Dagstuhl Seminar 07011 "Runtime Verification" January 2–6, 2007

Executive Summary

Bernd Finkbeiner[1], Klaus Havelund[2], Grigore Roşu[3] and Oleg Sokolsky[4]

[1] Saarland University, Reactive Systems Group
Saarbrücken, Germany
finkbeiner@cs.uni-sb.de
[2] NASA's Jet Propulsion Laboratory, Laboratory for Reliable Software
Pasadena, California, USA
Klaus.Havelund@jpl.nasa.gov
[3] University of Illinois at Urbana-Champaign, Formal Systems Laboratory
Urbana, Illinois, USA
grosu@cs.uiuc.edu
[4] University of Pennsylvania, Department of Computer and Information Science
Philadelphia, Pennsylvania, USA
sokolsky@saul.cis.upenn.edu

Abstract. From January 2 to January 6, 2007, the Dagstuhl Seminar 07011 *'Runtime Verification'* was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. Over the past few years, runtime verification has emerged as a focused subject in program analysis that bridges the gap between the complexity-haunted field of fully formal verification methods and the ad-hoc field of testing. Other terms for this subject are: program monitoring, dynamic program analysis, and runtime analysis. Thirty researchers participated in the seminar and discussed their recent work and recent trends in runtime verification.

Keywords. Program monitoring, dynamic program analysis, specification languages and logics, concurrency errors, program instrumentation, aspect-oriented programming, test oracles, fault protection, dynamic specification learning, combining static and dynamic analysis.

The 2007 Dagstuhl Seminar 07011 on *Runtime Verification*[5] was held from Tuesday January 2 to Saturday January 6, 2007. Thirty researchers participated and discussed their recent work and recent trends in runtime verification. Other terms for this subject are: program monitoring, dynamic program analysis, and runtime analysis. Over the past few years, this field has emerged as a focused subject

[5] The website for the seminar:
http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=07011.

Dagstuhl Seminar Proceedings 07011
Runtime Verification
http://drops.dagstuhl.de/opus/volltexte/2008/1369

in program analysis that bridges the gap between the complexity-haunted field of fully formal verification methods and the ad-hoc field of testing. Runtime verification supplements static analysis and formal verification with more light-weight dynamic techniques when the static techniques fail due to complexity issues. From the perspective of testing, runtime verification helps to formalize oracle specification. Runtime verification uses some form of instrumentation to extract, during test or in operation, a trace of observations from a program run and then applies formal verification to this trace. The focus on traces rather than on transition systems is of course what makes the approach more scalable but also less effective at the same time. However, applying rigor and advanced techniques in trace analysis may provide several practical advantages.

The seminar covered several areas, which we shall briefly touch upon. One of the corner stones of this field is the monitoring of program executions (theoretically thought of as traces) against formal specifications, for example represented in temporal logic, regular expressions or state machines. Specification logics can include real-time features enabling the monitoring of real-time properties. Some of the questions that arise in this context are the following: what expressive power is required of a monitoring logic; what characteristics should it have in order to make monitoring efficient with as little impact on the running program as possible; and what characteristics will make such a logic easy and attractive to use from a user's point of view. The two first questions are specific to runtime verification whereas the latter is of general interest to any formal method.

In order to monitor a program (or more generally: a system), the program (system) must be instrumented to feed the monitor. This can happen by instrumenting the program to generate a trace in a log-file, which can be analyzed off-line, or the program can be instrumented to drive the monitor directly during execution, in which case errors are detected immediately as they occur. Aspect oriented programming is an example of a technology for performing program instrumentation. An interesting trend is the concept of state-full aspects, which essentially extend the point-cut language of aspect oriented programming to temporal predicates over the execution trace. In this view an aspect advice consists of a temporal trace predicate and a statement to be executed when this predicate gets violated during a program execution. This approach can be seen as combining aspect oriented programming with runtime verification. The execution of repair code when a property gets violated is an example of a fault protection strategy. This leads into a paradigm for programming where programs are not expected to behave correctly and where a program is embedded in a protection armor, providing error diagnosis and recovery.

The formalization of properties in terms of specifications requires human effort, which is known to cause resistance. A branch of the field attempts to perform dynamic analysis in the absence of human-provided formal specifications. There are two variants of this work. In the first variant algorithms are pre-programmed that analyze for specific generic kinds of errors that are generally regarded as problems in any application. Examples are concurrency errors such as data races and deadlocks. The second variant, dynamic specification

learning, consists of learning specifications from runs. Each run that is accepted by a user is regarded as contributing to a nominal behavior specification of the program. After a period such a nominal behavioral specification can be turned into an oracle used to detect deviations.

An important topic is the interaction between static and dynamic analysis. Static analysis can be used to minimize the impact of monitoring a program by for example reducing the number of program points where the program needs to interact with the monitor. A dual view of this interaction between static and dynamic analysis is to regard dynamic analysis as a rescue plan when static analysis cannot determine whether a program satisfies a particular property. It may for example be the case that a property can be proved about a program, but only under the assumption of a set of proof-obligations (lemmas), each of which can then be dynamically monitored during test runs or during operation.

The field of runtime verification overlaps with the field of testing from the perspective of test oracles. Often, a monitor for a formally specified property can be used to evaluate whether a test execution has been successful. However, runtime verification is less concerned with the test case generation aspect of testing, where the goal is to drive the program into all its corners. Runtime verification focuses on analyzing or collecting information from individual runs, independently of how they have been obtained. The Dagstuhl event included contributions from the testing field on topics such as test case generation, fault injection, and unit testing. These contributions explored the relationship between the fields of testing and runtime verification.