Project Proposal

Xusheng Xiao
xxiao2@ncsu.edu

Project Topic:

Symbolic Execution in Software Engineering.

Goal:

Symbolic execution [8] is a form of program analysis that uses symbolic values instead of concrete values as inputs and symbolic expressions to represent the values of program variables. As a result, the outputs computed by a program are expressed as an expression of the symbolic inputs. This technique is widely used in automatic test data and test case generation in software engineering. In this project, we will provide a study of how finite state automaton, context-free grammer are used in assisting symbolic execution to generate test data and test cases more effectively. We also provide a study about the techniques that are used to alleviate the path explosion problem and optimize the performance of symbolic execution.

Project Description:

Instead of feeding normal input to programs, the symbolic execution proceeds with symbolic representing values[8]. In this project, we intend to explore the application of computation theory in this field and provide a detailed survey of the existing tools and techniques associating with symbolic execution in software engineering.

1. **Finite State Automaton: Regular Expression Operation.** Dynamic symbolic execution (DSE) is a variation of static symbolic execution. DSE performs a symbolic execution of the program by assigning symbolic variables to each program input and executes the program starting with arbitrary inputs. During the execution, DSE collects symbolic constraints on inputs obtained from predicates in branch statements along the execution. Then a constraint solver is used to compute new inputs in order to execute the program along different execution paths. Existing dynamic symbolic execution (DSE) [5, 2, 10] techniques can effectively generate test inputs for various programs to achieve high coverage. However, when dealing with the programs that use regular expression operations, the logic complexity makes it much more difficult for the DSE engine to achieve high branch coverage of the program under test within limited time and resources. With the help of finite state automaton, DSE is guided to achieve satisfactory branch coverage rate [9].

2. **Context-Free Grammar: Structured Inputs Generation.** Symbolic execution for programs can be effectively reduced to a constraint-generation phase followed by a constraint-solving phase. Symbolic execution has trouble creating test cases that achieve high coverage for programs that expect structured inputs, such as those that require input strings from a context-free grammar. Kiezun et al. [7] propose Hampi to create grammar-based input constraints and then fed those into symbolic execution engine to generate test cases for programs. Given a set of constraints over a string variable, Hampi constraints express membership in regular and fixed-size context-free languages and output a string that satisfies all the constraints. Their evaluation results show that using Hampi can improve the effectiveness of symbolic execution. We plan to discuss the details in our project.

3. **Computability and Computation Complexity: Exploration Of Program.** Dynamic symbolic execution (DSE) [5, 2, 10], is adopted by automatic test case generation tool to explore the feasible paths of the program under test for achieving high coverage. In theory, all feasible execution paths will be exercised eventually through the iterations of

constraint collection and constraint solving in DSE. A program under test can be modeled as a control flow graph (CFG) [1], whose nodes represent simple primitive statements (such as input, output, and assignment) and edges represent the flow of control. An execution path of the program is a path on CFG from the starting node, entry of the program, to the exit node, exit of the program. Thus, to explore all paths of the program under test, i.e. achieving 100% path coverage, is to enumerate all paths between two nodes in a graph, which is well known as a NP-hard problem [6]. To alleviate this path explosion problem and to reduce the computation complexity, different techniques has been proposed, such as performing symbolic execution compositionally [4], selective symbolic execution [3] and fitness-guided path exploration [11]. In our project, we plan to discuss these techniques and provide example issues that can be solved using these techniques. Additionally, we plan to discuss the existing issues that are still not solved in the area of symbolic execution.

References

[1] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, Cambridge, UK, 2008.

[2] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. Exe: automatically generating inputs of death. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 322–335, New York, NY, USA, 2006. ACM.

[3] Vitaly Chipounov, Vlad Georgescu, Cristian Zamfir, and George Candea. Selective Symbolic Execution. In *Proceedings of the 5th Workshop on Hot Topics in System Dependability (HotDep)*, 2009.

[4] Patrice Godefroid. Compositional dynamic test generation. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–54, New York, NY, USA, 2007. ACM.

[5] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *PLDI*, pages 213–223, 2005.

[6] Jonathan Gross and Jay Yellen. *Graph theory and its applications*. CRC Press, Inc., Boca Raton, FL, USA, 1999.

[7] Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. Hampi: a solver for string constraints. In *ISSTA '09: Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 105–116, New York, NY, USA, 2009. ACM.

[8] James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.

[9] Nuo Li, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, and Wolfram Schulte. Reggae: Automated test generation for programs using complex regular expressions. In *Proc. 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, November 2009.

[10] Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 263–272, New York, NY, USA, 2005. ACM.

[11] Tao Xie, Nikolai Tillmann, Peli de Halleux, and Wolfram Schulte. Fitness-guided path exploration in dynamic symbolic execution. In *Proc. the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*, June-July 2009.