Symbiotic 6

Marek Chalupa, Martina Vitovská, Tomáš Jašek, Michael Šimáček, Jan Strejček

Masaryk University, Brno

6. 4. 2019, SV-COMP, Prague
Symbiotic – workflow

1. Static analysis
2. Instrumentation
3. Program slicing
4. Symbolic execution

Flow directions:
- LLVM: Static analysis → Instrumentation → Program slicing → Symbolic execution
- TRUE, FALSE, UNKNOWN
Symbiotic – workflow

dg library -> Static analysis

sbt-instrumentation -> Instrumentation

sbt-slicer -> Program slicing

KLEE -> Symbolic execution

TRUE
FALSE
UNKNOWN
• We now do not instrument the program with code that tracks and checks memory operations. Instead, we slice directly w.r.t. memory safety errors (as determined by a static analysis) and let KLEE find the errors.

• We modified KLEE (and clang) to consider scope of local variables.

• We removed our heuristics from KLEE that could be unsound (not on SV-COMP benchmarks).
Symbiotic – MemSafety Results

[Graph showing time in seconds on the y-axis and accumulated score on the x-axis, with various lines representing different tools such as 2LS, CBMC-Path, CBMC, CFA-Seq, DepthK, DIVINE explicit, DIVINE-SMT, ESBMC-kind, Map2Check, PredatorHP, Symbiotic, UAutomizer, UKoial, UTaipan, and 2LS.]
The slicer in Symbiotic may slice away non-terminating loops. To fix that, we implemented replaying of error paths.

We extended Symbiotic to perform a simple termination analysis.

We plugged in a range analysis for no-overflow checking.
Symbiotic – Results