KeY + LARVA = STARVOORS

Wolfgang Ahrendt
Jesús Mauricio Chimento
and Gerardo Schneider
Chalmers University of Technology, Sweden

Gordon J. Pace
Department of Computer Science
University of Malta, Malta

***Abstract*—Over the past decades, various forms of automated verification techniques have been proposed and explored in the literature, mostly falling in one of two categories — static and dynamic verification. On one hand, static verification techniques have the ability to verify properties across all possible executions of a system, but fully automated verification is typically not easy to perform. On the other hand, dynamic verification techniques, such as runtime verification, can only give feedback on single executions of the system, but are easy to automate. However, despite the fact that monitoring is typically easy to automate, its adoption in industry is limited — primarily due to the overheads in the system execution that such monitoring typically induces. In this paper, we explore a novel approach in which we combine the two approaches — using static analysis to prune parts of the specification, thus reducing the overheads for the dynamic verification process. We present our framework STARVOORS, which embodies this approach by combining the static analysis tool KeY and the dynamic verification tool LARVA, and discuss how it has been applied to the Mondex case study — an electronic purse implementation. The results presented here have been accepted for publication in [1] and builds upon our previous results from [2], [3].**

I. INTRODUCTION

Runtime verification is concerned with the monitoring of software, providing guarantees that observed runs comply with specified properties. Such methods are 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 address these downsides 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 with low overhead (or fully automatically); (ii) the static results, even if only partial, are used to simplify the property specification such that generated monitors will not check dynamically what was confirmed statically.

As observed, static and dynamic verification have largely disjoint strengths — whereas the former excels in data-oriented properties and struggles to handle complex control-flow logic, the latter handles control-flow properties with substantially lower overheads than data-oriented ones. Combining the two approaches can thus allow the verification process to deal with richer properties with greater ease. However, one of the challenges is to identify a specification notation in which properties which refer to both the data- and control-flow of a system can not only be expressed, but also decomposed to ensure applicability of the different verification techniques.

In this paper, we present our ongoing work on building a framework combining the two approaches, in particular using the KeY [4] static analysis tool and LARVA [5]runtime verification tool. Given the difference in specification styles between the tools, we have also developed a specification language that captures both *control-oriented* properties (like the *DATE* language used in the runtime verification tool LARVA) and *data-oriented* properties (like the *Java Modelling Language* JML [6], [7]). In order to evaluate the proposed framework, we have applied our approach to the Mondex case study[8], an electronic purse application, which provides evidence that our approach substantially reduces the overheads on the runtime monitoring through the use of KeY.

The results reported here are the ones already reported in our published, and soon to appear works [3], [2], [1].

II. STARVOORS

The STARVOORS framework (STAtic and Runtime Verification of Object-ORiented Software), originally proposed in [3], combines the use of the deductive source code verifier KeY [4] with that of the runtime monitoring tool LARVA [5]. KeY is a deductive verification system for data-centric *functional correctness* properties of Java source code, which generates, from JML and Java, proof obligations in *dynamic logic* (a modal logic for reasoning about programs) and attempts to prove them. LARVA (*Logical Automata for Runtime Verification and Analysis*) [5] is an automata-based runtime verification tool for Java programs which automatically generates a runtime monitor from a property using an automaton-based specification notation *DATE*. LARVA transforms the specification into monitoring code together with AspectJ code to link the system with the monitors.

Fig. 1 gives an abstract view of the framework workflow. The framework starts with (i) a Java program P and (ii) a specification S of the properties to be verified written as a *ppDATE*— a specification language which we have developed to combine *DATE*s and pre/post-conditions. The pre-/post-conditions are extracted from S and verified for program P using the deductive verifier KeY. This verification procedure might, in principle, statically fully verify the properties related to pre/post-conditions. Typically, however, parts of the specification cannot be proved automatically — producing partial proofs which are used to simplify the specification S, and producing the simpler set of properties S', which trigger runtime checks only for executions which are not covered by

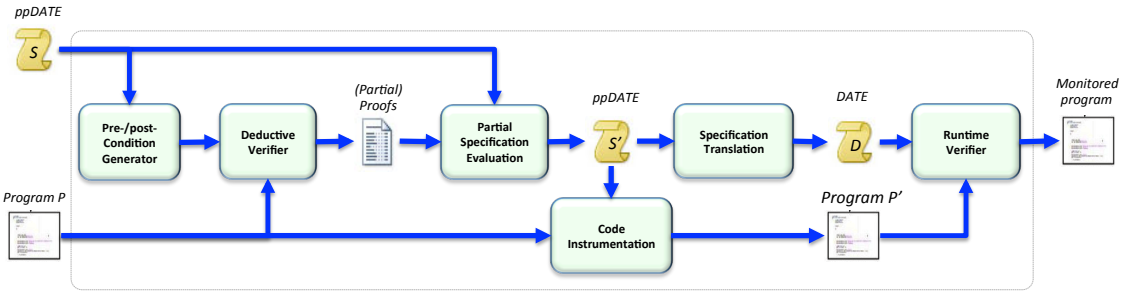

Fig. 1. StaRVOOrS workflow

the static verification. In order to achieve this, the original pre-conditions from S are refined to include path conditions for not statically verified executions. The *ppDATE* specification S' is then translated into a specification D, written in the *DATE* formalism — a formalism suitable for the runtime verifier Larva. As *DATE* has no native support for 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.

The framework has been implemented and is available from [9]. In order to evaluate how effective the reductions due to the static analysis are, it was applied to Mondex, a benchmark problem from the Verified Software Grand Challenge [8]. Mondex is an electronic purse application used by smart cards products [10]. Mondex's original sanitised specification written in Z, together with hand-written proofs of different properties, can be found in [11]. Our variant is strongly inspired by a JML formalisation given in [12]. However, using *ppDATE*, we could more naturally represent the major status of an observer as automata states (rather than in additional data). This also allowed us to naturally differentiate the legal behaviour of code units after the observer's status.

From experimental results (see [1] for further details), we have shown that monitoring overheads are substantially reduced through the use of our approach. The relative difference is quite remarkable: at least 10 times faster for low number throughput of transactions, and increasing up to 900 times faster as the transaction throughput increases. This large reduction in execution time overheads when optimising the monitor is primarily due to the fact that data-oriented monitoring can be prohibitively expensive in the first place. In fact, using our approach, each function with a satisfied precondition fires an additional automaton being traversed in parallel. This results in the large overheads in the case study. However, by pruning away many of these checks through the typical case of a strengthening of the precondition results in the gains we obtain. This indicates that using static analysis to pare down the data-oriented aspect of the properties is ideal in this situation, in that we are attacking directly the overlap between a strength of static analysis and a weakness of dynamic analysis.

III. Conclusions

In this paper we have presented the StaRVOOrS framework combining KeY and Larva, even if it can also be instantiated by other static and dynamic verifiers. Through the use of *ppDATE*, which allow us to combine control- and data-oriented properties in a single formalism, the specifications can be handled by the different tools. The difference in performance between the fully monitored and the version with simplified monitors is, in itself, motivation to look further into how we can extend our approach. We are currently proving the soundness of the transformation of *ppDATE*s to *DATE*s.

References

[1] J. M. Chimento, W. Ahrendt, G. J. Pace, and G. Schneider, "A specification language for static and runtime verification of data and control properties," in *Formal Methods — 20th International Symposium (FM 2015)*, vol. LNCS, 2015.

[2] W. Ahrendt, J. M. Chimento, G. J. Pace, and G. Schneider, "Starvoors: A framework for unified static and runtime verification of object-oriented software," 2014.

[3] W. Ahrendt, G. J. Pace, and G. Schneider, "A unified approach for static and runtime verification: Framework and applications," in *ISOLA '12*, ser. LNCS 7609, 2012.

[4] B. Beckert, R. Hähnle, and P. Schmitt, Eds., *Verification of Object-Oriented Software: The KeY Approach*, ser. LNCS. Springer-Verlag, 2007, vol. 4334.

[5] C. Colombo, G. J. Pace, and G. Schneider, "Larva - a tool for runtime monitoring of java programs," in *SEFM'09*. IEEE Computer Society, 2009, pp. 33–37.

[6] Y. Cheon and G. T. Leavens, "A runtime assertion checker for the Java Modeling Language (JML)," in *SERP'02*. CSREA Press, 2002, pp. 322–328. [Online]. Available: ftp://ftp.cs.iastate.edu/pub/techreports/TR02-05/TR.pdf

[7] G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Müller, J. Kiniry, and P. Chalin, *JML Reference Manual. Draft 1.200*, 2007.

[8] J. Woodcock, "First steps in the verified software grand challenge." in *SEW*. IEEE Computer Society, 2006, pp. 203–206.

[9] "StaRVOOrS," www.cse.chalmers.se/ chimento/starvoors/files.html.

[10] "MasterCard International Inc. Mondex," www.mondexusa.com/.

[11] J. W. S. Stepney, D. Cooper, "An electronic purse: Specification, refinement, and proof," *Technical monograph PRG-126, Oxford University Computing Laboratory*, 2000.

[12] I. Tonin, "Verifying the Mondex case study. The KeY approach," *Technical Report 2007-4, Universität Karlsruhe*, 2007.