Noname manuscript No.
(will be inserted by the editor)

Challenges of Software Verification

Vincenzo Arceri · Luca Negrini · Luca Olivieri · Pietro Ferrara

the date of receipt and acceptance should be inserted later

Abstract Software verification aims to prove that a program satisfies some given properties for all its possible executions. Software evolved incredibly fast during the last century, exposing several challenges to this scientific discipline. The goal of the "Challenges of Software Verification Symposium" is to monitor the state-of-the-art in this field. This special issue of Software Tools for Technology Transfer presents novel theoretical directions and practical applications of these techniques. The papers in this special issue are extended versions of selected symposium papers from the proceedings of the 3^{rd} Challenges of Software Verification Symposium (CSV), which took place at the Ca' Foscari University of Venice, Venice, Italy, from June 6^{th} and 7^{th}, 2024.

1 The History of the CSV Symposium

The first edition[1] of the "Challenges of Software Verification Symposium" was held on May 20^{th}, 2022, after the ceremony awarding Prof. Patrick Cousot a PhD in Computer Science Honoris Causa by the Ca' Foscari University of Venice. The symposium featured 15 invited short talks (15 minutes each), and extended versions of some of the talks were featured in Springer Nature [1].

Having gathered interest from the community, the second edition[2] of the Symposium was held May 25^{th} and 26^{th}, 2023, covering theoretical results in the software verification field and their applications, presenting new tools and their impact on the Software Engineering community. The symposium comprised 18 invited full talks (30 minutes each), split into several sessions on theoretical and practical aspects of static analysis, abstract interpretation, software engineering, and security. Extended versions of selected talks were featured in a Special Section of the journal Software Tools for Technology Transfer [4].

The third edition[3] of the Symposium was held on June 6^{th} and 7^{th}, 2024, following the format and topics of the previous edition. 21 invited researchers from international institutions presented their work at the Symposium, engaging in full talks (30 minutes each), and this special issue of the journal Software Tools for Technology Transfer (STTT) collects revised and extended versions of 8 such talks.

2 This Special Issue

The guest editors invited a selection of the speakers of CSV 2024 to submit extended versions of their presentations, which were all peer-reviewed by at least three referees in a single-blind process.

Vincenzo Arceri
University of Parma, Parma, Italy
E-mail: vincenzo.arceri@unipr.it

Luca Negrini
Ca' Foscari University, Venice, Italy
E-mail: luca.negrini@unive.it

Luca Olivieri
Ca' Foscari University, Venice, Italy
E-mail: luca.olivieri@unive.it

Pietro Ferrara
Ca' Foscari University, Venice, Italy
E-mail: pietro.ferrara@unive.it

[1] https://unive-ssv.github.io/events/2022/05/20/csv.html

[2] https://unive-ssv.github.io/events/2023/05/25/csv.html

[3] https://unive-ssv.github.io/events/2024/06/06/csv.html

The papers selected for final publication not only present advances in the field of software verification [8, 5, 7], but bring contributions aimed at improving or simplifying software engineering practices [9, 3], at proving the correctness of complex systems [10, 2], and at simplifying the development and maintenance of academic tools [6].

In the following sections, we provide a summary of each paper featured in this special issue.

2.1 Easing Maintenance of Academic Static Analyzers [6]

Academic research in static analysis produces software implementations. These implementations are time-consuming to develop, and some need to be maintained in order to enable building further research upon the implementation. While necessary, these processes can be quickly challenging. This article documents the tools and techniques the authors have come up with to simplify the maintenance of Mopsa since 2017. Mopsa is a static analysis platform that aims to be sound. First, the authors describe an automated way to measure precision that does not require any baseline of true bugs obtained by manually inspecting the results. Further, it improves the transparency of the analysis and helps discover regressions during the continuous integration process. Second, the authors have taken inspiration from standard tools observing the concrete execution of a program to design custom tools observing the abstract execution of the analyzed program itself, such as abstract debuggers and profilers. Finally, the authors report on some cases of automated test case reduction.

2.2 IntraJ: An On-Demand Framework for Intraprocedural Java Code Analysis [7]

Static analysis tools play an important role in software development by detecting bugs and vulnerabilities. However, running these tools separately from the code editing process often causes developers to switch contexts, which can reduce productivity. Previous work has shown how Reference Attribute Grammars (RAGs) can be used for declarative implementation of competitive tooling for intraprocedural control-flow and data-flow analysis of Java source code, embodied in the tool IntraJ. In this paper, the authors demonstrate how IntraJ can be leveraged to provide interactive analysis results directly in the editor, similar to compile-time error detection, relying on automatic on-demand evaluation of Reference Attribute Grammars. The authors discuss the architecture of IntraJ and demonstrate how it can be integrated into the development process in three different ways: in the command line, in an editor integration based on the Language Server Protocol, and in an integration with the debugging tool CodeProber. The authors showcase the extensibility of IntraJ by illustrating how new client analyses and language constructs can be added to the framework through Reference Attribute Grammar specifications. Finally, the authors evaluate the interactive performance of IntraJ on a set of real-world Java benchmarks, demonstrating that IntraJ can provide interactive feedback to developers, achieving a response time of under 0.1 seconds for most compilation units.

2.3 ccReact: a Rewriting Framework for the Formal Analysis of Reaction Systems [2]

Reaction Systems (RSs) are a computational framework inspired by biochemical systems, where entities produced by reactions can enable or inhibit other reactions. RSs interact with the environment through a sequence of sets of entities called the context. In this work, the authors introduce **ccReact**, a novel interaction language for implementing and verifying RSs. **ccReact** extends the classical RS model by allowing the specification of recursive, non-deterministic, and conditional context sequences, thus enhancing the interactive capabilities of the models.

The authors provide a rewriting logic (RL) semantics for **ccReact**, making it executable in the Maude system. The authors prove that our RL embedding is sound and complete, thereby offering a robust tool for analyzing RSs. Our approach enables various formal analysis techniques for RSs, including the simulation of RSs interacting with **ccReact** processes, the verification of reachability properties, model checking of temporal (LTL and CTL) formulas, and the exploration of the system evolution through a graphical tool to better understand its behavior. The authors apply our methods to analyze RSs from different domains, including computer science and biological systems. Notably, the authors examine a complex breast cancer case study, demonstrating that our analysis can suggest improvements to the administration of monoclonal antibody therapeutic treatments in certain scenarios.

2.4 The digest framework: Concurrency-sensitivity for abstract interpretation [8]

Thread-modular approaches to static analysis help mitigate the state space explosion encountered when analyzing multi-threaded programs. This is enabled by

abstracting away some aspects of interactions between threads. This paper proposes the notion of *concurrency-sensitivity*, which determines how an analysis takes the computation history of a multi-threaded program into account to exclude spurious thread interactions. Just as for other form of sensitivity, such as flow-, context, and path-sensitivity, there is a trade-off to be made between precision and scalability. The choice of concurrency-sensitivity is typically hard-coded into the analysis engine. However, the suitability of a chosen sensitivity hinges on the program and property to be analyzed. The authors thus propose to decouple the concurrency-sensitivity from the analysis and realize this in a generic framework. The framework allows for the seamless incorporation of custom abstractions of the computation history of a thread, so-called *digests*, to exclude spurious thread interactions. While *concrete* digests track properties precisely, the framework enables further abstraction through *abstract* digests. These may decrease the analysis cost while hopefully retaining precision for the property of interest. The authors propose digests that, e.g., track held mutexes, thread IDs, or observed events. Digests tailored to programming language features, e.g., condition variables or recursive mutexes, highlight the framework's versatility.

2.5 Abstract Domain Adequacy: Weakening Completeness towards Static Analysis Precision [5]

Abstract interpretation offers sound and decidable approximations for undecidable queries related to program behavior. The effectiveness of an abstract domain is entirely reliant on the abstract domain itself, and the worst-case scenario is when the abstract interpreter responds with "don't know", indicating that anything could happen during runtime. Conversely, a desirable outcome is when the abstract interpreter provides information that exceeds a specified level of precision, resulting in a more precise answer. The concept of completeness relates to the precision level forfeited when performing computations within the abstract domain. The authors focus on the domain's ability to express program behavior, which the authors refer to as adequacy. This paper presents a domain refinement strategy toward adequacy and a simple sound proof system for adequacy designed to determine whether an abstract domain can provide satisfactory responses to specified program queries. Notably, this proof system is both language and domain-agnostic and can be readily incorporated to support static program analysis.

2.6 Reformulating Regression Test Suite Optimization using Quantum Annealing - an Empirical Study [9]

Maintaining software quality is crucial in the dynamic landscape of software development. Regression testing ensures that software works as expected after changes are implemented. However, re-executing all test cases for every modification is often impractical and costly, particularly for large systems. Although very effective, traditional *test suite optimization* techniques are often impractical in resource-constrained scenarios, as they are computationally expensive. Hence, *quantum computing* solutions have been developed to improve their efficiency but have shown drawbacks in terms of effectiveness. The authors propose reformulating the regression test case selection problem to use quantum computation techniques better. Our objectives are (i) to provide more efficient solutions than traditional methods and (ii) to improve the effectiveness of previously proposed quantum-based solutions. The authors propose *SelectQA*, a *quantum annealing* approach that can outperform the quantum-based approach *BootQA* in terms of effectiveness while obtaining results comparable to those of the classic *Additional Greedy* and *DIV-GA* approaches. Regarding efficiency, SelectQA outperforms DIV-GA and has similar results with the Additional Greedy algorithm but is exceeded by BootQA.

2.7 Leveraging Static Analysis for Cost-aware Serverless Scheduling Policies [3]

Mainstream serverless platforms follow opinionated and hard-coded scheduling policies to allocate functions on the available workers. Such policies may decrease the performance of the application due to locality issues (e.g., functions executed on workers far from the data they use). `APP` is a platform-agnostic declarative language that mitigates these problems by allowing serverless platforms to support multiple per-function scheduling logics. However, defining the "right" scheduling policy in `APP` is far from trivial, often requiring rounds of refinement involving knowledge of the underlying infrastructure, guesswork, and empirical testing.

The authors propose a framework that lightens the burden on the shoulders of users by deriving cost information from the functions, via static analysis, into a cost-aware variant of `APP` that they call `cAPP`. They present a prototype of such framework, where they extract cost equations from functions' code, synthesise cost expressions through off the-shelf solvers, and implement `cAPP` to support the specification and execution of cost-aware allocation policies.

2.8 Inference of Access Policies through Static Analysis [10]

Robot Operating System 2 (ROS 2) is the de-facto standard framework for developing distributed robotic applications. However, ensuring the correctness and security of these applications remains a significant challenge. This paper presents a novel approach to statically analyze ROS 2 applications using abstract interpretation. By extracting the computational graph of the application, our method derives minimal access control policies that can be use to leverage security. The authors implemented our approach using the Library for Static Analysis (LiSA), providing a toolset that facilitates the development of sound static analyzers for ROS 2. The results demonstrate the effectiveness of our approach in enhancing the security of ROS 2 applications.

Acknowledgments

We thank all the authors for their contributions, the organizers of CSV 2024 for their work in making the event possible, and the referees who reviewed the extended versions of the papers that appear in this special issue. Work partially supported by SERICS (PE00000014 – CUP H73C2200089001) under the NRRP MUR program funded by the EU – NGEU, and by iNEST – Interconnected NordEst Innovation Ecosystem funded by PNRR (Mission 4.2, Investment 49 1.5) NextGeneration EU (ECS_00000043 – CUP H43C22000540006).

References

1. Arceri, V., Cortesi, A., Ferrara, P., Olliaro, M., et al.: Challenges of Software Verification. Springer (2023)
2. Ballis, D., Brodo, L., Falaschi, M., Olarte, C.: ccreact: a rewriting framework for the formal analysis of reaction systems. International Journal on Software Tools for Technology Transfer (this issue) (2025)
3. De Palma, G., Giallorenzo, S., Laneve, C., Mauro, J., Trentin, M., Zavattaro, G.: Leveraging static analysis for cost-aware serverless scheduling policies. International Journal on Software Tools for Technology Transfer (this issue) (2025)
4. Ferrara, P., Arceri, V., Cortesi, A.: Challenges of software verification: the past, the present, the future. International Journal on Software Tools for Technology Transfer **26**(4), 421–430 (2024)
5. Mastroeni, I.: Abstract domain adequacy: Weakening completeness towards static analysis precision. International Journal on Software Tools for Technology Transfer (this issue) (2025)
6. Monat, R., Ouadjaout, A., Miné, A.: Easing maintenance of academic static analyzers. International Journal on Software Tools for Technology Transfer (this issue) (2025)
7. Riouak, I., Fors, N., Hedin, G., Reichenbach, C.: Intraj: An on-demand framework for intraprocedural java code analysis. International Journal on Software Tools for Technology Transfer (this issue) (2025)
8. Schwarz, M., Erhand, J.: The digest framework: Concurrency-sensitivity for abstract interpretation. International Journal on Software Tools for Technology Transfer (this issue) (2025)
9. Trovato, A., De Stefano, M., Pecorelli, F., Di Nucci, D., De Lucia, A.: Reformulating regression test suite optimization using quantum annealing - an empirical study. International Journal on Software Tools for Technology Transfer (this issue) (2025)
10. Zanatta, G., Caiazza, G., Ferrara, P., Negrini, L.: Inference of access policies through static analysis. International Journal on Software Tools for Technology Transfer (this issue) (2025)