SpecifyThis – Bridging Gaps Between Program Specification Paradigms

Wolfgang Ahrendt[1(✉)], Paula Herber[4], Marieke Huisman[2], and Mattias Ulbrich[3]

[1] Chalmers University of Technology, Gothenburg, SE, Sweden
ahrendt@chalmers.se
[2] University of Twente, Enschede, NL, The Netherlands
m.huisman@utwente.nl
[3] Karlsruhe Institute of Technology, Karlsruhe, DE, Germany
ulbrich@kit.edu
[4] University of Münster, Münster, DE, Germany
paula.herber@uni-muenster.de

Abstract. We motivate and summarise the track *SpecifyThis – Bridging gaps between program specification paradigms*, taking place at the International Symposium on Leveraging Applications of Formal Methods, ISoLA 2022.

Keywords: Specification · Verification · Formal methods

1 Introduction

The field of program verification has seen considerable successes in recent years. At the same time, both the variety of properties that can be specified and the collection of approaches that solve such program verification challenges have specialised and diversified a lot. Examples include contract-based specification and deductive verification of functional properties, the specification and verification of temporal properties using model checking or static analyses for secure information flow properties. While this diversification enables the formal analyses of ever more kinds of properties, it may leave the impression of isolated solutions that solve different, unrelated problems.

Here lies a great potential that waits to be uncovered: If either of the approaches can be extended to enable the interpretation of specifications used in other approaches and to use them beneficially in its analyses, a considerable extension of the power and reach of formal analyses is achievable. A discipline of "separation and integration of concerns" can be obtained, by, e.g., combining temporal specifications of a protocol with a contract-based specification of its implementation units.

The theme of this track is to investigate and discuss what can be achieved in joint efforts of the communities of different specification and verification techniques. This track is a natural next step following a series of well-structured

T. Margaria and B. Steffen (Eds.): ISoLA 2022, LNCS 13701, pp. 3–6, 2022.
https://doi.org/10.1007/978-3-031-19849-6_1

online discussions within the VerifyThis community during the last year. There, we identified first candidates for the combination/interplay of formal program verification methods. Following that, the ISoLA 2022 track addresses questions such as how specifications which are shared between different approaches should look like, how different abstraction levels can be bridged, how semantical differences can be resolved, which application areas can benefit from which method combinations, what artifacts can be carried forward through different verification technologies, what role user interaction (in form of specifications) plays, and how one can integrate the various techniques into the development processes.

2 Summary of Contributions

Jesper Amilon, Christian Lidström, and Dilian Gurov [1] (*Deductive Verification Based Abstraction for Software Model Checking*) describe a combination of model-checking and deductive verification, where deductive verification is applied to prove local pre-post specifications of source code *units*, and model checking is applied to prove global (temporal) properties of the *system*. The model checker is not applied to the source code, but to the abstractions provided by the pre-post-specifications. The approach is theoretically well founded in abstract contract theory [7], implemented in a combination of Frama-C and TLA+, and demonstrated on an example.

David Cok and Gary Leavens [2] (*Abstraction in Deductive Verification: Model Fields and Model Methods*) report how different abstraction techniques available in the Java Modeling Language JML can be employed to verify Java programs. To this end, the authors describe how model fields and methods can serve as means of abstraction and sketch how existing logical encodings of heap memory can be enriched to accommodate model entities. Abstraction from concrete program states to more abstract notions of state is an important vehicle to cross boundaries between different verification techniques.

Gidon Ernst, Alexander Knapp and Toby Murray [3] (*A Hoare Logic with Regular Behavioral Specifications*) introduce a variant of Hoare logic, which allows to capture trace properties (of terminating programs). In this way, behavioural and state properties can be combined in a single specification. The idea is that a program explicitly emits certain statements, which together form the trace of the program. The Hoare triples specify the assumptions on the trace so far, and capture the trace that will be emitted by the current program fragment. The Hoare logic with traces is proven sound in Isabelle, and the sketch of a completeness argument is given. The approach is implemented in a prototype tool, and illustrated on two case studies.

Klaus Havelund [4] (*Specification-based Monitoring of C++*) proposes an approach for specification-based monitoring of C/C++ applications using a mix of a state machine and rule based language. He presents LogScope, a system for monitoring event streams against formal specifications that are expressed in state-machine style but with expressive rules to describe possible events. LogScope takes such a formal specification together with an application, and

translates them into a monitored program. The author illustrates the applicability and expressiveness of the specification language and the overall approach with a number of examples.

Igor Konnov, Markus Kuppe, and Stephan Merz [6] (*Specification and Verification With the TLA$^+$ Trifecta: TLC, Apalache, and TLAPS*) show how different verification paradigms can be applied to models in the same modelling language. The authors use the formalism TLA$^+$ (*temporal logic of actions*) to model a termination detection algorithm for distributed systems. The specifcation, in which operations are defined using before-after-predicates in first-order-logic over set theory, is then subjected to different formal verification approaches, like explicit-state model checking, bounded symbolic model checking, and theorem proving The paper provides insights into how the different formal tools for TLA$^+$ can be used in combination to solve a non-trivial case study.

Gordon Pace and Wolfgang Ahrendt [8] (*Selective Presumed Benevolence in Multi-Party System Verification*) extend an existing sequent calculus for smart contracts with the concept of selective benevolence. To achieve this, they define new proof rules that enable us to assume benevolence of some (specified) parties in smart contract verification, while assuming potentially malevolent (worst-case) behavior from others. The benevolent parties can be defined as a predicate, either statically or dynamically. The authors discuss the benefits of presumed benevolence and its potential uses.

Thomas Santen [9] (*On the Pragmatics of Moving from System Models to Program Contracts*) describes a case study on the VerifyThis Long Term Challenge problem (the Hagrid key server) [5] on how to construct a verified implementation from an abstract system model. It first presents a model for the key server in Alloy. This model consists of a state description, and various transitions that model the key server operations. It then describes how this is transformed into executable code with contracts.

References

1. Amilon, J., Lidström, C., Gurov, D.: Deductive Verification Based Abstraction for Software Model Checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 7–28. Springer, Cham (2022)
2. Cok, D., Leavens, G.: Abstraction in deductive verification: Model fields and model methods. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 29–44. Springer, Cham (2022)
3. Ernst, G., Knapp, A., Murray, T.: A Hoare logic with regular behavioral specifications. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 45–64. Springer, Cham (2022)
4. Havelund, K.: Specification-based Monitoring in C++. Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 65–87. Springer, Cham (2022)
5. Huisman, M., Monti, R., Ulbrich, M., Weigl, A.: The verifyThis collaborative long term challenge. In: Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Ulbrich, M. (eds.) Deductive Software Verification: Future Perspectives. LNCS, vol. 12345, pp. 246–260. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64354-6_10

6. Konnov, I., Kuppe, M., Merz, S.: Specification and verification with the TLA$^+$ trifecta: TLC, Apalache, and TLAPS. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 88–105. Springer, Cham (2022)
7. Lidström, C., Gurov, D.: An abstract contract theory for programs with procedures. In: FASE 2021. LNCS, vol. 12649, pp. 152–171. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-71500-7_8
8. Pace, G., Ahrendt, W.: Selective Presumed Benevolence in Multi-Party System Verification. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 106–123. Springer, Cham (2022)
9. Santen, T.: On the pragmatics of moving from system models to program contracts. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. LNCS, vol. 13701, pp. 124–138. Springer, Cham (2022)