CS569: Competitive Milestone #1

In my original project proposal, I aimed to implement a version of Adaptive Random Testing called Mirror Adaptive Random Testing. However, it was revealed that some of the functionality for implementing this algorithm might not be available in TSTL. Therefore, I have decided to go with a simpler and improved version of random testing. This algorithm borrows a lot of concepts that we have studied in class. Due to its simple nature, it may not be as effective as Adaptive Random Testing. For this milestone, I have decided keep things simple and form a basis for the Adaptive Random Testing algorithm. My goal over the next few days will be to improve this algorithm and implement Adaptive Random Testing using it. In addition to that, I would also like to explore how code coverage can be used to guide test generation.

The algorithm consists of the following stages: a random action is generated based on a random seed. The action is performed. If the performed action results in a failure, the failure is printed and is saved to a file. The algorithm is finished when the terminating condition is reached which can be if the timeout is reached. As of now, the algorithm is a simple random tester. Initially, my goal was to implement Adaptive Random Testing which I am working on the side. Unfortunately, I could not get the Adaptive Random Testing algorithm to work correctly so I decided to implement a random tester which will form the base for Adaptive Random Testing. By my next milestone, I hope to further improve this algorithm in two ways: improve the random testing and if time permits implement the Adaptive Random Testing. My goal would also be to provide a comparison of the improved algorithm to Random Testing algorithm discussed in class.

The tester requires a few different parameters as inputs from command line. These parameters include timeout, seed, depth, width, faults, coverage and running. Timeout is the time for how long the test needs to be run. Seed is used to determine the random generator seed. If the fault option is enabled, the failed test is saved to a file. Coverage enables the final results to be displayed such as branch count and statement count. Running info allows branch information to be displayed as the tests are performed.

As I have mentioned before, this tester is a very minimalistic random tester. In the next few days, I will implement a version of Adaptive Random Testing using this algorithm as the base. Also, I would like to study how we can analyze the information produced to provide a better approach to software testing. Overall, the next milestone will be the one that has a lot of improvement. If I cannot implement the Adaptive Random Testing algorithm, I will continue working on this Random Tester and improve it to see what type of performance benefits can be obtained.

References:

[1] T. Y. Chen, F. C. Kuo, R. G. Merkel and S. P. Ng, "Mirror adaptive random testing," Quality Software, 2003. Proceedings. Third International Conference on, 2003, pp. 4-11.

[2] F.-C. Kuo ,An Indepth Study of Mirror Adaptive Random Testing,&rdquo, Proc. Ninth Int',l Conf. Quality Software, pp. 51-58, 2009.

[3] T. Y. Chen, "Adaptive Random Testing," 2008 The Eighth International Conference on Quality Software, Oxford, 2008, pp. 443-443.

[4] Z. Q. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual, Seoul, 2010, pp. 208-213.