Concolic Testing: A Decade Later (Keynote)

Koushik Sen
EECS Department, UC Berkeley, CA, USA.
ksen@cs.berkeley.edu

Abstract

Symbolic execution for software testing has witnessed renewed interest in the recent years due to its ability to generate high-coverage test suites and find deep errors in software systems. In this talk, I will give an overview of a modern symbolic execution technique, called concolic testing, discuss its key challenges in terms of path exploration, and introduce MultiSE, a new technique for tackling the path exploration challenge.

Categories and Subject Descriptors D.2.5 [*Software Engineering*]: Testing and Debugging

General Terms Symbolic execution, Testing tools, Debugging aids

Keywords symbolic execution, MultiSE, value summary, JavaScript, test generation, concolic testing, Jalangi

1. Introduction

Symbolic execution, which was introduced more than four decades ago, is typically used in software testing to explore as many different program paths as possible in a given amount of time, and for each path to generate a set of concrete input values exercising it, and check for the presence of various kinds of errors including assertion violations, uncaught exceptions, security vulnerabilities, and memory corruption. A key limitation of classical symbolic execution is that it cannot generate useful test inputs if the program under test uses complex operations such as pointer manipulations and non-linear arithmetic operations.

Our research on Concolic Testing [2, 3, 6] (also known as DART: Directed Automated Random Testing or Dynamic Symbolic Execution) alleviated the limitations of classical symbolic execution by combining concrete execution and symbolic execution. We demonstrated that concolic testing is an effective technique for generating high-coverage test suites and for finding deep errors in complex software applications. The success of concolic testing in scalably and exhaustively testing real-world software was a major milestone in the ad hoc world of software testing and has inspired the development of several industrial and academic automated testing and security tools.

One of the key challenges of concolic testing is the huge number of programs paths in all but the smallest programs, which is usually exponential in the number of static branches in the code. We have been working on a variety of techniques [1, 4, 5, 7] to make automated test generation scalable and exhaustive for large programs. We have developed hybrid concolic testing [4], a combination of random testing, a fast and non-exhaustive method of testing, with concolic testing, an exhaustive and slow testing technique. We have also developed a novel strategy where concolic test generation is guided by the static control flow graph of the program under test to quickly achieve high code coverage [1]. We have proposed lazy test generation [5], an approach similar to the counterexample-guided refinement paradigm from static software verification. The technique first explores, using concolic testing, an abstraction of the function under test by replacing each called function with an unconstrained input.

In this talk I will describe MultiSE [7], a new technique for merging states incrementally during symbolic execution, without using auxiliary variables. The key idea of MultiSE is based on an alternative representation of the state, where we map each variable, including the program counter, to a set of guarded symbolic expressions called a value summary. MultiSE has several advantages over conventional symbolic execution and state merging techniques: 1) value summaries enable sharing of symbolic expressions and path constraints along multiple paths, 2) value-summaries avoid redundant execution, 3) MultiSE does not introduce auxiliary symbolic values, which enables it to make progress even when merging values not supported by the constraint solver, such as floating point or function values. We have implemented MultiSE for JavaScript programs in a publicly available open-source tool. Our evaluation of MultiSE on several programs shows that MultiSE can run significantly faster than traditional symbolic execution.

WODA'15, October 26, 2015, Pittsburgh, PA, USA
ACM. 978-1-4503-3909-4/15/10
http://dx.doi.org/10.1145/2823363.2823364

Acknowledgements

The work is supported in part by NSF grants CCF-1017810, CCF-0747390, CCF-1018729, CCF-1423645, CCF-1409872, and CCF-1018730, and gifts from Samsung and Mozilla.

References

[1] J. Burnim and K. Sen. Heuristics for scalable dynamic test generation. In *ASE'08*, pages 443–446. IEEE, 2008.

[2] C. Cadar and K. Sen. Symbolic execution for software testing: Three decades later. *Communications of the ACM (CACM)*, 56(2):82–90, February 2013.

[3] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *PLDI'05*, pages 213–223, 2005.

[4] R. Majumdar and K. Sen. Hybrid concolic testing. In *ICSE'07*, pages 416–426. IEEE, 2007.

[5] R. Majumdar and K. Sen. Latest : Lazy dynamic test input generation. Technical Report UCB/EECS-2007-36, EECS Department, University of California, Berkeley, Mar 2007.

[6] K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *ESEC/FSE'05*, pages 263–272. ACM, 2005.

[7] K. Sen, G. Necula, L. Gong, and W. Choi. Multise: Multi-path symbolic execution using value summaries. In *ESEC/FSE'15*. ACM, 2015.