Concolic Testing and Constraint Satisfaction

Koushik Sen
ksen@cs.berkeley.edu
EECS Department, University of California, Berkeley, CA, USA.

Software testing is the most common technique used in industry to improve reliability and quality of software. Unfortunately, testing is mostly a manual process that reportedly accounts for over half of the typical cost of software development and maintenance. Symbolic execution [6, 2, 3, 9, 5] was proposed in the 70s to automate software testing by generating test inputs. During symbolic execution, the program is run with symbolic (rather than concrete) inputs and generates a *path constraint*. This path contraint is updated whenever a conditional statement is executed and encodes the constraints on the input necessary to reach a given program point. Test generation is performed by solving the collected constraints using a constraint solver.

Although symbolic execution was proposed almost 35 years ago, we have hardly seen any practical test generation tool based on this technique. There are two key reasons behind this: 1) until recently, constraint solving techniques were not powerful enough to solve constraints that arise during symbolic execution of most real-world programs, and 2) constraints generated during symbolic execution of real-world programs often fall under theories that are not decidable. The first issue has been addressed by the recent advances in SAT and SMT solving techniques.

In this talk, I will describe *concolic testing* [4, 12, 10, 11, 7] (also known as *directed automated random testing* or *dynamic symbolic execution*), a technique that addressed the second challenge associated with symbolic execution and thus paved the way for development of practical automated test generation tools. Concolic testing improves classical symbolic execution by performing symbolic execution of a program along a concrete execution path. Specifically, concolic testing executes a program starting with some given or random concrete input. It then gathers symbolic constraints on inputs at conditional statements during the execution induced by the concrete input. Finally, a constraint solver is used to infer variants of the concrete input to steer the next execution of the program towards an alternative feasible execution path. This process is repeated systematically or heuristically until all feasible execution paths are explored or a user-defined coverage criteria is met.

A key observation in concolic testing is that *intractability in symbolic execution can be alleviated using concrete values*: whenever symbolic execution generates a constraint that is beyond a decidable theory, one can simplify this constraint by replacing some of the symbolic values with concrete values. In these cases, the concolic execution degrades gracefully by leveraging concrete values to keep the path constraint decidable.

Concolic testing and its variants are now the underlying technique of several popular testing tools: UIUC's CUTE and jCUTE[1], Stanford's KLEE[2] tool uses an approach similar to concolic testing, UC Berkeley's CREST[3] and BitBlaze[4], UCLA's SPLAT [8]. Concolic testing technology is now used in industrial practice at Microsoft (Pex[5], YOGI[6]) and IBM (Apollo [1]).

Acknowledgements

This research supported in part by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227), by NSF Grants CNS-0720906, CCF-0747390, CCF-1018729, and CCF-1018730, and by a Sloan Foundation Fellowship.

References

1. S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in dynamic web applications. In *ISSTA'08*, July 2008.
2. R. S. Boyer, B. Elspas, and K. N. Levitt. SELECT – a formal system for testing and debugging programs by symbolic execution. *SIGPLAN Not.*, 10:234–245, 1975.
3. L. A. Clarke. A program testing system. In *Proc. of the 1976 annual conference*, pages 488–491, 1976.
4. P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *PLDI'05*, June 2005.
5. W. Howden. Symbolic testing and the DISSECT symbolic evaluation system. *IEEE Transactions on Software Engineering*, 3(4):266–278, 1977.
6. J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19:385–394, July 1976.
7. R. Majumdar and K. Sen. Hybrid concolic testing. In *ICSE'07*, May 2007.
8. R. Majumdar and R.-G. Xu. Reducing test inputs using information partitions. In *CAV'09*, pages 555–569, 2009.
9. C. Ramamoorthy, S.-B. Ho, and W. Chen. On the automated generation of program test data. *IEEE Trans. on Software Engineering*, 2(4):293–300, 1976.
10. K. Sen. *Scalable Automated Methods for Dynamic Program Analysis*. PhD thesis, University of Illinois at Urbana-Champaign, June 2006.
11. K. Sen and G. Agha. CUTE and jCUTE : Concolic unit testing and explicit path model-checking tools. In *CAV'06*, 2006.
12. K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *ESEC/FSE'05*, Sep 2005.

[1] `http://osl.cs.uiuc.edu/~ksen/cute/`
[2] `http://klee.llvm.org/`
[3] `http://code.google.com/p/crest/`
[4] `http://bitblaze.cs.berkeley.edu/`
[5] `http://research.microsoft.com/en-us/projects/pex/`
[6] `http://research.microsoft.com/en-us/projects/yogi/`