Automatic Optimizations for Runtime Verification Specifications

JAN BAUMEISTER, CISPA Helmholtz Center for Information Security, Germany
BERND FINKBEINER, CISPA Helmholtz Center for Information Security, Germany
MATTHIS KRUSE, Saarland University, Germany
STEFAN OSWALD, CISPA Helmholtz Center for Information Security, Germany
NOEMI PASSING, CISPA Helmholtz Center for Information Security, Germany
MAXIMILIAN SCHWENGER, CISPA Helmholtz Center for Information Security, Germany

Automatically transforming program code to yield binaries optimized for execution speed is a heavily researched topic. Research on program transformations mainly focuses on conventional, imperative programming languages. Formal specification languages are easier to analyze due to their formal core, restrictiveness and freedom of side effects. This talk reports on recent work on how to best make use of these properties, exemplified on the stream-based runtime monitoring specification language RTLola. Since a specification is part of a safety-critical component, it should not forsake legibility for efficiency. At the same time, if the monitor will be deployed on an embedded system, reducing its resource footprint is essential. To this end, the RTLola framework employs transformations on several levels when synthesizing executable code from a specification. The first set of transformations work on an intermediate representation of the specification itself. The second one concerns code generation for high-level monitor code, and lastly the compilation of the high-level code into an executable employs conventional compiler transformations.

Additional Key Words and Phrases: Runtime Verification, Stream Monitoring, Specification Languages

Runtime monitors are deployed to increase the safety of and confidence into a cyber-physical system. If the monitor is synthesized from a formal specification, the process has to satisfy two major criteria. First, the underlying specification has to be legible, which reduces the chance of specification errors and improves understandability for third parties. Secondly, the resulting monitor needs to be able to operate in the low-resource environment of an embedded system.

Legibility and performance are often contrary, so it is desirable to transform a legible specification into a performant one automatically. Simple and naive solutions tend to be easiest to grasp and hence perfect in a safety-critical context. However, they often consume more memory or require more processing power than strictly necessary. A solution is to apply automatic transformations to a legible specification, leaving the semantics provably intact while benefiting the performance of the resulting monitor.

In this talk, we are going to report on how recent work employs this idea in the RTLola framework. In a nutshell, the framework synthesizes an executable monitor for a given specification. The underlying eponymous stream-based specification language contains information on how to refine incoming data, *input streams*, into output streams. It also contains criteria for when the refined data indicates a potentially unsafe situation. The synthesis of the executable monitor is a multi-step process as outlined in Figure 1. Here, each step can be subject to performance-enhancing transformations.

The first step is the transformation of a linear specification into a tree-like intermediate representation (IR). Transformations enhancing the IR are particularly effective as they have an immediate effect on any further steps. Hence,

This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center Foundations of Perspicuous Software Systems (TRR 248, 389792660), by the European Research Council (ERC) Grant OSARES (No. 683300), and by the Aviation Research Programm LuFo of the German Federal Ministry for Economic Affairs and Energy as part of "Volocopter Sicherheits-Technologie zur robusten eVTOL Flugzustands-Absicherung durch formales Monitoring" (No. 20Q1963C)..

Baumeister et al. [1] presented a collection of IR transformations. Apart from conventional techniques such as constant folding, there are transformations exclusive to RTLola or stream-based languages. The *Pacing Type Refinement (PTR)* considers the pacing types of each stream. This type dictates under which circumstances a stream is supposed to be re-evaluated. The PTR determines if any pacing type leads to evaluations that have no effect on the output behavior of the monitor. It then refines the type such that the monitor does not engage in unnecessary computations. An empirical evaluation showed that this transformation can boost performance by up to 200% for a specification monitoring an autonomous drone.

The next step is to compile the IR into high-level program code such as Rust. The compilation presented by Finkbeiner et al. [3] fulfills two secondary goals. First, it injects annotations which allows for verifying the correctness of the monitor with respect to the semantics of the underlying specification. Secondly, the generated code reduces the number of conditional statements and memory accesses. For this, an analysis of the specification determines three phases of the monitor execution, the prefix, loop, and postfix. A naive implementation of the former and latter contains conditionals with pre-determined outcome. Their identification is relatively simple on basis of the IR, but complex and non-local in the resulting Rust code. Lastly, the generated code is compiled into an executable using all conventional transformations of the Rust compiler.

When using interpretation [2] rather than compilation, the interpreter also benefits from IR and Rust compiler transformations. However, since the IR is part of the data fragment rather than the actual code, the interpreter lacks transformations depending on the particular input specification, and requires many more memory accesses. Hence, the performance of the compiled version is far superior to an interpretation: it runs 24 times faster for a specification monitoring a network and 73 times faster when monitoring an autonomous drone.

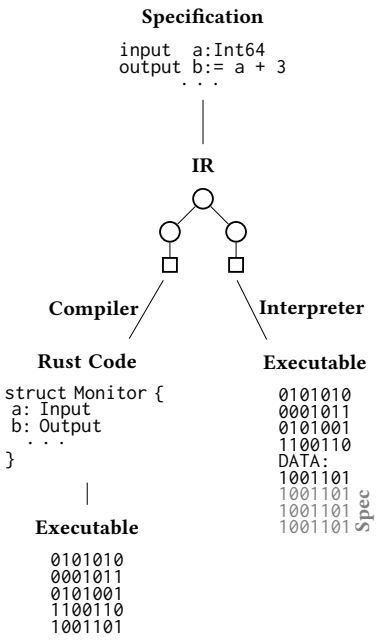

Fig. 1. Overview over the multi-step process synthesizing an executable monitor from a formal specification

Hence, we advocate for development and employment of transformation techniques specific to stream-based specification languages. Their formal core enables a variety of elaborate analyses resulting in huge performance gains. This further increases applicability in environments where resource consumption such as memory requirement, running time, and power consumption is a crucial factor.

REFERENCES

[1] Jan Baumeister, Bernd Finkbeiner, Matthis Kruse, and Maximilian Schwenger. 2020. Automatic Optimizations for Stream-Based Monitoring Languages. In *RV 2020 (LNCS)*, Jyotirmoy Deshmukh and Dejan Nickovic (Eds.), Vol. 12399. Springer, 451–461. https://doi.org/10.1007/978-3-030-60508-7_25

[2] Peter Faymonville, Bernd Finkbeiner, Malte Schledjewski, Maximilian Schwenger, Marvin Stenger, Leander Tentrup, and Hazem Torfah. 2019. StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In *CAV 2019, Part I (LNCS)*, Isil Dillig and Serdar Tasiran (Eds.), Vol. 11561. Springer, 421–431. https://doi.org/10.1007/978-3-030-25540-4_24

[3] Bernd Finkbeiner, Stefan Oswald, Noemi Passing, and Maximilian Schwenger. 2020. Verified Rust Monitors for Lola Specifications. In *RV 2020 (LNCS)*, Jyotirmoy Deshmukh and Dejan Nickovic (Eds.), Vol. 12399. Springer, 431–450. https://doi.org/10.1007/978-3-030-60508-7_24