Int J Softw Tools Technol Transfer (2015) 17:121–123
DOI 10.1007/s10009-014-0360-z

INTRODUCTION

Runtime verification: the application perspective

Yliès Falcone · Lenore D. Zuck

Published online: 25 November 2014

Abstract In the past decade, runtime verification (RV) has gained much focus, from both the research community and practitioners. RV combines a set of theories, techniques and tools aiming towards efficient analysis of systems' executions and guaranteeing their correctness using monitoring techniques. Major challenges in RV include characterizing and formally expressing requirements that can be monitored, offering intuitive and concise specification formalisms, and monitoring specifications efficiently for functional and non-functional behavior. Despite the major strides made in recent years, much effort is still needed to make RV an attractive and viable methodology for industrial use and to apply it to wider application domains, such as security, bio-health, power micro-grids. This special issue of STTT proposes extended versions of four papers that have been selected from the runtime verification track at ISoLA 2012 (Margaria and Steffen, Proceedings of the 5th international symposium on leveraging applications of formal methods, verification and validation, 2012).

Keywords Runtime verification · Applications · Statistical model checking · Expressiveness · Efficiency

The work of the L. D. Zuck was funded in part by NSF award CCF-0916438.

Y. Falcone (✉)
Laboratoire d'Informatique de Grenoble,
University of Grenoble I (UJF), Grenoble, France
e-mail: ylies.falcone@ujf-grenoble.fr

L. D. Zuck
University of Illinois at Chicago, Chicago, USA
e-mail: lenore@cs.uic.edu

1 Introduction

Static methods can guarantee program correctness. They are, however, not always applicable to a variety of systems and properties. Often the size of the system renders static methods prohibitively expensive. Systems to which static techniques are applied are those where correctness is to be proven under all circumstances, such as safety critical systems. In contrast, many "real-life" systems may be occasionally faulty, especially when the fault is not catastrophic (or even very expensive) and the system can recover from it. Similarly, static techniques are applicable to systems that are built top down and are often applied at the design stages. In contrast, many "real-life" systems are developed ad hoc so that their properties are not always known à priori, yet, they may be learnt during the systems' execution. For these and many other reasons, RV offers an interesting alternative to static methods.

In the past decade, runtime verification (RV) has gained much focus from both research community and practitioners [4,5,8,9,12,13]. RV combines a set of theories, techniques and tools aiming towards efficient analysis of systems' executions and guaranteeing their correctness using monitoring techniques. While some of the techniques used in RV have been applied in several areas, mainly by the testing community, it had only recently became a first-class citizen in the formal-method community after a 2001 workshop (now a conference), carrying the name *runtime verification*, which was initiated by Klaus Havelund (who authors in this special issue of STTT) and Grigore Rosu.

This issue of STTT presents some new directions in RV. Two papers focus on the exploration of the expressiveness-efficiency spectrum [1], that is, the observed duality between the expressiveness of the specification language and the resource consumption made by runtime monitors. They support the common conjecture that the more expressive the specification formalism is, the more complex (and resource consuming) the associated monitoring algorithm is.

The other two papers focus on *statistical model checking* (see [14]), which augments runtime verification with statistics. When one has access to several executions of the system under scrutiny, statistical model checking allows to complement runtime verification of properties using models of stochastic behavior to model the uncertainty of a system and to better assess the overall correctness of the system by applying statistical inference to reason on several execution traces.

2 The expressiveness/efficiency spectrum

2.1 On piggyback runtime monitoring of object-oriented programs (Hallé et al.)

The work in [6] proposes an original approach to the quest for expressive and efficient runtime monitors. It relies on the observation that when monitoring Java programs, many of the properties on the usage of data structures (usually addressed in benchmarks) are already monitored by the object instances at runtime. More precisely, the authors study the extent that the fields of an object carry information that can be piggybacked by a monitor to take a decision on the satisfaction of the property under verification. The authors also address the general question of how a monitor can use the information in the object's member fields. They empirically evaluate the benefit of piggyback monitoring and highlight the benefits of their approach.

2.2 Rule-based runtime verification revisited (Havelund)

Inspired by the RETE algorithm ([2]), Havelund [7] proposes to revisit rule-based monitoring. While rule-based RV employs somewhat less efficient runtime monitors, it offers for a concise and elegant expression of specifications. In both RV and artificial intelligence, a rule is specified by a collection of facts, or events, that trigger some actions. The paper shows that the RETE algorithm can be made an efficient solution to the event-matching problem, and thus to the problem of (efficiently) dispatching events carrying data value to the related monitor instances. It also shows how to adapt and optimize RETE for monitoring purposes. An implementation of the modified RETE is proposed in the Scala-based tool LOGFIRE, whose performance is compared to state-of-the-art ruled-based systems.

3 Statistical model checking: augmenting runtime verification with statistics

3.1 Statistical model checking QoS properties of systems with SBIP (Nouri et al.)

The work in [11] proposes SBIP—a stochastic extension of the Behavior Interaction Priority (BIP) framework which is an expressive and rigorous component-based design flow for the hierarchical construction of systems. Adding stochastic features to BIP allows to better assess the correctness of the system as well as to model uncertainty stemming from faults or assumptions on the runtime platform. As Nouri et al. [11] shows, SBIP allows to combine the results from different executions with statistical inference algorithms so as to add confidence measurement on the satisfaction of properties. To overcome the restrictions of using statistical model checking, the authors propose to consider properties expressed in Bounded Linear Temporal Logic and to eliminate the non-determinism in the system by randomizing transitions. Two case studies conducted with SBIP are presented.

3.2 Schedulability of Herschel–Planck revisited using statistical model checking (David et al.)

The work in [3] proposes a schedulability analysis of Herschel–Planck satellite system using symbolic model checking and statistical model checking. The paper shows the complementarity of these techniques for proving that a run is either realizable or that a deadline violation exists. It is demonstrated that statistical model checking improves performance analysis by providing response times when a system is schedulable, and probability of deadline violation otherwise.

Acknowledgments We would like to thank the organizing committees of ISoLA 2012 for setting up such a successful event, the programme committee and reviewers of the conference and the special issue of STTT for helping with the selection of papers, and all authors who contributed to the track. We would especially like to thank the authors for providing us with such excellent papers, and the referees for their diligent work.

References

1. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata: towards expressive and efficient runtime monitors. In: FM 2012, 18th International Symposium on Formal Methods, Paris, France, 27–31 August 2012. Lecture Notes in Computer Science, vol. 7436, pp. 65–79 (2012)
2. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule systems for run-time monitoring: from Eagle to RuleR. J. Log. Comput. **20**(3), 675–706 (2010)

3. David, A., Larsen, K.G., Legay, A., Mikučionis, M.: Schedulability of Herschel–Planck revisited using statistical model checking. In: STTT (2014)
4. Falcone, Y., Fernandez, J.-C., Mounier, L.: Runtime verification of safety-progress properties. In: Bensalem, S., Peled, D. (eds.) Runtime Verification, 9th International Workshop, RV 2009, Grenoble, France, 26–28 June 2009. Selected Papers. Lecture Notes in Computer Science, vol. 5779, pp. 40–59. Springer, New York (2009)
5. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Broy, M., Peled, D., Kalus, G. (eds.) Engineering Dependable Software Systems. NATO Science for Peace and Security Series D, vol. 34. Information and Communication Security, pp. 141–175. IOS Press, Amsterdam (2013)
6. Hallé, S., Vallet, J., Tremblay-Lessard, R.: On piggyback runtime monitoring of object-oriented programs. In: STTT (2014)
7. Havelund, K.: Rule-based runtime verification revisited. In: STTT (2014)
8. Havelund, K., Goldberg, A.: Verify your runs. In: Meyer, B., Woodcock, J. (eds.) VSTTE. Lecture Notes in Computer Science, vol. 4171, pp. 374–383. Springer, New York (2005)
9. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Progr. **78**(5), 293–303 (2008)
10. Margaria, T., Steffen, B. (eds.) Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2012, Amirandes, Heraclion, Crete, 15–18 October 2012. Lecture Notes in Computer Science. Springer, New York (2012)
11. Nouri, A., Bensalem, S., Bozga, B.D.M., Jegourel, C.,Legay, A.: Statistical model checking QOS properties of systems with SBIP. In: STTT (2014)
12. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM. Lecture Notes in Computer Science, vol. 4085, pp. 573–586. Springer, New York (2006)
13. Runtime Verification: http://www.runtime-verification.org (2001–2014). Accessed 4 Jan 2014
14. Younes, H.L.S.: Verification and planning for stochastic processes with asynchronous events. PhD thesis, Carnegie Mellon (2005)