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

Mikhail R. Gadelha[1], Enrico Steffinlongo[2], Lucas C. Cordeiro[3], Bernd Fischer[4], Denis A. Nicole[2]
m.gadelha@samsung.com
[1]Sidia Institute of Science and Technology [2]University of Southampton [3]University of Manchester [4]Stellenbosch University

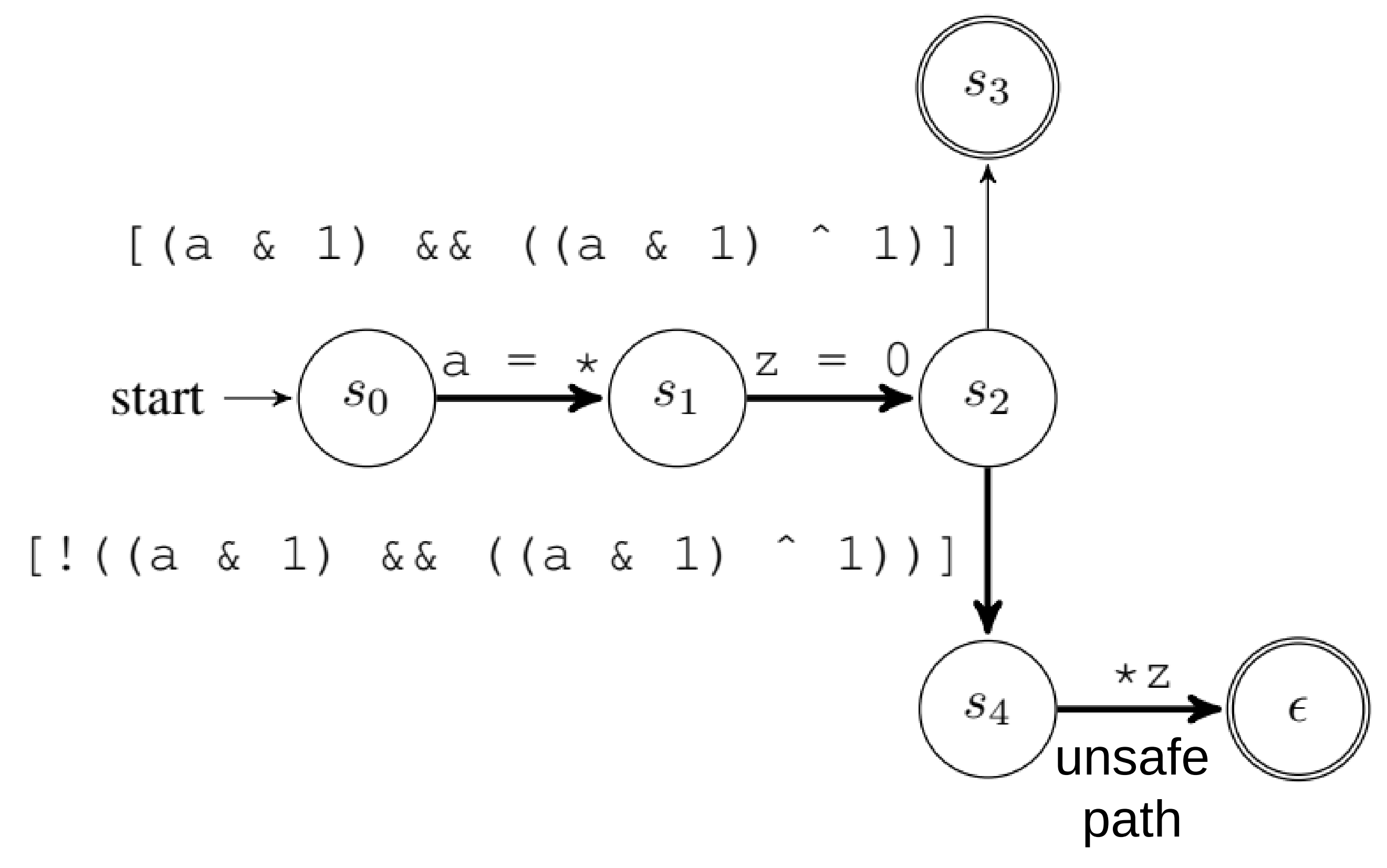
The Clang Static Analyser (CSA)

Consider the following program:

```
1 unsigned int func(unsigned int a) {
2   unsigned int *z = 0;
3   if ((a & 1) && ((a & 1) ^ 1))
4     return *z;
5   return 0;
6 }
```

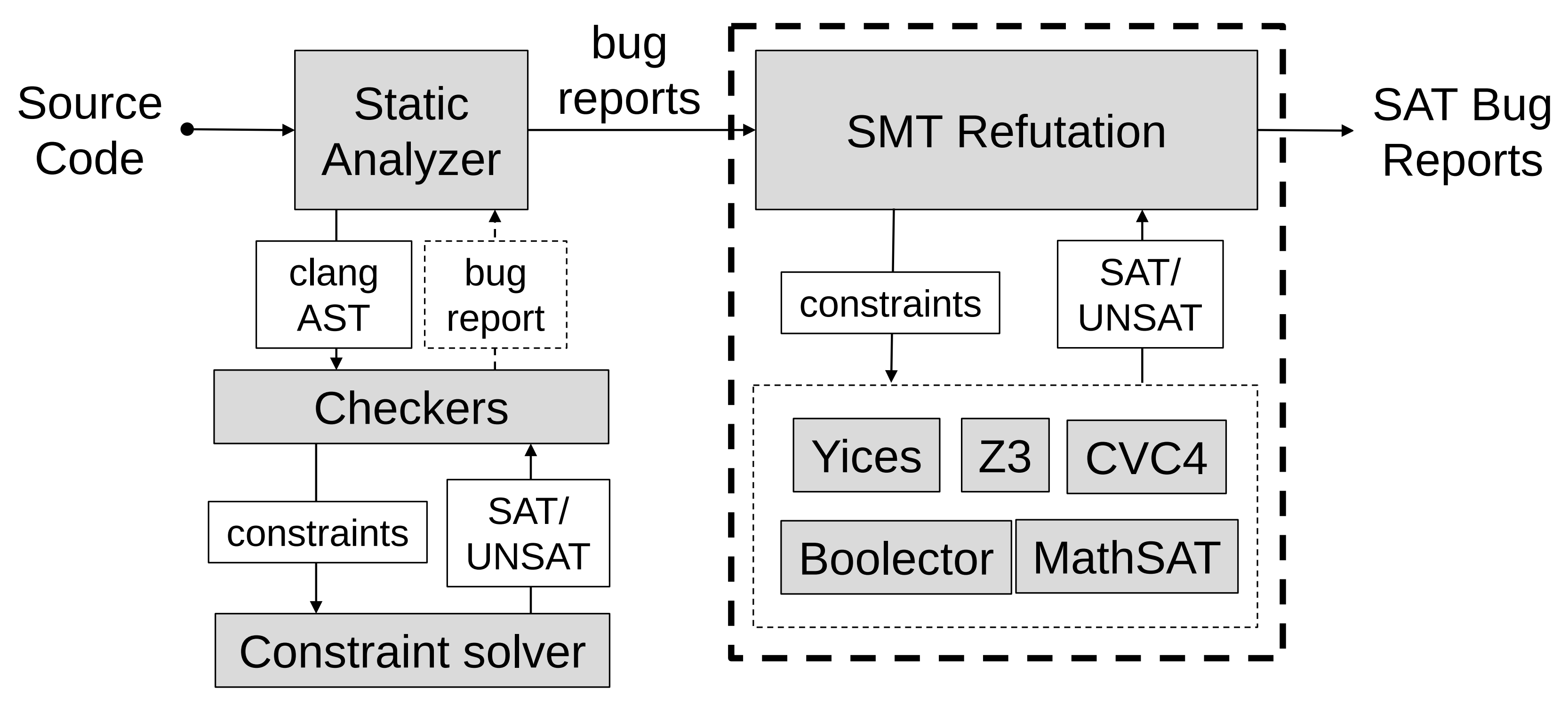
This program is safe, i.e., the unsafe pointer dereference in line 4 is unreachable because the guard in line 3 is never holds; `a & 1` holds if the last bit in a is one, and `(a & 1) ^ 1` inverts the last bit in a. The clang static analyzer, however, produces the following (spurious) bug report when analyzing the program:

```
main.c:4:12: warning: Dereference of null pointer (loaded
from variable 'z')
    return *z;
           ^~
1 warning generated.
```



Such spurious bug reports are in practice common; in our experience, about 50% of the reports in large systems are actually spurious. Identifying spurious bug reports and refactoring the code to suppress them puts a large burden on developers and runs the risk of introducing actual bugs. Our refutation algorithm removes false bugs introduced by the unsound constraint solver in the clang static analyzer.

Static analysis with refutation



Encoding path constraints in SMT

We developed an solution to prevent false bugs found by the unsound constraint solver from being reported to the user: we use the more precise SMT solvers to reason about bug reachability in the post processing step. The clang static analyzer already has heuristics in place to remove incorrect bug reports, so we extended those heuristics to precisely encode the constraints in SMT and to check for satisfiability.

After the static analyzer generates the bug reports, the SMT-based refutation extension will encode the bug constraints as SMT formulas and check them for satisfiability. If the constraint are unsatisfiable, the bug is false and no bug report is generated for that bug.

Experimental evaluation

Projects	time (s) (no ref)	time (s) (with ref)*	reported bugs (no ref)	refuted bugs
tmux	86.5	89.9	19	0
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
curl	79.8	79.9	39	0
libWebM	43.9	44.2	6	0
Memcached	96	96.2	25	0
xerces-c++	489.8	433.2	81	2
XNU	3441.7	3405.1	557	51
Total	7683.7	7408.5	1270	89

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

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 analyze 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.

Extending the SMT solver support

We also developed a new generic SMT API which makes it easier to support new solvers. The API is minimal and only requires the solver to support quantifier-free bit-vector theories (QF_BV). In particular, the clang static analyser already supported Z3 as a backend but we rewrote it to use the new generic SMT API and the new backend was released as part of clang v7.0. We also implemented support for four new solvers: Boolector, Yices, MathSAT, and CVC4, but they are not part of the mainstream clang yet.

The development of the generic SMT API opens the door for SMT solvers in all LLVM projects: the SMT API is now part of the LLVM project and can be used by any project under the LLVM umbrella.

Acknowledgements

Thanks to George Karpenkov, Artem Dergachev, Devin Coughlin, Anna Zaks, Réka Kovács, Dominic Chen and Gábor Horváth for the help during the development of this project.

This project was developed as part of the Google Summer of Code 2018.