# Symbiotic 3: New Slicer and Error-Witness Generation[*]
## (Competition Contribution)

Marek Chalupa[†], Martin Jonáš, Jiri Slaby, Jan Strejček, and Martina Vitovská

Faculty of Informatics, Masaryk University, Brno, Czech Republic

**Abstract.** Symbiotic 3 is a new generation of a bug-detection tool for C programs. The tool sticks to the combination of program instrumentation, slicing, and symbolic execution. Large parts of the tool are rewritten, in particular the managing and instrumentation scripts and slicer (including points-to analysis). Further, the symbolic executor Klee has been modified to produce error-witnesses. The changes are commented in the description of the tool workflow.

## 1 Verification Approach and Software Architecture

As the previous versions of Symbiotic [9,7], the new version also follows the approach suggested in [8]: an analyzed program is (*i*) instrumented with code that tracks a finite-state machine describing erroneous behaviors, (*ii*) reduced by slicing [10] that removes code not influencing the state machine moves, and (*iii*) symbolically executed [6] to find erroneous runs in the program.

The workflow of Symbiotic 3 (together with indication of chosen programming languages and employed external tools with their respective versions) is provided in Figure 1. Our tool currently focuses on the *Error Function Unreachability* property (however, the approach can handle the other properties as well and we plan to support them in near future). The *code cleanup* modifies the C source (e.g. to bypass the known bug in clang where inlined functions are omitted). The program is then translated to llvm, checked for unsupported functionality (e.g. creation of new threads), and instrumented. As we support only the unreachability property, the instrumentation is trivial. This step makes also another small modifications of the program, e.g. each allocated variable is initialized to a nondeterministic value (to solve problems with uninitialized variables appearing in some benchmarks). After linking with `lib.bc` (which contains our definitions of `__VERIFIER_*` functions) and some optimization passes, namely control flow graph optimization and constant propagation, we slice the program.

The slicer in Symbiotic 3 is written from scratch. While the previous slicer followed the slicing algorithm of [10], the current one implements slicing based on dependence graphs [3,5]. The slicer relies on field-sensitive, flow-insensitive

[*] The research was supported by The Czech Science Foundation, grant GA15-17564S.

[†] Corresponding author: `xchalup4@fi.muni.cz`

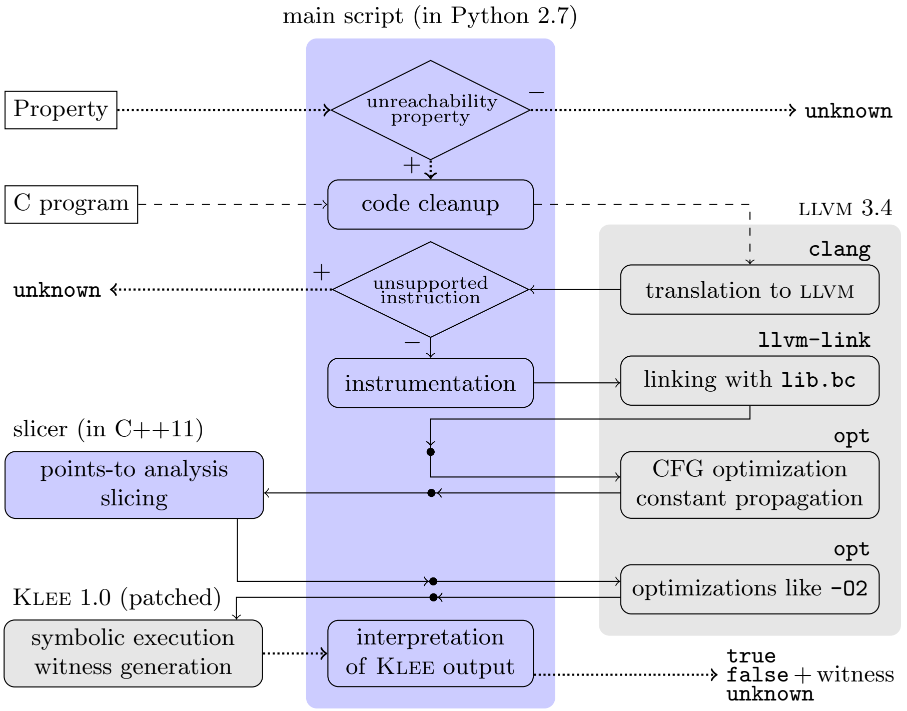

Fig. 1: Workflow of Symbiotic 3. Dashed lines represent C programs, solid lines LLVM bytecode, and dotted lines text data.

points-to analysis (extended with an "unknown offset" value), which has been also reimplemented. The new slicer is substantially faster than the previous one.

The sliced program is optimized again (with passes similar to `-O2` optimization level) and symbolically executed with our fork of Klee [1]. We modified it to stop the computation when assertion violation is detected and to produce the corresponding error witness. The exact versions of Klee and the solvers STP [4] and MiniSat [2] called by Klee can be found in the Symbiotic 3 distribution. Finally, the Klee output is translated into the required form. In particular, a witness is translated to the GraphML format by a Perl script.

## 2 Strengths and Weaknesses

The main strengths of the approach are its soundness and universality; the approach can be applied also to the *Concurrency* benchmarks and these with more complex properties, which are currently not supported by our implementation (and thus skipped). Another advantage is the modularity of the tool architecture.

The main disadvantage is the high computational cost of symbolic execution. Especially programs with loops, recursion, or intensive branching cannot be analyzed within reasonable time unless an erroneous execution is detected soon.

The fundamental problem are programs with infinite paths as these cannot be fully symbolically executed in finite time.

## 3 Tool Setup and Configuration

- *Download:* `https://github.com/staticafi/symbiotic/releases/tag/3.0.1`
- *Installation:* Unpack the archive. Further, `gcc 4.9` or higher, GNU utils (`sed`), `python 2.7`, and `perl` with the `XML::Writer` module are required.
- *Participation Statement:* Symbiotic 3 participates in all categories.
- *Execution:* Run `./symbiotic OPTS <source>`, where available `OPTS` include:
  - `--64` sets environment for 64-bit benchmarks
  - `--prp=file` sets the specification file to use
  - `--help` shows the full list of possible options

## 4 Software Project and Contributors

Symbiotic 3 has been developed by M. Chalupa, J. Slaby, M. Vitovská, and M. Jonáš under supervision of J. Strejček. The tool is available under the GNU GPLv2 License. The project is hosted by the Faculty of Informatics, Masaryk University. llvm, Klee, stp, and MiniSat are also available under open-source licenses. The project web page is: `https://github.com/staticafi/symbiotic`

## References

1. C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pages 209–224. USENIX Association, 2008.
2. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
3. J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. In *International Symposium on Programming*, volume 167 of *LNCS*, pages 125–132. Springer, 1984.
4. V. Ganesh and D. L. Dill. A decision procedure for bit-vectors and arrays. In *CAV*, volume 4590 of *LNCS*, pages 519–531. Springer, 2007.
5. S. Horwitz, T. W. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, 1990.
6. J. C. King. Symbolic execution and program testing. *Communications of ACM*, 19(7):385–394, 1976.
7. J. Slaby and J. Strejček. Symbiotic 2: More precise slicing - (competition contribution). In *TACAS*, volume 8413 of *LNCS*, pages 415–417. Springer, 2014.
8. J. Slaby, J. Strejček, and M. Trtík. Checking properties described by state machines: On synergy of instrumentation, slicing, and symbolic execution. In *FMICS*, volume 7437 of *LNCS*, pages 207–221. Springer, 2012.
9. J. Slaby, J. Strejček, and M. Trtík. Symbiotic: Synergy of instrumentation, slicing, and symbolic execution - (competition contribution). In *TACAS*, volume 7795 of *LNCS*, pages 630–632. Springer, 2013.
10. M. Weiser. Program slicing. In *Proceedings of ICSE*, pages 439–449. IEEE, 1981.