on.

2 Contributions

2.1 Topic 1: Application Areas for Combining Static and Runtime Verification

This topic is dedicated to different application areas of system development exploiting static and dynamic techniques.

In *Programming Safe Robotics Systems: Challenges, Advances, and Opportunities* [14], Ankush Desai, Sanjit Seshia, and Shaz Qadeer present a programming framework for building a safe robotics system. It consists of a high-level programming language for implementing, specifying, and systematically testing the reactive robotics software, as well as a runtime enforcement system ensuring that assumptions made in the testing phase actually hold at runtime.

In *Generating Component Interfaces by Integrating Static and Symbolic Analysis, Learning, and Runtime Monitoring* [19], Falk Howar, Dimitra Giannakopoulou, Malte Mues, and Jorge Navas present extensions to a tool for interface generation that integrates interpolation and symbolic search. Also, they discuss how to use information from runtime monitoring to validate the generated interfaces.

2.2 Topic 2: What are the Relevant Program Properties?

This topic is dedicated to the specific classes of program properties that deserve particular attention when combining static and runtime verification.

In *Monitoring Hyperproperties by Combining Static Analysis and Runtime Verification* [8], Borzoo Bonakdarpour, César Sánchez, and Gerardo Schneider study the problem of runtime verifying temporal hyperproperties, in particular those that involve quantifier alternation. Starting from the observation that virtually no $\forall\exists$ property can be fully monitored at runtime, they propose a combination of static analysis and runtime verification to manage the checking of such formulas. In addition, they also discuss how the notion of hyperproperties should be extended to also consider properties that relate three or more traces.

In *Temporal Reasoning on Incomplete Paths* [16], Dana Fisman and Hillel Kugler explore semantics for temporal logics on paths that are incomplete in different ways. They study incomplete ultimately periodic paths, segmentally broken paths, and combinations thereof, and discuss whether systems biology can benefit from the suggested extensions.

In *A Framework for Quantitative Assessment of Partial Program Correctness Proofs* [6], Bernhard Beckert, Mihai Herda, Stefan Kobischke, and Mattias Ulbrich introduce the concept of state-space coverage for partial proofs, which estimates to what degree the proof covers the state space and the possible inputs of a program. This concept brings together deductive verification techniques with runtime techniques used to empirically estimate the coverage.

2.3 Topic 3: Putting Combinations of Static and Runtime Verification into Practice

This topic is dedicated to practical (current and future) combinations of static and runtime verification.

In *Generating Inductive Shape Predicates for Runtime Checking and Formal Verification* [9], Jan H. Boockmann, Gerald Lüttgen, and Jan Tobias Mühlberg show how memory safety predicates, statically inferred by a program comprehension tool, can be employed to generate runtime checks for securely communicating dynamic data structures across trust boundaries. They also explore to what extent these predicates can be used within static program verifiers.

In *Runtime Assertion Checking and Static Verification: Collaborative Partners* [20], Fonenantsoa Maurica, David Cok, and Julien Signoles discuss how to achieve, on the architectural level, static and dynamic analysis systems featuring a narrow semantic gap as well as similar levels of sound and complete checking. They also describe designs and implementations that add new capabilities to runtime assertion checking, bringing it closer to the feature coverage of static verification.

In *A Language-Independent Program Verification Framework* [10], Chen and Rosu describe an approach to language-independent deductive verification, using the $\mathbb{K}$ semantics framework. They show how a program verifier as well as other, also dynamic, language tools are generated automatically, from the semantics description, correct-by-construction.

2.4 Panel Discussion

Inspired by the different talks and discussions, the final panel discussion aimed for convergence on matters that lead the way forward, and concrete next steps for the community to achieve the set goals.

References

1. Aceto, L., Francalanza, A., Ingolfsdottir, A.: Proceedings First Workshop on Pre- and Post-Deployment Verification Techniques. ArXiv e-prints, May 2016
2. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M.: Deductive Software Verification-The KeY Book: From Theory to Practice. LNCS, vol. 10001. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49812-6
3. Ahrendt, W., Chimento, J.M., Pace, G.J., Schneider, G.: Verifying data- and control-oriented properties combining static and runtime verification: theory and tools. Form. Methods Syst. Des. **51**(1), 200–265 (2017). https://doi.org/10.1007/s10703-017-0274-y
4. Ahrendt, W., Pace, G.J., Schneider, G.: StaRVOOrS — episode II - strengthen and distribute the force. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 402–415. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_28
5. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 1–33. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_1

6. Beckert, B., Herda, M., Kobischke, S., Ulbrich, M.: Towards a notion of coverage for incomplete program-correctness proofs. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 53–63. Springer, Cham (2018)
7. Bodden, E., Lam, P., Hendren, L.J.: Partially evaluating finite-state runtime monitors ahead of time. ACM Trans. Program. Lang. Syst. **34**(2), 7:1–7:52 (2012). https://doi.org/10.1145/2220365.2220366
8. Bonakdarpour, B., Sánchez, C., Schneider, G.: Monitoring hyperproperties by combining static analysis and runtime verification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 8–27. Springer, Cham (2018)
9. Boockmann, J.H., Lüttgen, G., Mühlberg, J.T.: Generating inductive shape predicates for runtime checking and formal verification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 64–74. Springer, Cham (2018)
10. Chen, X., Rosu, G.: A language-independent program verification framework. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 92–102. Springer, Cham (2018)
11. Cheon, Y., Leavens, G.T.: A Runtime Assertion Checker for the Java Modeling Language (JML) (2002)
12. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.P.: Handbook of Model Checking. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-10575-8
13. Clarke, L.A., Rosenblum, D.S.: A historical perspective on runtime assertion checking in software development. ACM SIGSOFT Softw. Eng. Notes **31**(3), 25–37 (2006)
14. Desai, A., Seshia, S., Qadeer, S.: Programming safe robotics systems: challenges, advances, and opportunities. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 103–119. Springer, Cham (2018)
15. Filliâtre, J.C.: Deductive software verification. Int. J. Softw. Tools Technol. Transf. **13**(5), 397 (2011). https://doi.org/10.1007/s10009-011-0211-0
16. Fisman, D., Kugler, H.: Temporal reasoning on incomplete paths. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 28–52. Springer, Cham (2018)
17. Francalanza, A., Pace, G.J.: Proceedings Second International Workshop on Pre- and Post-Deployment Verification Techniques. ArXiv e-prints, August 2017
18. Gurov, D., Havelund, K., Huisman, M., Monahan, R.: Static and runtime verification, competitors or friends? (Track Summary). In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 397–401. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_27
19. Howar, F., Giannakopoulou, D., Mues, M., Navas, J.: Generating component interfaces by integrating static and symbolic analysis, learning, and runtime monitoring. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 120–136. Springer, Cham (2018)
20. Maurica, F., Cok, D., Signoles, J.: Runtime assertion checking and static verification: collaborative partners. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 75–91. Springer, Cham (2018)
21. Rupprecht, T., Chen, X., White, D.H., Boockmann, J.H., Lüttgen, G., Bos, H.: DSIbin: identifying dynamic data structures in C/C++ binaries. In: Rosu, G., Penta, M.D., Nguyen, T.N. (eds.) Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, 30 October–3 November 2017, pp. 331–341. IEEE Computer Society (2017)