Part 2

CS 569 Spring 2016

Omkar Thakur

932-704-163

In my approach the FCFS and the mirror adaptive random testing was going to be implemented. But it was revealed that the implementation may not work in all the aspects. My main aim is also to improve random testing with respect to combination logic. Each and every path of the algorithm has to be visited in order to be proficient. But the main problem occurring is that each node is not visited.

Bugs were not found in this part of the implementation but a number of branches were covered effectively. The basic implementation of FCFS was tried. First the system is restarted with the help of the sut.restart(). Then after each action of range, the random action is taken place. The efficiency of the algorithm has also to be seen. The current implementation does not look as efficient. The next milestone of this implementation will be tried to be more efficient. Much of the functions of the sut were a major help in the making of tester. For example the random action which has to be made after each major action. In this implementation, the length as well as breadth is tried on for loop. I have tried to keep the implementation simple for the first deliverable.

I have used the following for taking the values. I have used parser for this purpose. The following are the categories:

1. timeout

2. seed

3. depth

4. width

5. faultcheck

6. coverage_report

7. detail_of_running

With sut.internalreport , the final execution is published. I have also tried to increase the failure counts with each step in the failure loop. The prettyprinttest will print the results in a more efficient way. Therefore, more such functions were useful in implementing.

Input:

Python2.7 tester1.py 30 1 100 1 0 1 1

Output:

/avl.py ARCS: 197 [(-1, 6), (-1, 11), (-1, 16), (-1, 31), (-1, 35), (-1, 44), (-1, 55), (-1, 70), (-1, 85), (-1, 111), (-1, 136), (-1, 148), (-1, 159), (-1, 171), (-1, 184), (-1, 233), (-1, 254), (6, -5), (11, 12), (12, 13), (13, -10), (16, 17), (16, 18), (17, -15), (18, 19), (18, 20), (19, -15), (20, 21), (21, 23), (23, 24), (24, 25), (25, 26), (25, 27), (26, -15), (27, 28), (27, 29), (28, -15), (29, -15), (31, -30), (35, 36), (36, 37), (37, 39), (39, -34), (39, 40), (40, -34), (40, 41), (41, 40), (44, 45), (44, 46), (45, -43), (46, 47), (46, 48), (47, -43), (48, 49), (48, 50), (49, -43), (50, 51), (50, 52), (51, -43), (52, -43), (55, -54), (70, 72), (72, 73), (72, 75), (73, -69), (75, 76), (75, 78), (76, -69), (78, 79), (78, 80), (79, -69), (80, 81), (81, -69), (85, 87), (87, 89), (89, 90), (89, 95), (90, 91), (91, 92), (92, 93), (93, 104), (95, 96), (95, 98), (96, 104), (98, 99), (98, 102), (99, 104), (102, 104), (104, -84), (111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (114, 123), (115, 116), (115, 119), (116, 117), (117, 118), (118, 119), (119, 120), (120, 121), (121, 123), (123, 113), (123, 124), (124, 125), (124, 128), (125, 126), (126, 127), (127, 128), (128, 129), (129, 130), (130, 113), (136, 137), (137, 138), (138, 139), (139, 141), (141, 142), (142, 143), (143, -134), (148, 149), (149, 150), (150, 151), (151, 153), (153, 154), (154, 155), (155, -146), (159, 160), (159, 168), (160, 161), (160, 166), (161, 162), (162, 163), (163, 164), (164, 166), (166, -158), (168, -158), (171, 172), (171, 180), (172, 173), (172, 178), (173, 174), (174, 175), (175, 176), (176, 178), (178, -170), (180, -170), (184, 185), (184, 214), (185, 186), (185, 207), (186, 187), (187, 188), (187, 190), (188, 205), (190, 191), (190, 192), (191, 205), (192, 193), (192, 197), (193, 205), (197, 198), (198, 199), (199, 200), (200, 203), (203, 205), (205, 206), (206, -182), (207, 208), (207, 209), (208, 212), (209, 210), (210, 212), (212, -182), (214, -182), (233, 234), (234, 236), (236, 237), (237, 238), (238, 239), (238, 241), (239, -229), (241, 236), (254, 255), (254, 257), (255, -253), (257, 258), (258, 259), (259, 260), (259, 262), (260, 259), (262, 264), (264, 265), (265, 266), (265, 268), (266, 265), (268, -253)]

/home/omkar29/Desktop/cs569s16/tstl/examples/AVL/avl.py LINES: 145 [6, 11, 12, 13, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 31, 35, 36, 37, 39, 40, 41, 44, 45, 46, 47, 48, 49, 50, 51, 52, 55, 70, 72, 73, 75, 76, 78, 79, 80, 81, 85, 87, 89, 90, 91, 92, 93, 95, 96, 98, 99, 102, 104, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 130, 136, 137, 138, 139, 141, 142, 143, 148, 149, 150, 151, 153, 154, 155, 159, 160, 161, 162, 163, 164, 166, 168, 171, 172, 173, 174, 175, 176, 178, 180, 184, 185, 186, 187, 188, 190, 191, 192, 193, 197, 198, 199, 200, 203, 205, 206, 207, 208, 209, 210, 212, 214, 233, 234, 236, 237, 238, 239, 241, 254, 255, 257, 258, 259, 260, 262, 264, 265, 266, 268]

TSTL BRANCH COUNT: 197

TSTL STATEMENT COUNT: 145

Future work

My future work will be to implement such a tester in a way that, the combinational logic is explored. The entire failed test is saved and it is compared with newly generated test. It also does not find any bugs, therefore this will be given paramount importance in the coming implementation. Another goal will be to compare this algorithm to the Dr Groce's Random Tester.

Reference

[1] 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.

[2]. A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, “An Experimental Determination of Sufficient Mutant Operators,” ACM Transactions on Software Engineering and Methodology, vol. 5, no. 2, pp. 99–118, April 1996.

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

[4]. P. S. Loo and W. K. Tsai. “Random testing revisited”. Information and Software Technology, Vol.30. No 7, pp. 402-417, 1988.