CS 569 PROJECT

Xuhao Zhou

Jun 6, 2016

1. Introduction

In automated test generation, a test harness defines the set of valid tests for a system, and usually also defines a set of correctness properties. Automated test generation and model checking are major difficulties in writing test harnesses. Groce et al. [1] presents TSTL, the Template Scripting Testing Language, a domain-specific language (DSL) for writing test harnesses. TSTL compiles harness definitions into a graph-based interface for testing, making generic test generation and manipulation tools that apply to any SUT possible. Automated generation of tests relies on the construction of test harnesses. A test harness defines the set of valid tests (and, usually, a set of correctness properties for those tests) for the Software Under Test (SUT). A TSTL harness defines a template for action definition, and the compiler instantiates the template exhaustively. TSTL enables a declarative style of test harness development, focuses on defining the actions in valid tests, produces a SUT interface and makes it possible for users to easily apply different test generation algorithms to the same system without much effort. There are two concepts we need to understand: "First, at any state of the system, the only actions that are enabled are those that do not use any non-initialized pool values. Second, a value that has been initialized cannot be initialized until after at least one action that uses it has been executed. [1]" According to [2], "in some circumstances, random testing methods are more practical than any alternative, because information is lacking to make reasonable systematic test-point choices." Therefore, this project implements a tester to generate improved random test cases using the TSTL test harness.

2. Algorithm

In order to improve the performance of the regular random testing method, this tester adapts the idea from the class and the paper *Feedback-directed Random Test Generation* [3] to implement an efficient test case generation algorithm. The general idea is to supervise the process of the action generation and avoids the action generator to generate redundant actions. Our algorithm is based on the GenerateSequences algorithm [3].

```
NewGenerateSequences (contracts, timeLimit)
1.  errorSeqs $\leftarrow$ {}        // Their execution violates a contract.
2.  nonErrorSeqs $\leftarrow$ {} // Their execution violates no contract.
3.  newSeq $\leftarrow$ {}          // New sequence
4.  while timeLimit not reached do
5.      // Generate new action.
6.      action $\leftarrow$ sut.randomEnabled(rgen)
7.      newSeq $\leftarrow$ newSeq + action
8.      // Discard duplicates.
9.      for s in 0 $\rightarrow$ depth/2
10.         if newSeq $\notin$ nonErrorSeqs $\cup$ errorSeqs then
11.             break
12.         else
13.             // Remove the old action from newSeq
14.             newSeq $\leftarrow$ newSeq – action
15.             // Generate new action.
16.             action $\leftarrow$ sut.randomEnabled(rgen)
17.             // Append the new action to newSeq
18.             newSeq $\leftarrow$ newSeq + action
19.         end if
20.     // Execute new action and check contracts.
21.     $\langle$o, violated$\rangle$ $\leftarrow$ execute(action, contracts)
22.     // Classify new sequence and outputs.
23.     if violated = true then
24.         errorSeqs $\leftarrow$ errorSeqs $\cup$ {newSeq }
25.     else
26.         nonErrorSeqs $\leftarrow$ nonErrorSeqs $\cup$ {newSeq }
27.     end if
28. end while
29. return <nonErrorSeqs, errorSeqs>
```

3. Implementation

(1) There are some arguments can be read from the command line. (a) timeout: time in seconds can be used for testing. (b) seed: seed for Python Random.random object used for random number generation in the code. (c) depth: maximum length of a test generated by the algorithm. (d) width: maximum memory/BFS queue/other parameter that is basically a search width. (e) faults: either 0 or 1 depending on whether tester should check for faults in the SUT; if true, the tester saves a test case for each discovered failure (terminating the test that generated it), in the current directory, as failure1.test failure2.test, etc. (f) coverage: either 0 or 1 depending on whether a final coverage report should be produced. (g) running: either 0 or 1 depending on whether running info on branch coverage should be produced.

(2) Code Explanation

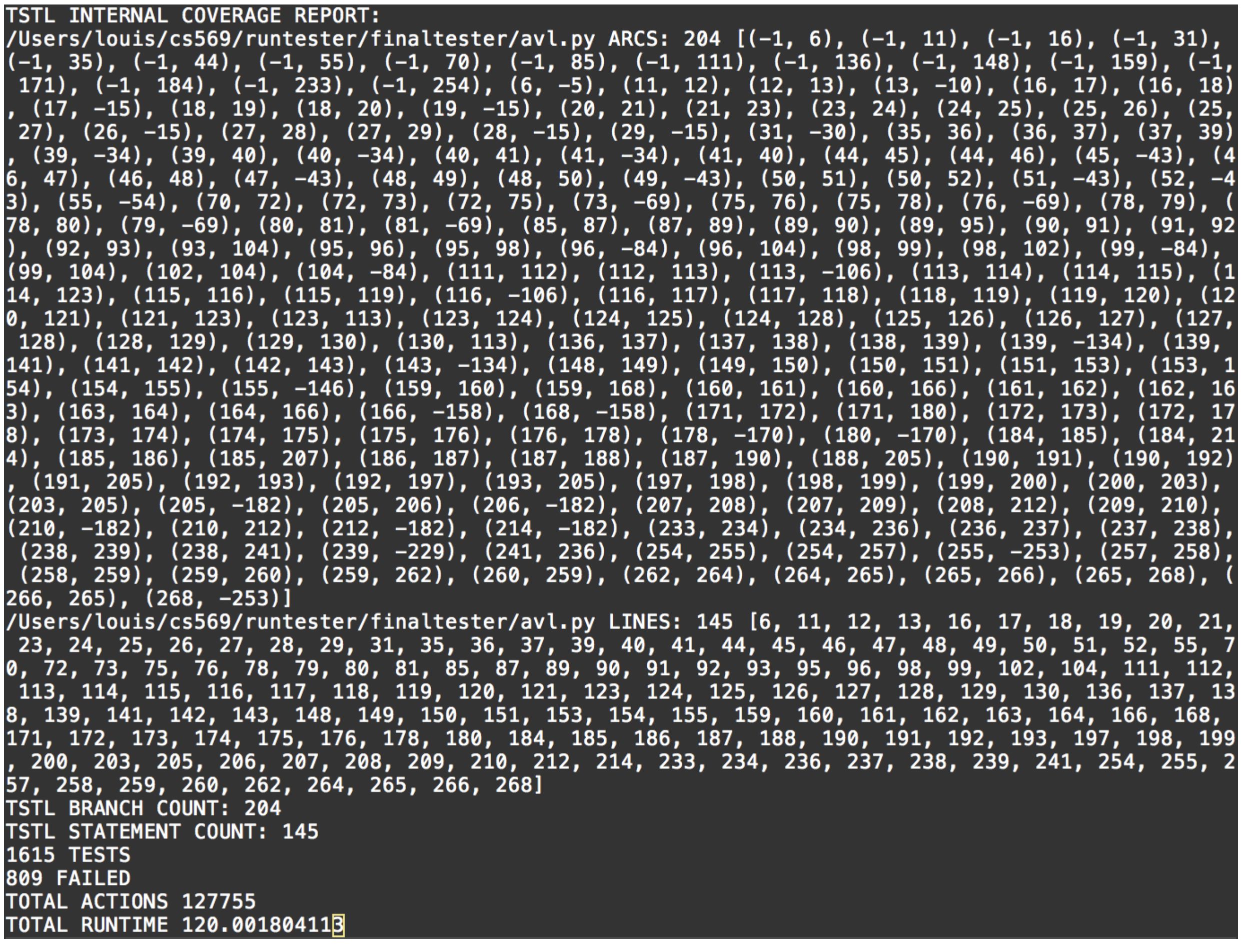
```
def run(act):          // run function is for printing out the branch information
    if running:      // if the argument running was set
        if len(sut.newBranches()) > 0:    // if the length of sut.newBranches() is greater than 0
            print "ACTION:", act[0]    // print action
            for b in sut.newBranches():    // print new branch
                print time.time() - start, len(sut.allBranches()), "New branch", b
def action():
    global actCount, bugs
    act = sut.randomEnabled(rgen)    // generate an action
    actCount += 1                              // actCount+1
    newSeq.append(act)                       // append the new action to newSeq
    for s in xrange(0,depth/2):                // for loop from 0 to depth/2
        if time.time() > start + timeout:    // if timeout then break
            break
        // if newSeq not in (nonErrorSeqs ∪ errorSeqs) then break
        if (newSeq not in nonErrorSeqs) and (newSeq not in errorSeqs):
            break
        else:    // otherwise
            newSeq.pop()    // remove the new action from newSeq,
            act = sut.randomEnabled(rgen)    // generate an new action
            newSeq.append(act)    // append the new action to newSeq

    ok = sut.safely(act)                    // check whether the action is safe
    run(act)                                // call run function to print out the branch information
    if ok:                                  // if the action is safe
        nonErrorSeqs.append(newSeq)      // append newSeq to nonErrorSeqs
    if not ok:                                              // if the action is not safe
        if time.time() > start + timeout:             // check timeout
            return not ok
        if faults:                                      // if the faults argument was set
            bugs += 1                                     // bugs+1
            print "FOUND A FAILURE"
            print sut.failure()                              // print the failure
            fname="failure" + str(bugs) + ".test"
            sut.saveTest(sut.test(),fname)             //save test to file
            errorSeqs.append(newSeq)             // append newSeq to errorSeqs
            sut.restart()                              // reset test
    return ok

while time.time() < start + timeout:                   // while it is not timeout
    sut.restart()                                      // reset test
    ntests += 1                                        // ntests +1
    for s in xrange(0,depth):                        // for loop from 0 to depth
        if time.time() > start + timeout:            // check timeout
            break
        if not action():                              // if the action() found a fault, terminate the test
            break
if coverage:                                          // if the coverage argument was set, show report
    sut.internalReport()
```

4. Evaluation

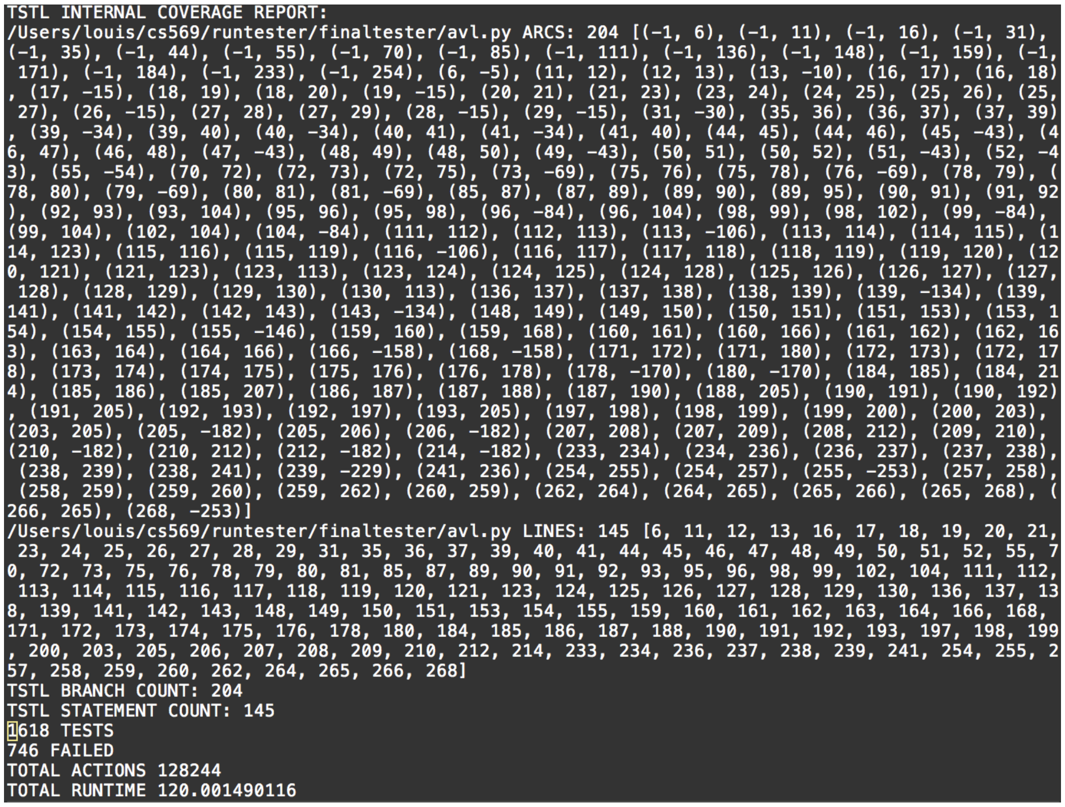
(1) Using runtesters.py to run my finaltester.py for 120 seconds, the statistics as shown in the below graph is from the file zhouxuh.z3.120s.39.tout.



```
TSTL INTERNAL COVERAGE REPORT:
/Users/louis/cs569/runtester/finaltester/avl.py ARCS: 204 [(-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, -34), (41, 40), (44, 45), (44, 46), (45, -43), (4
6, 47), (46, 48), (47, -43), (48, 49), (48, 50), (49, -43), (50, 51), (50, 52), (51, -43), (52, -4
3), (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, -84), (96, 104), (98, 99), (98, 102), (99, -84),
(99, 104), (102, 104), (104, -84), (111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (1
14, 123), (115, 116), (115, 119), (116, -106), (116, 117), (117, 118), (118, 119), (119, 120), (12
0, 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, -134), (139,
141), (141, 142), (142, 143), (143, -134), (148, 149), (149, 150), (150, 151), (151, 153), (153, 1
54), (154, 155), (155, -146), (159, 160), (159, 168), (160, 161), (160, 166), (161, 162), (162, 16
3), (163, 164), (164, 166), (166, -158), (168, -158), (171, 172), (171, 180), (172, 173), (172, 17
8), (173, 174), (174, 175), (175, 176), (176, 178), (178, -170), (180, -170), (184, 185), (184, 21
4), (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, -182), (205, 206), (206, -182), (207, 208), (207, 209), (208, 212), (209, 210),
(210, -182), (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)]
/Users/louis/cs569/runtester/finaltester/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, 7
0, 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, 13
8, 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, 2
57, 258, 259, 260, 262, 264, 265, 266, 268]
TSTL BRANCH COUNT: 204
TSTL STATEMENT COUNT: 145
1615 TESTS
809 FAILED
TOTAL ACTIONS 127755
TOTAL RUNTIME 120.001804113
```

According to the above graph, TSTL branch count is 204; TSTL statement count is 145; total tests is 1615; we found 809 bugs; total actions is 127755; running time is about 120 secs. Bug rate is 50% (809/1615).

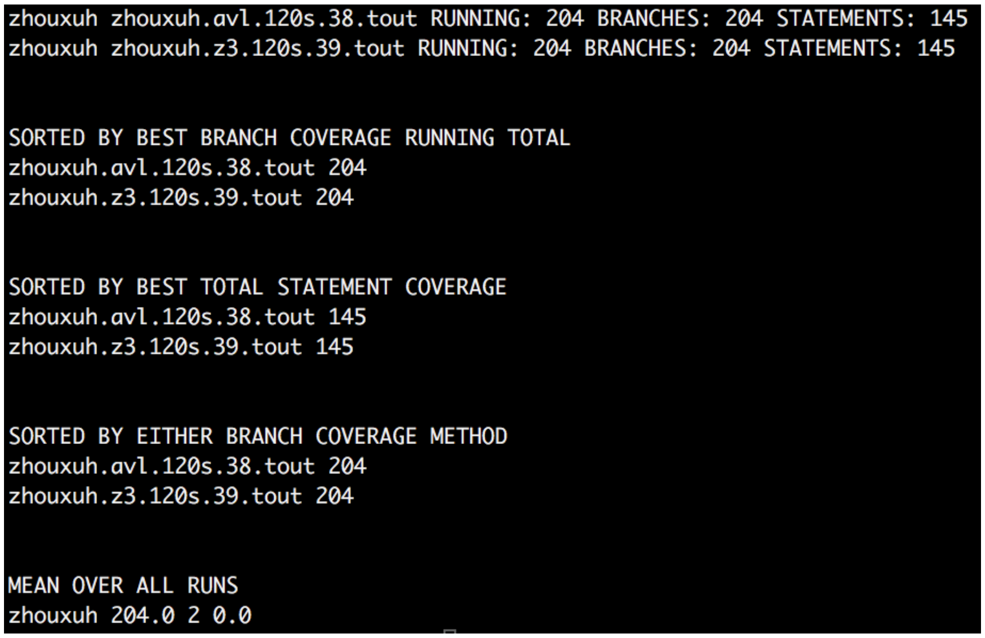
(2) Using runavl.py to run my finaltester.py for 120 seconds, the statistics as shown in the below graph is from the file zhouxuh.avl.120s.38.tout.



```
TSTL INTERNAL COVERAGE REPORT:
/Users/louis/cs569/runtester/finaltester/avl.py ARCS: 204 [(-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, -34), (41, 40), (44, 45), (44, 46), (45, -43), (4
6, 47), (46, 48), (47, -43), (48, 49), (48, 50), (49, -43), (50, 51), (50, 52), (51, -43), (52, -4
3), (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, -84), (96, 104), (98, 99), (98, 102), (99, -84),
(99, 104), (102, 104), (104, -84), (111, 112), (112, 113), (113, -106), (113, 114), (114, 115), (1
14, 123), (115, 116), (115, 119), (116, -106), (116, 117), (117, 118), (118, 119), (119, 120), (12
0, 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, -134), (139,
141), (141, 142), (142, 143), (143, -134), (148, 149), (149, 150), (150, 151), (151, 153), (153, 1
54), (154, 155), (155, -146), (159, 160), (159, 168), (160, 161), (160, 166), (161, 162), (162, 16
3), (163, 164), (164, 166), (166, -158), (168, -158), (171, 172), (171, 180), (172, 173), (172, 17
8), (173, 174), (174, 175), (175, 176), (176, 178), (178, -170), (180, -170), (184, 185), (184, 21
4), (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, -182), (205, 206), (206, -182), (207, 208), (207, 209), (208, 212), (209, 210),
(210, -182), (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)]
/Users/louis/cs569/runtester/finaltester/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, 7
0, 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, 13
8, 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, 2
57, 258, 259, 260, 262, 264, 265, 266, 268]
TSTL BRANCH COUNT: 204
TSTL STATEMENT COUNT: 145
1618 TESTS
746 FAILED
TOTAL ACTIONS 128244
TOTAL RUNTIME 120.001490116
```

According to the above graph, TSTL branch count is 204; TSTL statement count is 145; total tests is 1618; we found 746 bugs; total actions is 128244; running time is about 120 secs. Bug rate is 46% (746/1618).

(3) Using tabulateCov.py to get the statistics as shown in the below graph



```
zhouxuh zhouxuh.avl.120s.38.tout RUNNING: 204 BRANCHES: 204 STATEMENTS: 145
zhouxuh zhouxuh.z3.120s.39.tout RUNNING: 204 BRANCHES: 204 STATEMENTS: 145


SORTED BY BEST BRANCH COVERAGE RUNNING TOTAL
zhouxuh.avl.120s.38.tout 204
zhouxuh.z3.120s.39.tout 204


SORTED BY BEST TOTAL STATEMENT COVERAGE
zhouxuh.avl.120s.38.tout 145
zhouxuh.z3.120s.39.tout 145


SORTED BY EITHER BRANCH COVERAGE METHOD
zhouxuh.avl.120s.38.tout 204
zhouxuh.z3.120s.39.tout 204


MEAN OVER ALL RUNS
zhouxuh 204.0 2 0.0
```

According to the above graph, the information of the RUNNING, BRANCHES and STATEMENTS is successfully shown.

5. Conclusion

This case generation algorithm supervises the process of the action generation and avoids the generator to generate redundant actions. Thereby, the tester is running valued test cases. Therefore, from the Evaluation section, we can observe that using this new test case generation algorithm, the tester can perform more tests and find out more faults. Also, the high bug rate (about 50%) is appealing.

Reference

[1] Groce, Alex, et al. "TSTL: a language and tool for testing." Proceedings of the 2015 International Symposium on Software Testing and Analysis. ACM, 2015.

[2] Hamlet, Dick. "When only random testing will do." Proceedings of the 1st international workshop on Random testing. ACM, 2006.

[3] Pacheco, Carlos, et al. "Feedback-directed random test generation." Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007.

Appendix

```
import sut
import random
import sys
import time
def run(act):
    if running:
        if len(sut.newBranches()) > 0:
            print "ACTION:", act[0]
            for b in sut.newBranches():
                print time.time() - start, len(sut.allBranches()), "New branch", b
def action():
    global actCount, bugs
    act = sut.randomEnabled(rgen)
    actCount += 1
    newSeq.append(act)
    for s in xrange(0,depth/2):
        if time.time() > start + timeout:
            break
        if (newSeq not in nonErrorSeqs) and (newSeq not in errorSeqs):
            break
        else:
            newSeq.pop()
            act = sut.randomEnabled(rgen)
            newSeq.append(act)
    ok = sut.safely(act)
    run(act)
    if ok:
        nonErrorSeqs.append(newSeq)
    if not ok:
        if time.time() > start + timeout:
            return not ok
        if faults:
            bugs += 1
            print "FOUND A FAILURE"
            print sut.failure()
            fname="failure" + str(bugs) + ".test"
            sut.saveTest(sut.test(),fname)
            errorSeqs.append(newSeq)
            sut.restart()
    return ok
timeout    = int(sys.argv[1])
```

```
seed        = int(sys.argv[2])
depth       = int(sys.argv[3])
width       = int(sys.argv[4])
faults      = int(sys.argv[5])
coverage = int(sys.argv[6])
running    = int(sys.argv[7])
rgen = random.Random()
rgen.seed(seed)
sut = sut.sut()
actCount = 0
bugs = 0
ntests = 0
errorSeqs= []
nonErrorSeqs = []
newSeq = []
start = time.time()
while time.time() < start + timeout:
        sut.restart()
        ntests += 1
        for s in xrange(0,depth):
                if time.time() > start + timeout:
                        break
                if not action():
                        break
if coverage:
        sut.internalReport()
print ntests,"TESTS"
print bugs,"FAILED"
print "TOTAL ACTIONS",actCount
print "TOTAL RUNTIME",time.time()-start
```