Techniques for Efficient Automated Elimination of False Positives

20th IEEE International Working Conference on Source Code Analysis and Manipulation

September 27-September 28, 2020 - Adelaide, Australia

Tukaram Muske
Tata Consultancy Services, Pune, India
t.muske@tcs.com

Alexander Serebrenik
Eindhoven University of Technology, Eindhoven, The Netherlands
a.serebrenik@tue.nl
Motivation

- Static analysis – large number of alarms

- Automated false positives elimination (AFPE)
  - Postprocessing using model checkers

- Techniques combined for scalability
  - Application-level slicing
  - Verification using context expansion
  - Verification context-level slicing

- The combination
  - Useful
  - Time taken is a major concern
    (too many slicing and model checking calls)

4 slicing calls and 3 model checking calls*

1. Slicing with “main” as entry-point
2. Slicing with “baz” as entry-point, and then model checking
3. Slicing with “bar” as entry-point, and then model checking
4. Slicing with “foo” as entry-point, and then model checking

*It is under assumption that first two model checking calls result in counter-example and the third call times out/proves that the assertion holds.
Technique 1 - Identification of Redundant Slicing Calls

- Observation
  - Many context-level slicing calls are redundant

- Contribution – sound technique
  - Based on data dependencies
  - Implementation using PDGs

- Under evaluation

1. Slicing with “main” as entry-point
2. Slicing with “baz” as entry-point, and then model checking
3. Slicing with “bar” as entry-point, and then model checking
4. Slicing with “foo” as entry-point, and then model checking

Highlighted calls can be redundant!
Technique 2 - Redundant Calls on Partitioned-code

- **Code partitioning to scale static analysis**
  - Breaks the system into multiple modules

- **Observation**
  - Context-level slicing and model checking calls can be repetitive (redundant)

- **Memoization-based technique**
  - Reuses results across partitions

- **Initial evaluation**
  - Using 5 applications
  - AFPE time reduction – by up to 56%, median 12%

Calls with “foo” and “bar” as entry-points are repetitive*

*It is under assumption that no model checking call times out or proves that the assertion holds.
Conclusions & Future work

❑ Conclusion
   • Reducing redundancy in AFPE helps to improve its efficiency
   • Design of more such techniques is required

❑ Future work
   • Evaluate the two techniques
   • Improve technique 1 to identify more redundant slicing calls
   • Design technique to skip calls based on the history of model checking calls
Provoking Questions

❑ Are developers from industry really using the software engineering practices and techniques being researched by us?

❑ With the hype in machine learning, are we sometimes unnecessarily using it in tasks in source code analysis and manipulation, and even in software engineering?