Symbolic Execution for Software Testing: Three Decades Later: A Summary

Caleb Bowers

11-5-2019

I. Summary

What follows is my best attempt at summary information for [1]. This paper provides an "overview of modern symbolic execution techniques, discus[es] their key challenges in terms of path exploration, constraint solving, and memory modeling, and discuss[es] several solutions drawn primarily from the authors' own work." The authors do not propose that they will provide a comprehensive summary of symbolic execution techniques, but rather they wish to highlight the key challenges as exhibited in their own work.

A. Background

Symbolic execution provides a powerful methodology for generating test suites that can offer higher coverage than traditional testing approaches, and is capable of finding errors hidden deep within complex software applications. These capabilities are provided by symbolic execution's ability to explore as many different paths of program execution as possible within a given amount of time. For each path explored, symbolic execution *i*) generates a set of concrete input variables that execute along that path and *ii*) checks for the presence of traditional property violations and program errors.

The historical development of symbolic execution began with the idea that researchers make use of *symbolic values*, rather than actual data values as inputs to an execution instance. Additionally, program variables are represented as *symbolic expressions* over the symbolic input values. The output values of a program executing on such a symbolic logic are expressed as a function of the input values. All symbolic execution paths of a program are represented in a tree structure and can all be evaluated by running the program over a set of inputs such that all the symbolic paths are explored exactly once. Symbolic execution may fail to generate an input if the constraint on a symbolic path cannot be efficiently solved by a constraint solver. More plainly, even though a symbolic input may exist and the path itself may be valid, symbolic execution will not be able to determine the veracity of this path, because of the constraint is too complex to evaluate. Similar solution restrictions exist for looping (where infinite paths may exist and only a timeout can resolve this).

B. Modern Symbolic Execution Techniques

Modern symbolic execution techniques offer the ability to mix concrete and symbolic execution, so a researcher can test their program on sample inputs for trivial input values while symbolically abstracting the more crucial for more rigorous testing. Two examples of these hybrid approaches are:

- *Concolic Testing*: Or, Directed Automated Random Testing
  - *Advantages*:
    - Maintains a concrete state and a symbolic state.
    - Concrete state maps all variables to their concrete values.
    - Symbolic state maps only variables that have non-concrete values.
  - *Disadvantages*:
    - Requires initial concrete values for inputs.
    - Could be constrained by limitations of constraint solver
- *Execution-Generated Testing*
  - *Advantages*:
    - Intermixes concrete and symbolic execution by dynamically checking if all values are concrete
    - If all concrete, use actual program execution; otherwise, perform symbolically
  - *Disadvantages*:

C. Challenges and Solutions

D. Tools

Here we have an initial section discussing tomatoes.

References

[1] C. Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," *Commun. ACM*, vol. 56, no. 2, pp. 82–90, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408795