Software and Systems Modeling

Software Engineering and Formal Methods

SEFM 2019 Special Section

Peter Csaba Ölveczky · Gwen Salaün

This special section of *Software and Systems Modeling* contains extended versions of selected papers from the 17th International Conference on Software Engineering and Formal Methods (SEFM), which was held in Oslo, Norway, in September 2019. SEFM 2019 was the seventeenth edition of an annual series of conferences that aims at bringing together leading researchers and practitioners from academia and industry to advance the state of the art in formal methods, to facilitate their uptake in the software industry, and to encourage their integration within practical software engineering methods and tools.

SEFM 2019 received 89 paper submissions, from which the program committee accepted 27 papers for inclusion in the proceedings of SEFM 2019 that was published as volume 11724 in Springer's Lecture Notes in Computer Science series. The conference also featured invited talks by Wil van der Aalst, David Basin, and Koushik Sen. Out of those 27 accepted papers, we invited the authors of eight papers to submit to this special section. After an extensive and rigorous reviewing process, in which each paper was reviewed by at least three expert reviewers, we decided to include six of them in this special section.

With the increasing use of machine learning methods in critical applications, such as self-driving cars, there is a need to reason formally about machine learning. The paper "An epistemic approach to the formal specification of statistical machine learning" by Yusuke Kawamoto presents a logical formalization of statistical properties of machine learning. The author proposes a formalization of supervised learning models and test datasets using a distributional Kripke model. An extension of statistical epistemic logic is then introduced as a formal language to describe various properties of machine learning models. This formalization also shows some relationships among properties of classifiers, such as different levels of robustness, and relationships between classification performance and robustness.

P. C. Ölveczky
Department of Informatics, University of Oslo
E-mail: peterol@ifi.uio.no

G. Salaün
University Grenoble Alpes
E-mail: gwen.salaun@univ-grenoble-alpes.fr

Session types are a syntax-based approach and behavioral contracts are an operational approach for describing the communication behavior of processes. The correspondence between these two approaches has previously been studied for synchronous communication. In their paper "Asynchronous session subtyping as communicating automata refinement," Mario Bravetti and Gianluigi Zavattaro study the relationship between session types and behavioral contracts, formalized as communicating finite state machines (CFSMs), when processes communicate *asynchronously.* This paper provides a new theory of asynchronous behavioral contracts that coincide with CFSMs and includes the notions of contract compliance and contract refinement. Bravetti and Zavattaro show under what conditions refinement coincides with asynchronous session subtyping, and also provide an operational interpretation of session types and asynchronous subtyping thanks to a mapping to behavioral contracts and refinement.

Runtime verification is a lightweight formal method which aims to check whether the current system execution satisfies a property that is being monitored. Monitorability deals with classifying which properties can be monitored by observing finite (prefixes) of an execution. There are a number of different notions and definitions of monitorability, which differ in their specification formalisms, operational models, and semantic domains. In their paper "An operational guide to monitorability with applications to regular properties," Adrian Francalanza, Luca Aceto, Antonis Achilleos, Anna Ingolfsdottir, and Karoliina Lehtinen therefore propose a unified, operational view on existing notions of monitorability. In particular, they study different notions of monitorability, define them in an uniform operational view, and characterize these monitorability notions with their corresponding monitorable syntactic fragments of recursive Hennessy-Milner logic.

Verifying, say, C and C++ programs which interact heavily with their execution environments (e.g., operating system) is hard, also because results may not be reproducible due to the side effects of running the program in the first place. The paper "Reproducible execution of POSIX programs with DiOS" by Petr Rockai, Zuzana Baranová, Jan Mrázek, Katarína Kejstová, and Jiri Barnat therefore presents DiOS, a POSIX-compatible operating system designed to offer reproducible execution, with special focus on applications in program verification. DiOS is modular, extensible, and implemented mostly in portable C and C++, although its primary platform is DiVM, a verification-oriented virtual machine. DiOS was evaluated in different ways, as a component of a program verification platform based on DiVM, by combining it with the symbolic executor KLEE, and as a standalone user-mode kernel.

In model-based testing, a model describing the intended behavior of the system is used to guide the testing, to systematically explore and test the system's states. When the model is nondeterministic, it may be impossible to know which states in the model have been visited by a series of tests. Calculating the probability of covering a given coverage goal (where each nondeterministic choice in the model is annotated with probabilities) is the topic of the paper "Test model coverage analysis under uncertainty" by Wishnu Prasetya and Rick Klomp, which introduces a notion of probabilistic coverage to express the coverage of a test suite in this setting. This paper also presents an algorithm to efficiently calculate the probabilistic coverage of both aggregate and non-aggregate coverage goals. Experiments show that in most cases the algorithm is very efficient compared to the brute force approach.

Model-based testing is also the topic of the final paper in this special section. A mutant is a small modification of the description of the system under test, and the aim of mutation-based test generation is to construct tests that reveal these modifications. The goal of the paper "Mutation testing with hyperproperties" by Andreas Fellner, Georg Weissenbacher, and Mitra Tabaei Befrouei is to automatically generate high quality test suites, for which they present a method by solving mutation-based test generation via hyperproperty model checking, where hyperproperties allow us to express properties over multiple executions. The authors formalize several notions of mutation killing for both deterministic and nondeterministic models. In addition to reusing an existing model checking tool to generate test cases for hyperproperties, the authors also propose an alternative approach to obtain such test cases for nondeterministic models.

Acknowledgments We would like to express our most sincere thanks to all the people who have made this special section possible: to the authors for writing the papers and incorporating all the corrections and improvements required by a thorough reviewing process; to the reviewers for kindly contributing their time and effort to ensure the highest quality of each paper; and to Martin Schindler for agreeing to publish this special section in Software and Systems Modeling and for his very valuable and friendly assistance in bringing it to publication.