BLISS: Improved Symbolic Execution by Bounded Lazy Initialization with SAT Support*

Nicolás Rosner[1], Jaco Geldenhuys[2], Nazareno Aguirre[3,5], Willem Visser[2] and Marcelo F. Frias[4,5]

[1] Dept. of Computer Science, Universidad de Buenos Aires, Argentina.
[2] Dept. of Mathematical Sciences, Stellenbosch University, South Africa.
[3] Dept. of Computer Science, Universidad Nac. de Río Cuarto, Argentina.
[4] Dept. of Computer Engineering, Instituto Tecnológico de Buenos Aires.
[5] National Scientific and Technical Research Council (CONICET), Argentina.

Abstract. Traditional testing is a widely adopted approach to guaranteeing software correctness, but its well-known limitations threaten its effectiveness as a bug-finding technique. Therefore, more thorough program analysis techniques, which may offer greater levels of confidence, constitute an important research topic in software engineering. A technique that offers better guarantees of correctness is model checking. Java PathFinder (JPF) is a well-known tool based on this technique, that targets Java source code and, through an extension called Symbolic PathFinder (SPF), is able to automatically generate test cases, search for violations of user-provided assertions, handle arithmetic constraints and complex data structures. SPF combines *symbolic execution* with model checking and constraint solving, to systematically explore program paths for verification, as well as for automated test input generation by solving the path constraints obtained during the exploration.

To effectively handle heap-allocated structures, SPF generalizes symbolic execution (which traditionally targeted basic datatypes) by introducing *Lazy Initialization* (LI): it constructs the heap as the program paths are explored, and defers concretization of symbolic heap object attributes as much as possible. LI produces a significant reduction in spurious and redundant symbolic structures, which is improved by Bounded Lazy Initialization (BLI), by taking advantage of precomputed relational bounds on the interpretation of class fields in order to reduce the number of spurious structures even further.

In this article we present BLISS, a novel technique that builds upon BLI, extending it with field bound refinement and satisfiability checks. Field bounds are refined while a symbolic structure is concretized, avoiding cases that, due to the concrete part of the heap and the field bounds, can be deemed redundant. Satisfiability checks on refined symbolic heaps allow us to prune these heaps as soon as it can be confirmed that they cannot be extended to any valid concrete heap. Compared to LI and BLI, BLISS reduces the time required by LI by up to 4 orders of magnitude for the most complex data structures. Moreover, the number of partially symbolic structures obtained by exploring program paths is reduced by BLISS by over 50%, with reductions of over 90% in some cases (compared to LI). BLISS uses less memory than LI and BLI, which enables the exploration of states unreachable by previous techniques.

* in *Transactions on Software Engineering*, 41(7), IEEE CS, 2015.