Verifying Execution Traces

Klaus Havelund*
Jet Propulsion Laboratory, California Institute of Technology, USA

Runtime Verification (RV) [8,10] is an approach to modeling and verification of software systems, which focuses on analyzing program executions. An execution can abstractly be considered as represented by an execution trace, a sequence of events. RV is, amongst other things, concerned with checking such traces against formal specifications. Other topics include learning specifications from traces, and even trace visualization. RV covers fundamentally any technique that supports analysis of program (or more generally: systems) executions, for test purposes, program understanding, or for detection of specific behavior during deployment of a system, for example as part of a fault protection strategy. RV is typically not concerned with how such executions are obtained, and is for example not concerned with test input generation. RV can be considered as a research focus on dynamic program analysis.

In this series of lectures we shall focus on checking program executions against formal specifications. A variety of formalisms and algorithms have been proposed for this purpose, specifically over the last decade. These include state machines, temporal logics, regular expressions, grammar systems and rule-based systems. A major research challenge is how to conveniently specify and efficiently monitor properties of traces containing data parameterized events. It turns out that data pose a challenge both wrt. formalisms as well as wrt. algorithms used for checking such data rich properties.

We shall present a set of different data-centric formalisms as well as their associated monitoring algorithms, illustrating the state of the art in RV. Some of these logics are designed in order to make monitoring efficient, often at the cost of expressiveness of the logic. Other logics are designed to be expressive, often at the cost of efficiency. We shall explore both ends of the spectrum, and look at some solutions to bridge the conflict between efficiency and expressiveness. We shall specifically study the following runtime verification systems: JavaMOP [9], TraceMatches [1], Ruler [7], TraceContract [6], and QEA [3]. Other systems [4,5] might be discussed as time allows.

Prototype implementations of the various logics will be shown in the Scala programming language [11]. In addition we shall study how programs can be instrumented for runtime verification using aspect oriented programming, specifically using the AspectJ system [2] for Java.

References

[1] C. Allan, P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, O. Lothák, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. *Adding Trace Matching with Free Variables to AspectJ.* SIGPLAN Not., Vol. 40, pp. 345-364; 2005.

[2] *AspectJ.* `http://www.eclipse.org/aspectj`.

[3] H. Barringer, Y. Falcone, K. Havelund, G. Reger, D. Rydeheard. *Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors.* 2012.

*Part of the work described in this publication was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

[4] H. Barringer, A. Goldberg, K. Havelund, K. Sen. *Rule-based Runtime verification*. In: VMCAI, pp. 44-57; 2004.

[5] H. Barringer, A. Groce, K. Havelund, M. Smith. *Formal Analysis of log files*. Journal of Aerospace Computing, Information, and Communication; 2010.

[6] H. Barringer, K. Havelund. *Tracecontract: a Scala DSL for Trace Analysis*. In: Proc. of the 17th International Conference on Formal Methods, pp. 57-72; 2011.

[7] H. Barringer, D. Rydeheard, K. Havelund. *Rule Systems for Run-time Monitoring: from EAGLE to RuleR*. J Logic Computation, Vol. 20(3); pp. 675-706; 2010.

[8] K. Havelund, A. Goldberg. *Verify your Runs*. In: Verified Software: Theories, Tools, Experiments, VSTTE 2005, pp. 374-383; 2008.

[9] P. Meredith, D. Jin, D. Griffith, F. Chen, G. Roşu. *An Overview of the MOP runtime verification framework*. J Software Tools for Technology Transfer, pp. 1-41; 2011.

[10] *Runtime Verification*. `http://www.runtime-verification.org`; 2001-2012.

[11] *Scala*. `http://www.scala-lang.org`.