StaRVOOrS: Unifying Static and Runtime Verification of Java

Wolfgang Ahrendt[1], Jesús Mauricio Chimento[1], Gordon Pace[2] and Gerardo Schneider[3]

[1] Chalmers University of Technology, Sweden.
ahrendt@chalmers.se, chimento@chalmers.se
[2] University of Malta, Malta.
gordon.pace@um.edu.mt
[3] University of Gothenburg, Sweden.
gerardo@cse.gu.se

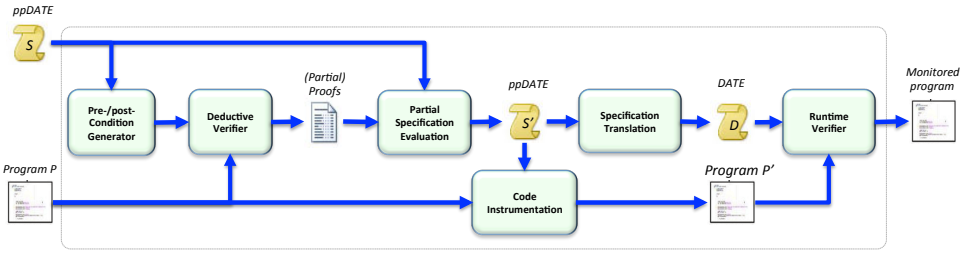
1 Introduction

Over the past decades, several forms of automated verification techniques have been proposed and explored in the literature. These techniques mostly fall in one of two categories: static and dynamic verification. *Runtime verification* is a dynamic technique concerned with the monitoring of software, providing guarantees that observed runs comply with specified properties. It is strong in analysing systems of a complexity that is difficult to address by *static verification*, like systems with numerous interacting sub-units, heavy usage of mainstream libraries, real (as opposed to abstract) data, and real world deployments. On the other hand, the major drawbacks of runtime verification are the impossibility to extrapolate correct observations to all possible executions, and that monitoring introduces runtime overheads. In the work we present here, these issues are addressed by combining runtime verification with static verification, such that: (i) Static verification attempts to 'resolve' those parts of the properties which can be confirmed statically; (ii) the static results, even if only partial, are used to change the property specification such that generated monitors will not check dynamically what was confirmed statically.

In addition to combining static and runtime verification, we introduce the specification language *ppDATE* [3], and a verification tool which embodies this approach, called StaRVOOrS [7], which captures both *control-oriented* properties (like the *DATE* language used in the runtime verification tool Larva [9]) and *data-oriented* properties (like the *Java Modelling Language* JML [6, 10]).

2 The StaRVOOrS Framework

Below we show an abstract view of the framework, which was initially sketched in [4].



Given a Java program P and a specification π of the properties to be verified (given in the language *ppDATE*, see Sec. 3), these are transformed into suitable input for the deductive verifier KeY [5] which attempts to statically verify the properties related to pre/post-conditions. What is not proved statically will then be left to be proven at runtime. Here, not only the completed but also the *partial* proofs will be used by the *Partial Specification Evaluator* module in order to rewrite the original π into π' which triggers runtime checks for the parts which were not possible to prove statically. To achieve this, the original pre-conditions from π are refined to express also path conditions for not statically verified executions.

The *ppDATE* specification π' is then translated into a specification D, written in the *DATE* formalism [8] — a formalism suitable for the runtime verifier Larva [9]. As *DATE* does not handle pre/post-conditions, these are simulated by pure *DATE* concepts. This also requires changes to the code base (done by the *Code Instrumentation* module), like adding counters to distinguish different executions of the same code unit, or adding methods which operationalise pre/post-condition evaluation. The instrumented program P' and the *DATE* specification D are passed on to the runtime verification tool Larva, which uses aspect-oriented programming techniques to capture relevant system events and monitors, thus producing a monitored program, essentially equivalent to running the original program in parallel with a monitor of the original property, albeit more efficiently.

3 A Specification Language for Static and Runtime Verification of Data and Control Properties

StaRVOOrS uses *ppDATE* as the input language for its properties, which enables the combination of data- and control-based properties in a single formalism. *ppDATE*s are a composition of the control-flow language *DATE*, and of data-oriented specifications in the form of Hoare triples with *p*re-/*p*ost-conditions expressed using JML boolean expression syntax [10], which is designed to be easily usable by Java programmers. The data-oriented features of the specification appear in the states. A state may have a number of Hoare triples assigned to it. Intuitively, if Hoare triple $\{\pi\}\texttt{f}\{\pi'\}$ appears in state q, the property ensures that: if the system enters code block `f` while the monitor lies in state q and precondition π holds, upon reaching the corresponding exit from `f`, postcondition π' should hold. To ensure efficient execution of monitors, *ppDATE*s are assumed to be deterministic by giving an ordering in which transitions are executed.

For a full and detail description of *ppDATE*, refer to [3].

4 StaRVOOrS Tool

The tool is a fully automated implementation of the theoretical results presented in [3, 4]. Given a property specification and the original Java program, our tool chain produces a statically optimised monitor and the weaved Java program to be monitored. This includes the automated triggering of numerous verification attempts of the underlying static verification tool, the analyses of resulting partial proofs, and the monitor generation.

The tool works following these steps: (1) A property is written using our script language for *ppDATE*; (2) Hoare triples are extracted from the specification of the property, are translated into JML contracts to be added to the Java files; (3) KeY attempts to verify all JML contracts, generating (partial) proofs, the analysis of which results in an XML file; (4) The *ppDATE* is refined based on the XML file; (5) Declarative pre/post-conditions are operationalised; (6) The code is instrumented with auxiliary information for the runtime verifier; (7) The *ppDATE*

specification is encoded into *DATE*s; (8) The LARVA compiler generates a runtime monitor. See [7] for more details about the tool.

5 Conclusion

In [3] we have formalised the language for combining (partial) static and (optimised) runtime verification, which we called *ppDATE*, we have introduced an algorithm to transform a *ppDATE* specification in a *DATE* specification (the input language of the runtime verifier LARVA). In [7] we have introduced the fully automated tool STARVOORS, which implements the theoretical results presented in [3, 4]. StaRVOOrs combines the deductive theorem prover KeY and the runtime verification tool LARVA, and uses properties written using the *ppDATE* specification language. In addition, we have demonstrated the effectiveness of the tool by applying it to *Mondex* [1], an electronic purse application for smart cards products.

At the moment we are working on the proof of soundness of the *ppDATE* transformation algorithm introduced in [3] and we are analysing a new and larger case study based on *SoftSlate* [2], a full-featured, high-performance, open-source Java shopping cart that powers various of e-commerce websites.

Our future work includes: (i) improving the automation of the 'operationalisation' of pre/post-conditions containing algorithmic content; (ii) introduction of a mechanism to deal with the runtime verification of private information; (iii) development of an analyser for the output produced by the monitors generated by STARVOORS.

References

[1] MasterCard International Inc. Mondex. `www.mondexusa.com`.

[2] SoftSlate. `www.softslate.com`.

[3] Wolfgang Ahrendt, Jesús Mauricio Chimento, Gordon J. Pace, and Gerardo Schneider. A specification language for static and runtime verification of data and control properties. In *FM'15*, volume 9109 of *LNCS*, pages 108–125. Springer, 2015.

[4] Wolfgang Ahrendt, Gordon Pace, and Gerardo Schneider. A Unified Approach for Static and Runtime Verification: Framework and Applications. In *ISOLA'12*, LNCS 7609, pages 312–326. 2012.

[5] Bernhard Beckert, Reiner Hähnle, and Peter Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *LNCS*. Springer, 2007.

[6] Yoonsik Cheon and Gary T. Leavens. A runtime assertion checker for the Java Modeling Language (JML). In *SERP'02*, pages 322–328. CSREA Press, 2002.

[7] Jesús Mauricio Chimento, Wolfgang Ahrendt, Gordon Pace, and Gerardo Schneider. StaRVOOrS: A tool for combined static and runtime verification of Java. In *RV'15*, LNCS. Springer, 2015. To appear. Available online at `http://www.cse.chalmers.se/~chimento/starvoors/publications.html`.

[8] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Dynamic Event-Based Runtime Monitoring of Real-Time and Contextual Properties. In *FMICS'08*, volume 5596 of *LNCS*, pages 135–149. Springer-Verlag, September 2009.

[9] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. LARVA - A Tool for Runtime Monitoring of Java Programs. In *SEFM'09*, pages 33–37. IEEE Computer Society, 2009.

[10] Gary T. Leavens, Erik Poll, Curtis Clifton, Yoonsik Cheon, Clyde Ruby, David Cok, Peter Müller, Joseph Kiniry, and Patrice Chalin. *JML Reference Manual. Draft 1.200*, 2007.