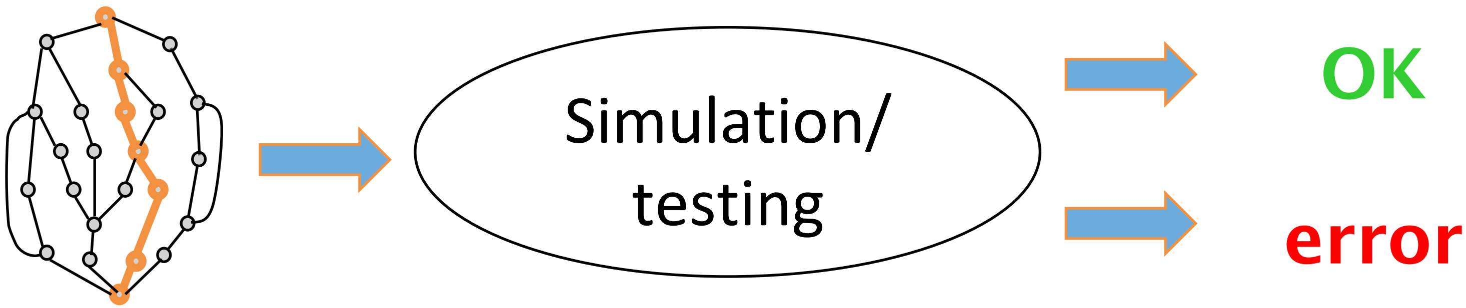
SMT-Based Refutation of Spurious Bug Reports in the Clang Static Analyzer

Mikhail R. Gadelha*, Enrico Steffinlongo, Lucas C. Cordeiro, Bernd Fischer, Denis A. Nicole

*Sidia Institute of Science and Technology

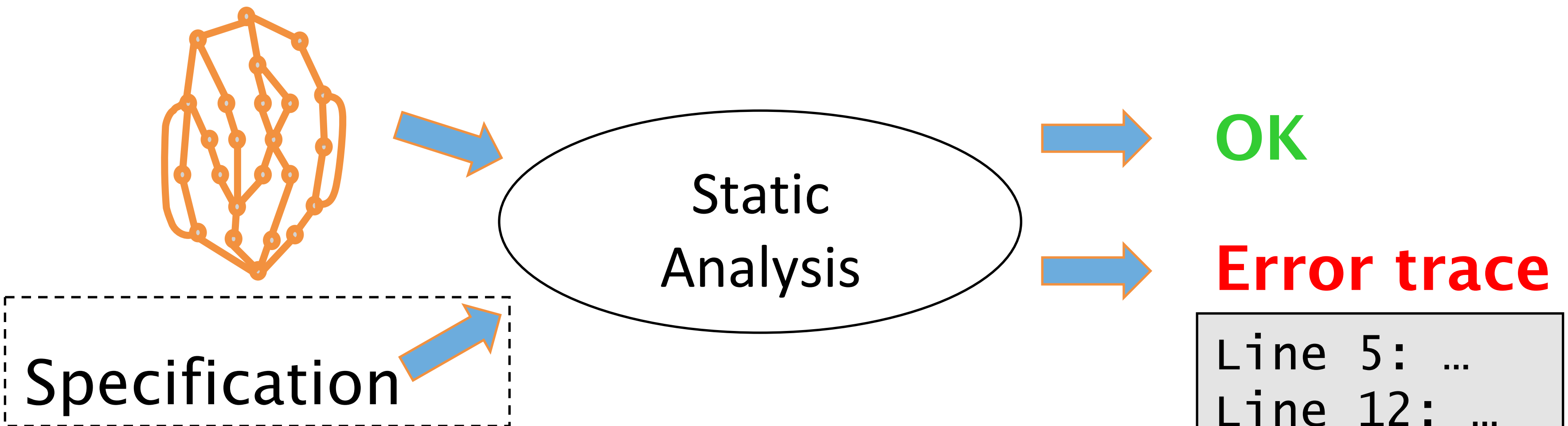
m.gadelha@samsung.com

Static Analysis vs Testing



- Usually checks one path in the program.
- May miss errors.
- It’s fast.

Static Analysis vs Testing




```
Line 5: …
Line 12: …
…
Line 41:…
```

- Report errors as traces.
- Explores all executions, might over-approximate paths.
- Might present false positives due to over-approximations.
- Does not scale well (state/path explosion).

Clang Static Analyzer (CSA)

- Fast and easy to use state-of-the-art static analyzer framework built on top of clang.

Clang Static Analyzer (CSA)

- Fast and easy to use state-of-the-art static analyzer framework built on top of clang.

- Performs context-sensitive interprocedural analysis in each translation units of a project.

Clang Static Analyzer (CSA)

- Fast and easy to use state-of-the-art static analyzer framework built on top of clang.
- Performs context-sensitive interprocedural analysis in each translation units of a project.
- Offer a wide range of checkers, including pattern matching checkers and path-sensitive checkers.
  - Constraints generated from symbolically executing the program; no abstract interpretation involved.

Clang Static Analyzer (CSA)

- Fast and easy to use state-of-the-art static analyzer framework built on top of clang.

- Performs context-sensitive interprocedural analysis in each translation units of a project.

- Offer a wide range of checkers, including pattern matching checkers and path-sensitive checkers.
  - Constraints generated from symbolically executing the program; no abstract interpretation involved.

- Sacrifices precision for speed.

Clang Static Analyzer (CSA)

```
unsigned int func(unsigned int a) {
  unsigned int *z = 0;
  if ((a & 1) && ((a & 1) ^ 1))
    return *z;
  return 0;
}
```

Is this program safe?

Clang Static Analyzer (CSA)

```
unsigned int func(unsigned int a) {
  unsigned int *z = 0;
  if ((a & 1) && ((a & 1) ^ 1))
    return *z;
  return 0;
}
```

- This program is safe, i.e., the null pointer dereference is unreachable.

Running the CSA

DEMO

Refuting False Bugs using SMT Solvers

- Why don’t we replace the imprecise solver?

Refuting False Bugs using SMT Solvers

- Why don’t we replace the imprecise solver?

- First SMT backend implemented (Z3) in late 2017 by Dominic Chan. It was aimed to replace the built-in constraint solver in the CSA.

Refuting False Bugs using SMT Solvers

- Why don't we replace the imprecise solver?
- First SMT backend implemented (Z3) in late 2017 by Dominic Chan. It was aimed to replace the built-in constraint solver in the CSA.
- It was up to 20 times slower than the built-in constraint solver :/

Refuting False Bugs using SMT Solvers

We developed an alternative solution: to use the more precise SMT solvers to reason about bug reachability only as a **post processing** step.

Refuting False Bugs using SMT Solvers

- Our extension refutes false bug reports produced by the path sensitive checkers.

Refuting False Bugs using SMT Solvers

- Our extension refutes false bug reports produced by the path sensitive checkers.
- We use SMT solvers to check the reachability of reported bugs: all the constraints in a bug path are encoded and checked for satisfiability.

Refuting False Bugs using SMT Solvers

- Our extension refutes false bug reports produced by the path sensitive checkers.

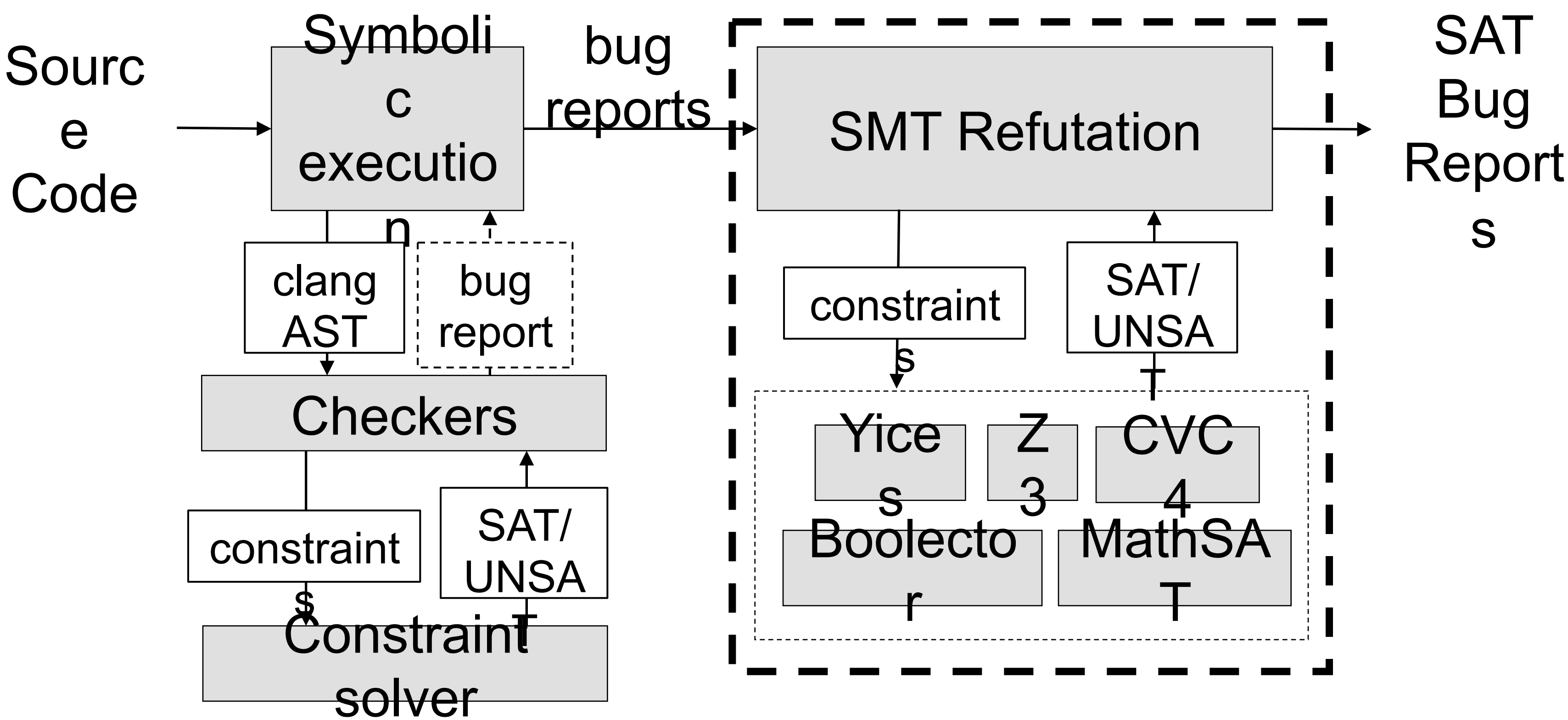
- We use SMT solvers to check the reachability of reported bugs: all the constraints in a bug path are encoded and checked for satisfiability.

- We implemented support for five different state-of-the-art SMT solvers in the CSA: Z3, Boolector, MathSAT, Yices and CVC4.

Running the CSA with SMT refutation

DEMO

Clang Static Analyzer with SMT Refutation



Experimental Evaluation

- We evaluated twelve open-source projects:
  - tmux, Redis, openSSL, twin, git, postgreSQL, sqlite3, curl, libWebM, Memcached, Xerces-c, and XNU.
- Using five different SMT solvers:
  - Z3, Boolector, MathSAT, CVC4 and Yices
- Instructions to reproduce the experiments in: https://github.com/mikhailramalho/analyzer-projects

Experimental Evaluation

Projects	time (s) (no refutation)	time (s) (refutation)*	reported bugs (no refutation)	refuted bugs
redis	347.8	338.3	93	1
openSSL	138	128	38	2
twin	225.6	216.7	63	1
git	488.7	405.9	70	11
postgreSQL	1167.2	1112.4	196	6
SQLite3	1078.6	1058.4	83	15
xerces-c++	489.8	433.2	81	2
XNU	3441.7	3405.1	557	51
tmux	86.5	89.9	19	0
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
memcached	96	96.2	25	0

* average time of Z3, Boolector, MathSAT, Yices and CVC4.

Experimental Evaluation

Projects	time (s) (no refutation)	time (s) (refutation)*	reported bugs (no refutation)	refuted bugs
redis	347.8	338.3	93	1
openSSL	138	128	38	2
twin	225.6	216.7	63	1
git	488.7	405.9	70	11
postgreSQL	1167.2	1112.4	196	6
SQLite3	1078.6	1058.4	83	15
xerces-c++	489.8	433.2	81	2
XNU	3441.7	3405.1	557	51
tmux	86.5	89.9	19	0
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
memcached	96	96.2	25	0

* average time of Z3, Boolector, MathSAT, Yices and CVC4.

Experimental Evaluation

Average 10% bugs removed

Projects	time (s) (no refutation)	time (s) (refutation)*	reported bugs (no refutation)	[illegible]
redis	347.8	338.5	95	1
openSSL	138	128	38	2
twin	225.6	216.7	63	1
git	488.7	405.9	70	11
postgreSQL	1167.2	1112.4	196	6
SQLite3	1078.6	1058.4	83	15
xerces-c++	489.8	433.2	81	2
XNU	3441.7	3405.1	557	51
tmux	86.5	89.9	19	0
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
memcached	96	96.2	25	0

* average time of Z3, Boolector, MathSAT, Yices and CVC4.

Experimental Evaluation

Average 6% speedup

Projects	time (s) (no refutation)	(refutation)	reported bugs (no refutation)	refuted bugs
redis	347.8	338.5	93	1
openSSL	138	128	38	2
twin	225.6	216.7	63	1
git	488.7	405.9	70	11
postgreSQL	1167.2	1112.4	196	6
SQLite3	1078.6	1058.4	83	15
xerces-c++	489.8	433.2	81	2
XNU	3441.7	3405.1	557	51
tmux	86.5	89.9	19	0
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
memcached	96	96.2	25	0

* average time of Z3, Boolector, MathSAT, Yices and CVC4.

Experimental Evaluation

Projects	time (s) (no refutation)	time (s) (refutation)*	reported bugs (no refutation)	refuted bugs
redis	347.8	338.3	93	1
openSSL	138	128	38	2
twin	225.6	216.7	63	1
git	488.7	405.9	70	11
postgreSQL	1167.2	1112.4	196	6
SQLite3	1078.6	1058.4	83	15
xerces-c++	489.8	[illegible]	81	2
XNU	3441.7	[illegible]	557	51
tmux	86.5	89.9	19	0
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
memcached	96	96.2	25	0

Average 1% slowdown

* average time of Z3, Boolector, MathSAT, Yices and CVC4.

Experimental Evaluation

- In total, 89 bugs were refuted and an in-depth analysis of them show that all of them were false positives.

Experimental Evaluation

- In total, 89 bugs were refuted and an in-depth analysis of them show that all of them were false positives.

- The average time to analyse the projects with refuted bugs was 35.0 seconds faster, a 6.25% speed up.

Experimental Evaluation

- In total, 89 bugs were refuted and an in-depth analysis of them show that all of them were false positives.

- The average time to analyse the projects with refuted bugs was 35.0 seconds faster, a 6.25% speed up.

- Out of the four projects where no bug was refuted the analysis was 1.0 second slower on average: a 1.24% slowdown.

How do I run CSA on my project?

DEMO

Conclusions

- The technique removes from 0% to 20% bugs in real-world projects:
  - Empirical evidences shows that, on average, 50% of the bugs reported are spurious.
- The technique only incurs in a small overhead, and can actually make the analysis faster in a number of real-world projects.
- Further improvements can only be achieved through cross translation-unit support in the CSA.

The future?

- D54978: Move the SMT API to LLVM:
  - Part of the clang 9.0.
- Validation of optimizations using SMT:
  - Already done in the ScalarEvolution pass.
- Maybe an SMT backend in LLVM:
  - Memory handling?
  - Loops?

Acknowledgments

- Thank you to:
  - George Karpenkov
  - Artem Dergachev
  - Devin Coughlin
  - Anna Zaks
  - Réka Kovács
  - Dominic Chen
  - Gábor Horváth

Thank you!

- Me: mikhail.ramalho@gmail.com
- Experiments: https://github.com/mikhailramalho/analyzer-projects
- Clang static analyzer: https://clang-analyzer.llvm.org/
- 5 min video: https://www.youtube.com/watch?v=ylW5iRYNsGA