International Journal on Software Tools for Technology Transfer (2021) 23:155–156
https://doi.org/10.1007/s10009-021-00610-6

GENERAL

Special Issue: RV 2018

Preface

Martin Leucker[1] · Christian Colombo[2]

Accepted: 11 January 2021 / Published online: 1 February 2021

One of the driving forces for introducing runtime verification was the difficulties in performing model checking on large systems. Faced with the so-called state-space explosion problem when trying to check all sequences of an underlying system, runtime verification considers (only) the actual execution or a finite set of recorded executions.

Based on ideas from model checking, it comes natural that the specification languages used in runtime verification are linear-time temporal logic or variations thereof. These have the limitation that only atomic propositions along the state sequences can be specified and consequently checked. For practical applications, however, it is desirable that one can reason about data and its changes during the execution. This has led to the development of richer specification language formalism allowing to deal with data and computations.

In this special issue, we have three papers falling into this category. The first paper is titled Stream Runtime Verification of Real-Time Event-Streams with the Striver Language, written by Felipe Gorostiaga and César Sánchez: Stream runtime verification has been active within the runtime verification community at least since the seminal work on LOLA in 2005, with other languages including TeSSLa and STL. The appeal of this area is that over and above the simplicity and economy of available operators, the reasoning about time is kept separate from computation of data values. This paper proposes a version of Striver, which building on these principles, generalises the time assumption to cover real-time event streams. Effectively, the existing languages mentioned above are shown to be translatable into Striver, which ships with an implementation whose empirical evaluation is reported in the paper.

✉ Christian Colombo
christian.colombo@um.edu.mt

Martin Leucker
leucker@isp.uni-luebeck.de

1 University of Lübeck, Lübeck, Germany

2 University of Malta, Msida, Malta

The second paper in this special issue is written by Joshua Schneider, David Basin, Frederik Brix, Srdan Krstic, and Dimitry Traytel and titled as Scalable Online First-Order Monitoring. It considers runtime verification with data using a first-order logic based approach. While the authors have developed an underlying formal framework over the past decade, the current paper deals with speeding up the verification process by parallelization. To this end, they use slicing techniques to obtain sub-streams that can be monitored independently. As the technique may lead to data duplication, hash-based partitioning techniques used in databases are adapted to and applied in their setting. The authors implement their slicing approach based on Apache Flink and the two RV tools MonPoly and DejaVu, showing that substantial scalability improvements for both tools can be obtained.

The third paper From Parametric Trace Slicing to Rule Systems by Giles Reger and David Rydeheard also looks at parametric runtime monitoring, focusing on the correspondence between the automata-based and rule-based flavours. This involves a Scala implementation of the translation of parametric trace-slicing-based Quantified Event Automata into a rule-based in the style of RuleR, a proof of correctness, and an optimisation based on a notion of redundancy observed during the development of the translation.

Whenever a model of the underlying system is given, one may trace an execution of the system within the model, allowing to check if or to predict when an error has occurred. If the underlying system is only partially observable, such tracing can only give an approximate answer, in saying that a fault must, may be, or may not have occurred. Within the field of diagnosis, one aims at constructing such a diagnoser that identifies the current state (approximately) of an underlying system within a given model. For example, for finite-state models, a diagnoser can be built by keeping track of all possible states that can be reached after each observable step of the system. This set can then answer the question if a failure state is, may be, or is not, reached and computing the reachable states from this set of states in the model may be used to predict the same answer for the ongoing execution of the underlying system.

The paper Diagnosing timed automata using timed markings, written by Patricia Bouyer, Léo Henry, Samy Jaziri, Thierry Jéron, and Nicolas Markey studies the diagnosis problem within the setting of timed systems, which typically have an infinite state space. More precisely, they study the problem for one clock timed automata (as model) and introduce timed markings to keep track of the configurations reachable over time. They show how timed markings can be used to efficiently represent the closure under silent transitions of such automata. They report on their implementation of this approach, compare it to the seminal work by Tripakis on Fault diagnosis for timed automata (2002), and discuss possible generalizations to n-clock timed automata.

The runtime verification community has been actively developing tools for since its inception, some 20 years ago. Going forward, it is crucial to look back and be aware of the vast experience accumulated with the available tools. Therefore, we think it is appropriate to conclude this special issue with the paper A Taxonomy for Classifying Runtime Verification Tools by Yliès Falcone, Srđan Krstić, Giles Reger, and Dmitriy Traytel, which provides an in depth classification across several dimensions of 60 state-of-the-art tools!