International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-024-00766-x

GENERAL

Special Issue: SOAP 2023

State of the art in program analysis

Pietro Ferrara[1] · Liana Hadarean[2]

Accepted: 10 September 2024

Abstract
Over the last several decades, static and dynamic program analysis techniques have received widespread attention. Their application to mainstream programming languages always requires extending theories and finding practical solutions. 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 workshop papers from the proceedings of the 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis (SOAP'23).

Keywords Static analysis · Software engineering · Formal methods

1 SOAP

The 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis (SOAP'23) [3], co-located with the 44th ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI'23), comprised three invited and ten regular talks. This special issue of the journal Software Tools for Technology Transfer (STTT) contains revised and extended versions of 4 papers selected from this program.

Static and dynamic analysis techniques and tools for mainstream programming languages (such as Java, C, and JavaScript) have received widespread attention for a long time. These analyses' application domains range from core libraries to modern technologies like web services and mobile applications. Over time, various analysis frameworks have been developed that provide techniques for optimizing programs, ensuring code quality, and assessing security and compliance.

Given this context, SOAP aims to bring together the members of the program analysis community to share new developments and shape innovations in program analysis. This workshop covered novel analysis framework ideas, the application of existing static analysis techniques to industrial software, the adoption of static analysis in software engineering practices (such as DevOps), innovative designs, and analysis techniques, including preliminary results or work in progress. A main focus was also on the state of the practice of program analysis in the industry. The workshop agenda consisted of lively discussions on extensions of existing frameworks, the development of novel analyses and tools, and how program analysis is used in real-world scenarios.

✉ P. Ferrara
pietro.ferrara@unive.it

L. Hadarean
hadarean@amazon.com

[1] Ca' Foscari University, Venice, Italy

[2] Amazon Web Services, Santa Clara, USA

2 This special issue

The guest editors selected the peer-reviewed papers that are part of this special issue among the papers presented at SOAP'23. They represent a good balance between academic results and industrial applications [4, 5], both in authorship and content.

In particular, the first two papers investigate novel approaches to improve analyzes' efficiency [1], and to define the run-time semantics of low-level constructs of the C programming language [2]. In contrast, the following two papers explain how Amazon CodeGuru Reviewer was highly optimized to achieve scalability [4], and the inner technical details of CodePeer [5], an industrial static analyzer developed by AdaCore.

Below, we give a summary of each paper.

2.1 Speeding up static analysis with the split operator [1]

This paper introduces a new abstract operator modeling the split of control flow paths in the context of Abstract

Published online: 13 September 2024

Interpretation-based static analysis. The ultimate goal of this operator is to enable a more efficient analysis when using abstract domains that are computationally expensive. This operator has no negative effect on precision and occasionally results in a more precise analysis.

This work focuses on conditional branches guarded by numeric linear constraints, including implicit numerical branches. The experimental evaluation on real-world test cases shows that, by using the split operator, one can achieve significant efficiency improvements with respect to the classical approach for a static analysis based on the domain of convex polyhedra.

2.2 When long jumps fall short: control-flow tracking and misuse detection for non-local jumps in C [2]

The C programming language offers setjmp/longjmp as a mechanism for non-local control flow. This mechanism has complicated semantics. As most developers do not encounter it day-to-day, they may be unfamiliar with all its intricacies – leading to subtle programming errors. At the same time, most static analyzers lack proper support, implying that otherwise sound tools miss whole classes of program deficiencies.

This paper introduces a concrete semantics of a subset of C with setjmp/longjmp, where interprocedural longjmps are performed directly, and an equivalent formulation where such jumps are implemented via stack-unwinding at the call-sites. Reflecting this semantic equivalence, an approach for lifting existing interprocedural analyzes to support setjmp/longjmp and flag their misuse is proposed. To deal with the non-local semantics, this approach leverages side-effecting transfer functions, which, when executed, may additionally trigger contributions for program points that are not static control-flow successors. The analysis is applied to a real-world example and a set of litmus tests for other analyzers.

2.3 User-assisted code query customization and optimization [4]

Running static analysis rules in the wild, as part of a commercial service, demands special consideration of time limits and scalability given the large and diverse real-world workloads. Furthermore, these rules do not run in isolation, which exposes opportunities to reuse partial evaluation results across rules. Amazon CodeGuru Reviewer and its underlying rule-authoring toolkit, known as the Guru Query Language (GQL), encountered performance and scalability challenges.

This paper identifies corresponding optimization opportunities such as caching, indexing, and customization of data-flow specification, which rule authors can take advantage of as built-in GQL constructs. The experimental evaluation on a dataset of open-source GitHub repositories shows three times speedup and perfect recall using indexing-based configurations and two times speedup and a 51% increase in the findings for caching-based optimization. Customizing the data-flow specification, such as expanding the tracking scope, can yield a remarkable increase in the number of findings, as much as 136%. However, this enhancement comes at the expense of a longer analysis time.

2.4 Sound and precise static analysis using a generalization of static single assignment and value numbering [5]

This paper presents CodePeer, an industrial static analysis tool based on compiler optimization techniques such as static single assignment and value numbering. CodePeer infers and reports on implicit preconditions for each function of the program based on limitations it identifies within the algorithm of the function. Presuming these inferred preconditions are satisfied, CodePeer then simulates the execution of each function and identifies places where a program might still fail at run time due to violating some run-time check or an error that leads to undefined behavior. CodePeer uses static single assignment and global value numbering to ensure that the determination of possible run-time values of each variable and expression encountered during the simulation of the execution of each function is both sound and precise. The approximations performed to ensure that the determination of possible values converges in the face of loops and recursion are systematic and based on the kinds of conservative analysis performed by compiler optimizers. The output of CodePeer includes, for each function, an enumeration of its global inputs and global outputs and inferred preconditions and postconditions. The output also includes, interspersed within a listing of the source of the function, an identification of the places where possible run-time failures or undefined behavior could occur. This output is designed to support code review, which gives the tool its CodePeer name.

Acknowledgement First, we thank all the authors for their contributions. A special thanks also goes to the program committee of SOAP 2023 for their help in selecting the papers for the conference program and to all the referees who reviewed the extended versions of the papers that appear in this special issue.

References

1. Arceri, V., Dolcetti, G., Zaffanella, E.: Speeding up static analysis with the split operator. Int. J. Softw. Tools Technol. Transf. (2024, in press)

2. Erhard, J., Schwarz, M., Vojdani, V., Saan, S., Seidl, H.: When long jumps fall short: Control-flow tracking and misuse detection for non-local jumps in c. Int. J. Softw. Tools Technol. Transf. (2024, in press)
3. Ferrara, P., Hadarean, L. (eds.): Proceedings of the 12th ACM SIGPLAN International Workshop on the State of the Art in Program Analysis, SOAP ACM, New York (2023). https://doi.org/10.1145/3589250
4. Liblit, B., Lyu, Y., Mukherjee, R., Tripp, O., Wang, Y.: User-assisted code query customization and optimization. Int. J. Softw. Tools Technol. Transf. (2024, in press)
5. Taft, T.: Sound and precise static analysis using a generalization of static single assignment and value numbering. Int. J. Softw. Tools Technol. Transf. (2024, in press)