Automated Student Code Assessment with Concolic Testing Thesis Validation

Karl Bell

June 8, 2012

1 Validation Summary

There are a number of ways which I could validate the work of my thesis. My plan is to create a tool which will allow professors to have student code automatically validated by providing only a reference implementation. This tool will also provide feedback to students about what may be wrong with their program in the case that it fails this validation.

1.1 Code Coverage

My project will rely on dynamic test generation to create test inputs to run student programs with (using the instructor reference implementation as a test oracle). I plan to implement this using the Java PathFinder, an open source tool maintained by the NASA Ames Research Center[5]. It may be necessary for me to add on to the test generation capabilities of the plaform to make my project viable. Even if this is not the case, it would be useful to test the code coverage of student code which is acheived by the tests my tool generates against that of other test generation tools like jCUTE[4].

1.2 Evaluation of Code Correctness

The most important aspect of my thesis work will be its ability to determine the correctness of a student solution to a programming exercise. To assess this ability, I plan to create a number of labs for the WebIDE platform[1] which will be offered to students currently enrolled in beginning programming courses at Cal Poly. The programs (or program portions) which students write for these labs will be evaluated using the work of this thesis. All of the student submissions will also be recorded to be evaluated for correctness by a manually constructed test suite. This will allow my tool to be validated against the current approach used by most automated assessment frameworks[3]. The hope is that my tool will not report any submissions as correct for which the manually-written tests would fail. If my tool is able to catch errors in student code which the manually-written tests would not, this would be further evidence of its efficacy.

1.3 Constructive Feedback

Finally, it may be worthwhile to assess the effectiveness of the feedback which my tool can provide. With the path constraints created with symbolic execution, it may be possible to give the student feedback which defines the conditions under which their code fails in the abstract (e.g., "Your code fails when x ¿ y") rather than stating exact values (e.g., "You code fails when x = 5 and y = 4"). To evaluate which of these approaches might be better, I may try both with different sets of students to see which group finds the messages more helpful.

Current research suggests that the abstract information may be more helpful[2].

References

[1] 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.

[2] 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.

[3] 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.

[4] Sen, K., and Agha, G. Cute and jcute: Concolic unit testing and explicit path model-checking tools. In *Computer Aided Verification*, T. Ball and R. Jones, Eds., vol. 4144 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 419–423.

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