Introduction to the Special Issue on Runtime Verification

Oleg Sokolsky
University of Pennsylvania
Philadelphia, USA

Grigore Roşu
University of Illinois,
Urbana-Champaign, USA

Runtime verification is a research area that is concerned with monitoring and dynamic analysis of evolving executions with respect to precisely specified properties. The primary motivation for this field of study, at its inception over a decade ago, was to overcome the scalability limitations of exhaustive design-time formal verification. Traditionally, model checking techniques suffer from state explosion that limits the size of systems that can be verified. Moreover, model checkers operate on models and thus introduce additional proof obligations on the correctness of abstraction or model creation. On the other hand, verification based on theorem proving usually involves significant amount of manual effort that, effectively, limits the size of the system that can be verified. By concentrating on the current execution of the actual system, runtime verification techniques allow us to have automatic analysis that is less dependent on the size of the system and, at the same time, does not require as much abstraction.

Early efforts in the runtime verification community focused on characterizing properties suitable for checking at run time [2], on the generation of efficient monitors for properties specified in a variety of formal languages [7, 8] and on improving efficiency of monitoring by static analysis [3]. These efforts led to the development of mature tools, such as the MOP framework [9], within less than one decade of runtime verification research. A somewhat separate line of research was concerned with "specification-less" monitoring, which targets runtime checking algorithms for a set of common and well-defined problems. Typically, these algorithms are related to concurrency within the system, such as freedom from race conditions [5], atomicity [11], serializability [4], etc.

The runtime verification domain has seen an increased interest over the recent years, generating enough traction for the community to form its own international conference. This special issue contains selected revised papers presented at the First International Conference on Runtime Verification, held in St. Julians, Malta, on November 1–4, 2010. Collectively, the papers represent some of the most relevant current research directions within the runtime verification area.

Reduction of monitoring overhead remains an active research area. In contrast to the general-purpose techniques studied in the initial runtime verification

research, current research concentrates on approaches specific to a particular application domain. Techniques applicable to low-level monitoring of embedded systems [10] and by necessity very different compared to techniques that can be used for the monitoring of web services [6]. This issue includes the paper "Optimized Temporal Monitors for SystemC" by Deian Tabakov, Kristin Rozier, and Moshe Vardi. The authors compare several ways to encode monitors embedded in SystemC designs and identify the most efficient ones in different scenarios through extensive evaluation. This paper will serve as a useful guide to implementors of efficient monitors in resource-constrained software systems.

An important question that remains when a monitoring approach to verification is used, is what happens when a problem is discovered by the checker. Runtime verification systems typically involve the capability of invoking recovery actions in response to property violations. Effectiveness of such recovery actions often depends on the assumption that system execution does not progress beyond the point of violation. That is, that checking is synchronous and the system is effectively stopped while checking is performed. This assumption may have a serious impact on the response time of the system and would be too restrictive in some cases. This issue contains the paper "Safer Asynchronous Runtime Monitoring Using Compensations," by Christian Colombo, Gordon Pace, and Patrick Abela, which relaxes the synchronous checking assumption and aims to impose transaction-like semantics on system executions and explores the means of unrolling the execution back once a violation is discovered. The authors also propose means of switching between synchronous and asynchronous monitoring modes as means of a trade-off between performance and the level of assurance.

Another important direction of research in the runtime verification are is the generation of a trace from the execution of a system. In some cases, traces can be obtained by passive observation; for example, monitoring of bus traffic. However, in the cases when we are checking an execution of a software system, some kind of instrumentation is needed to extract observations. An important observation made in [1] was that construction of a monitor, which has to observe all accesses to an object of interest throughout a system execution, corresponds to an aspect in terms of aspect-oriented programming. This observation has led to significant advances in state of the art. A variety of tools for program instrumentation using aspect-oriented techniques has been proposed, becoming a dominant approach in runtime verification of software systems. This issue includes the paper "InterAspect: Aspect-Oriented Instrumentation with GCC" by Justin Seyster, Ketan Dixit, Xiaowan Huang, Radu Grosu, Klaus Havelund, Scott A. Smolka, Scott D. Stoller, and Erez Zadok, which builds upon a powerful and widely used compiler infrastructure. An important advantage of the framework is the access to static analysis modules within GCC, which can help to reduce overhead of the instrumentation. Combined with the large number of language front-ends supported by GCC, InterAspect provides an instrumentation framework that rivals most of the common instrumentation approaches for high-level languages.

One of the most pervasive research topics in specification-less monitoring is

to devise specialized and efficient algorithms to dynamically detect very specific types of concurrency errors. Data-races remain one of the major types of concurrency errors, which can have catastrophic consequences. For example, they were among the flaws of the Therac-25 radiation therapy machine, which led to the death and serious injuries of several patients. Also, the North American Blackout of 2003 was caused by a race condition. There are many race detector systems developed by the runtime verification and testing communities, but in order to make their use widespread and an integral part of software development cycles, they need to be efficient. In this context, efficiency means low runtime overhead. This issue includes the paper "Efficient Data Race Detection for Async-Finish Parallelism" by Raghavan Raman, Jisheng Zhao, Vivek Sarkar, Martin Vechev, and Eran Yahav, which shows that various kinds of data-races can be detected with an average slowdown of about 3 times. This number is a significant improvement over the current state-of-the art dynamic race detectors, which often add an order of magnitude or more slowdown to the running system.

We hope that this special issue gives a quick, yet technical snapshot of the current research in runtime verification, and that it will encourage the reader to investigate and to contribute to this dynamic and growing field.

References

[1] C. Allan, P. Avgustinov, S. Kuzins, O. de Moor, D. Sereni, G. Sittampalam, J. Tibble, A. S. Christensen, L. Hendren, and O. Lhoták. Adding trace matching with free variables to AspectJ. In *Proceedings of the 20th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA'05)*, pages 345–364, October 2005.

[2] A. Bauer, M. Leucker, and C. Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[3] E. Bodden, L. Hendren, and O. Lhoták. A staged static program analysis to improve the performance of runtime monitoring. In *Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07)*, volume 4609 of *LNCS*, pages 525–549, July 2007.

[4] K. Chen, S. Malik, and P. Patra. Runtime validation of transactional memory systems. In *International Symposium on Quality Electronic Design*, pages 750–756, 2008.

[5] T. Elmas, S. Qadeer, and S. Tasiran. Goldilocks: Efficiently computing the happens-before relation using locksets. In *Proceedings of the Workshop on Formal Approaches to Testing and Runtime Verification (FATES/RV'06)*, August 2006.

[6] S. Hallé, T. Bultan, G. Hughes, M. Alkhalaf, and R. Villemaire. Runtime verification of web service interface contracts. *IEEE Computer*, pages 59–66, March 2010.

[7] K. Havelund and G. Roşu. Synthesizing monitors for safety properties. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 342–356, April 2002.

[8] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan. Java-MaC: a run-time assurance approach for Java programs. *Formal Methods in Systems Design*, 24(2):129–155, March 2004.

[9] P. Meredith, D. Jin, D. Griffith, F. Chen, and G. Roşu. An overview of the MOP runtime verification framework. *Software Tools for Technology Transfer, Special Section on Runtime Verification*, 14(3):249–289, 2011.

[10] L. Pike, A. Goodloe, R. Morisset, and S. Niller. Copilot: A hard real-time runtime monitor. In *International Conference on Runtime Verification (RV 2010)*, volume 6418 of *LNCS*, November 2010.

[11] L. Wang and S. D. Stoller. Runtime analysis of atomicity for multi-threaded programs. *IEEE Transactions on Software Engineering*, 32:93–110, February 2006.