

# Program Slicing with Test Data Generation

## Competitive Milestone 2

Punyapich Limsuwan  
May 18, 2016

### I. BACKGROUND

Previously, I proposed to implement program slicing with the random test generator. In general, the idea of this algorithm is to reduce search space by constraining the test suite before generating random data. Slicing criteria need to be defined and the program will be executed once to check if variables are affected. Therefore, the variables that are affected will be included in the slice. Then, slice will be used to identify constraints for generating test data. The algorithm for test data generation requires specific goal statement, which will be satisfied by generated random input from the slice's constraints.

### II. ALGORITHM

To implement program slicing with the given `randomtester.py` and `sut.py`, I have to think it differently because there is neither variable nor statement that I could focus on. The random tester runs several actions and check if there are existing failures, also performs reduction, normalization and generalization if needed. Thus, the slicing may not be applied easily since we didn't directly generate test cases over `avl.py`. I have come up with the idea to be using the general concept of program slicing with a little bit of adjustment. Thus, I need to reduce the search space by using a constraint while randomly generating test cases. In our case, I will focus on generating actions in order to find the suspect actions that may cause the failure. Instead of enabling all actions, limiting to the small set of actions at first allows us to verify that given test cases are either bug-free or buggy. Furthermore, the set of actions will be expandable, if the failures are not found in the limited amount of time, the size of set will grow larger, until it reaches the size that contains all actions. However, if the failure is found within the time, the size of set will stay the same for another amount of time to see whether it will produce new failure or the same failure, if it produces the new failures the time will be expanded until timeout or find the same failure. From my aspect, this algorithm may take longer than the generic random testing to have code coverage, but I hope that it is more efficient to locate failures, and also easier to see the test cases that cause the failure came from which actions. The function in `sut.py` that I use for this algorithm is `randomEnableds`, which allows me to specify number of actions in the list. Thus, when it constrains is satisfied by time, the input number for `randomEnableds` will be increased, until it reaches the maximum number defined.

### III. PROGRESS

In this milestone, I wanted to measure how action set size affects the coverage. So I have changed the major constraints, which are the action time tolerance, number of initial set size, and number of maximum set size. The default of tor action time tolerance is 1 sec. When I increase this tolerance to 3, 5 and 7, the failure found reduces to 5, 5 and 4, respectively. When I decreased the tolerance to 0.5 sec, the failure found also decrease to 4. Thus, I assume that 1 second is still the optimum tolerance, so far. However, I test the same procedure with another random seed, the action time tolerance does not affect the failure found at all. In previous milestone, I set the initial set size to ten. In this milestone, I started by decrease initial size to two, the result was significant improve. It increased from 1 failure found to 16 failures found. When I increased the initial set size to higher number as 30, 50, and 70. The failure found stay as one or none. Thus, I would prefer to initial the set size as low as 2-10. And the last constraint I tested is the the maximum set size. Actually this constraint depends on timeout and action time tolerance, which means that, I could set the maximum set size to higher number for the higher time out. But if the timeout is not too long, the set size will not be reaching to the maximum number, unless I changed the tolerance time. Therefore, what I've found here is that the maximum set size does not affect too much by the limited timeout. Thus, I will probably let the maximum set size be as large as possible.

### IV. FUTURE WORK

Up to this point, I have learned that reducing randomness of test case could find more failure. The next step, I want to test it on several condition, such as depth, width, and most importantly timeout. I strongly believe that this could find some more bugs.