Automated Student Code Assessment with Concolic Testing Thesis Related Work

Karl Bell

June 8, 2012

1 Related Work

There has been a large amount of research in the areas of combined concrete and symbolic execution as a way of generating test inputs [4, 8, 3]. However, the application of these techniques to the domains of education and code comparison has not been explored nearly as well. There are a number of tools aimed at automatic assessment of student programs but these usually rely on professor-generated test data[6]. There has been only one tool which I could find that tried to automatically generate test inputs for student programs, but it was never released[5].

1.1 Dynamic Test Generation

There has been a large amount of work in the area of automated test generation. The tools which are most relevant to this thesis are those using a system of combined symbolic and concrete execution known as dynamic test generation or "concolic" testing, as that is what I plan to implement my tool with. A number of tools exist which implement this test-generation strategy[8, 4, 9].

1.1.1 Symbolic Execution

Symbolic execution is the process of running a program on symbolic inputs rather than real inputs. It consists of using branch points in the program to construct a set of path constraints for each possible program path. These path constraints are actually just a set of constraints which define sets of inputs which will cause the same path of execution to be followed[4].

1.1.2 Concolic Testing

Concolic testing, or dynamic test generation, extends this by interleaving concrete execution so that portions of a program which are difficult to reason about symbolically can be abstracted out by using actual values in these portions instead of symbolic ones[3].

There have been numerous advances in the area of dynamic test generation in the past few years which have increased its effectiveness. Many new strategies and algorithms have made the technique much faster and more able to target relevant areas of code[3, 1, 7]. Many of these appraoches could be integrated into my work to increase the efficacy of my test data generation. One such approach is function summarization[3], which generalizes concolic execution over function calls by creating logical summaries of those functions to be included in the path constraint, rather than adding all of the constraints of the function to the path constraint and taking the time to actually run through the function.

1.1.3 Java PathFinder

For this thesis, I will be utilizing and extending the Java PathFinder, an open source tool maintained by the NASA Ames Research Center[10]. Java PathFinder (JPF) functions by providing a reimplementation of

the JVM which allows developers using the framework to instrument the running of Java programs at the bytecode level. Java PathFinder includes a module for symbolic and concolic test generation, which I will be using for this project. The extent to which I will modify JPF's symbolic framework is unknown at this point in time.

1.2 Automated Assessment

A useful survey of automated assessment tools was performed in [6]. This survey described trends among a number of different automated testing tools. This survey showed the different approaches which others have taken with these testing tools. Most relied on Unit testing, testing frameworks, scripting, and output comparison.[6]

1.2.1 Tools

One such tool is Web-CAT, which tries to get students to generate tests for their own code, but also provides tools for instructors to automate other testing of student code. Since Web-CAT is extensible, it may be worthwhile to attempt to adapt the tool built as part of this thesis to it.

WebIDE, a teaching platform developed at Cal Poly, also incorporates automated assessment of student code correctness[2]. It works by sending code to "evaluators" which are actually just web servers which compile and run this code, testing with whatever unit tests the instructor provides. There are also currently limited evaluators for output comparison.

1.2.2 My Work

It is my intention to create a tool that can be used independently or as as an evaluator for the WebIDE platform, which will allow the professor to provide only a reference implementation in order to test student code. It will use this reference implementation along with concolic input space exploration to develop a suite of tests which tests student code fully, using the instructor code as a test oracle. This has advantages over the current approach which requires instructors to generate their own test cases, as instructor-generated test cases will not take into account the structure of student submissions as they must by their nature be written before the student submits their code for evaluation.

1.3 Test Data Generation in Education

A similar form of test input generation for assessing student submissions has been tried once before, as detailed in [5]. This tool was built using an early version of JPF's symbolic testing framework which was not publicly available at the time. As such, the tool was not made available. In addition, the symbolic framework of JPF has since changed a large amount, meaning the work done in [5] would no longer be compatible with the current JPF codebase.

All of this means that this tool will not be useful in creating the project my thesis is based on. However, the lessons learned in its creation which are detailed in the paper may be. The paper explains a number of strategies that can be used with JPF for the purpose of generating tests, including symbolic method sequence exploration, which can be used to generate a number of possible method call sequences for an object which would be useful for generating test data, and generalized symbolic execution with lazy initialization, which relies on annotation of classes to generate valid method call sequences. While annotation would not be something I could add automatically, the author also has some suggestions in this regard which I plan to try.

This paper, while creating a tool somewhat related to the one I plan to build, still focused more on strategies for test generation, rather than on a description and evaluation of the tool itself. In my thesis, I plan to construct such a tool and make it available for instructors using the WebIDE platform. I also plan to examine the effectiveness of the tool for evaluating the correctness of student code, and determine what kind of feedback will be most helpful for students to receive from the tool.

References

[1] Anand, S., Godefroid, P., and Tillmann, N. Demand-driven compositional symbolic execution. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems* (Berlin, Heidelberg, 2008), TACAS'08/ETAPS'08, Springer-Verlag, pp. 367–381.

[2] Dvornik, T., Janzen, D. S., Clements, J., and Dekhtyar, O. Supporting introductory test-driven labs with webide. In *Proceedings of the 2011 24th IEEE-CS Conference on Software Engineering Education and Training* (Washington, DC, USA, 2011), CSEET '11, IEEE Computer Society, pp. 51–60.

[3] Godefroid, P. Compositional dynamic test generation. *SIGPLAN Not. 42*, 1 (Jan. 2007), 47–54.

[4] Godefroid, P., Klarlund, N., and Sen, K. Dart: directed automated random testing. *SIGPLAN Not. 40*, 6 (June 2005), 213–223.

[5] Ihantola, P. Test data generation for programming exercises with symbolic execution in java pathfinder. In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006* (New York, NY, USA, 2006), Baltic Sea '06, ACM, pp. 87–94.

[6] Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), Koli Calling '10, ACM, pp. 86–93.

[7] Păsăreanu, C. S., Rungta, N., and Visser, W. Symbolic execution with mixed concrete-symbolic solving. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2011), ISSTA '11, ACM, pp. 34–44.

[8] Sen, K., Marinov, D., and Agha, G. Cute: a concolic unit testing engine for c. *SIGSOFT Softw. Eng. Notes 30*, 5 (Sept. 2005), 263–272.

[9] Tillmann, N., and de Halleux, J. Pex–white box test generation for .net. In *Tests and Proofs*, B. Beckert and R. Hhnle, Eds., vol. 4966 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, pp. 134–153.

[10] Visser, W., Păsăreanu, C. S., and Khurshid, S. Test input generation with java pathfinder. *SIGSOFT Softw. Eng. Notes 29*, 4 (July 2004), 97–107.