CS569: Static Analysis and Model Checking for Dynamic Analysis
Part 3: COMPETITIVE MILESTONE 2

Hafed Alghamdi
Email: Alghamha@oregonstate.edu

Introduction

Generating test cases for a Software Under Test (SUT) is not an easy task as there are many approaches and techniques that can be adapted depending on the SUT. Moreover, it is more challenging to generalize the test generator on deferent SUTs. The most easiest and effective approach is Random Testing as it is easy to implement and it provides excellent results most of the time. Therefore, I decided to work on a simple Random Tester with a little modification as it will help in the overall competitions and its implementation is going to be relatively easy. Moreover, another idea is currently under exploration as discussed with Professor Alex that works on grouping actions based on timeout divisions. This interesting idea will be implemented separately as an extra effort to the class. A simple algorithm have been developed and tested but it needs more time to be finalized, hopefully before the end of this class period.

Random/Sequential Algorithm Based on Random Probability and Good Test Cases

This algorithm combines both sequential and random tester techniques to search for faults as quickly as possible. Since some actions in the SUTs could trigger a fault by just executing them with any random values, then it is a good idea to execute all actions sequentially at least once before applying any other techniques. Thus, sequential search has been adapted in this algorithm as a first step to execute TSTL generated SUTs only once. The second step is generating random test cases based on Random probability selection that satisfies 0.5 probability condition. It is a normal implementation of any random tester with the addition of saving good test cases in memory to be replayed back based on random selection of a probability that is less than 0.5. Testing this on the modified avl SUT file reveals that triggering the combination luck problems is promising.

Modifications Applied on this Algorithm (Part 3)

1- Saving test case after triggering a fault has been modified to save the test case as is without any changes as discussed with Professor Alex. The function responsible for that has been tested and the output file was successfully loaded using TSTL $sut.loadTest(FileName)$ function.
2- The Deadlock problem (No Actions Enabled) was not considered in the last version. In this version this issue has been sorted out by checking the selected action before executing it
3- Checking properties was not part of this submission, however, it has been implemented. As discussed with Professor Alex, it will be committed for the time being until this part covered in class because it will discover more branches and statements while other testers don't.

Figure 1 explains the algorithm in details.

```
# Function To Save The Faults
def         saveFaults(bug, testCase):
            FileName = 'failure'+str(bug)+'.test'
            file = open(FileName, 'w+')
            for s in testCase:
                        print >> file, sut.serializable(s)
            file.close()
            #print sut.loadTest(FileName)


# Sequntial algorithm that will traverse over all actions and execute them one by one
for act in sut.enabled():
            seq = sut.safely(act)
            if (not seq) and (FaultsEnabled == 1):
                        #Sequential = "Discovered By Sequential Algorithm"
                        elapsedFailure = time.time() - startTime
                        bugs += 1
                        print "FOUND A FAILURE"
                        sut.prettyPrintTest(sut.test())
                        test = sut.test()
                        Fault = sut.failure()
                        saveFaults( bugs, test)
                        sut.restart()
                                    # Print the new discovered branches
            if (len(sut.newBranches()) > 0) and (RunningEnabled == 1):
                        #print "sequential found this branch"
                        print "ACTION:",act[0]
                        elapsed1 = time.time() - startTime
                        for b in sut.newBranches():
                                    print elapsed1,len(sut.allBranches()),"New branch",b
sut.restart()
rgen = random.Random(seeds)
#rgen = random.Random()
#rgen.seed(seeds)
action = None
# RandomTester based on randomly selcted propability
while (time.time() - startTime <= timeout):
            # This will work only Memory input is set. It is good for finding combanition luck faults
            if (len(goodTests) > 0) and (rgen.random() < 0.5):
                        sut.backtrack(rgen.choice(goodTests)[1])
                        if (time.time() - startTime >= timeout):
                                    break
            else:
                        sut.restart()

            # Based on the depth randonly execute an action
            for s in xrange(0,depth):
                        if (time.time() - startTime >= timeout):
                                    break
                        action = sut.randomEnabled(rgen)
                        if (action == None):
                                    print "TERMINATING TEST DUE TO NO ENABLED ACTIONS"
                                    break
                        r = sut.safely(action)
                        # Start saving discovered fault on Disk
                        if (not r) and (FaultsEnabled == 1):
                                    #RandomAlgorithm = "Discovered By Random Algorithm"
                                    elapsedFailure = time.time() - startTime
                                    bugs += 1
                                    print "FOUND A FAILURE"
                                    print sut.failure()
                                    sut.prettyPrintTest(sut.test())
                                    test = sut.test()
                                    #Saving discovered fault on Disk
                                    saveFaults(bugs, test)
                                    # Rest the system state
                                    sut.restart()
                        if (time.time() - startTime >= timeout):
                                    break
                        # This part is for checking the property
                        '''
                        checkResult = sut.check()
                        if (not checkResult):
                                    bugs += 1
                                    print "FOUND A FAILURE"
                                    print sut.failure()
                                    sut.prettyPrintTest(sut.test())
                                    test = sut.test()
                                    #Saving discovered fault on Disk
                                    saveFaults(bugs, test)
                                    # Rest the system state
                                    sut.restart()
                        if (time.time() - startTime >= timeout):
                                    break
                        '''
                        # Print the new discovered branches
                        if (len(sut.newBranches()) > 0) and (RunningEnabled == 1):
                                    #print "Random Found this branches"
                                    print "ACTION:",action[0]
                                    elapsed1 = time.time() - startTime
                                    for b in sut.newBranches():
                                                print elapsed1,len(sut.allBranches()),"New branch",b
                        if (time.time() - startTime >= timeout):
                                    break
                        # When getting new branches, save the test case into goodTest list to be re-executed based on random propability
                        if ((length != 0) and ((len(sut.newBranches()) > 0) or (len(sut.newStatements()) > 0))):
                                    goodTests.append((sut.currBranches(), sut.state()))
                                    goodTests = sorted(goodTests, reverse=True)[:length]
                        # Cleanup goodTest list based on the length of the goodTests
                        if (length != 0) and (len(sut.newBranches()) == 0) and (len(goodTests) >= length):
                                    RandomMemebersSelection = random.sample(goodTests,int(float((len(goodTests))*.20)))
                                    for x in RandomMemebersSelection:
                                                goodTests.remove(x)

# Printing Report
elapsed = time.time() - startTime
print "\n                ############ The Final Report ############# \n"
print elapsed, "Total Running Time"
print bugs, " Failures Found"
if CoverageEnabled == 1:
            print len(sut.allBranches()),"BRANCHES COVERED"
            print len(sut.allStatements()),"STATEMENTS COVERED"
            sut.internalReport()
```

Figure 1

How to Use This Algorithm:

This algorithm can be used as specified in project part 2 requirements. When searching for combination lock faults, it is recommended to set the MEMORY/WIDTH to some value. If you want to use the algorithm for any other faults, you can use the default values as commented in the code. Here are some examples:

1- Combination Lock Faults like in avl SUT:
   ```
   python tester1.py 60 0 100 100 1 1 1
   ```
2- For any other test cases, the following can be used as an example:
   ```
   python tester1.py 60 0 100 0 1 1 1   # To use the normal Random Tester
   ```
   OR
   ```
   python tester1.py 60 10 100 0 1 1 1  # To change the number of seed
   ```

You can use as *Figure 2* as a reference to what input you should use to correctly run this algorithm.

```
# Terminate the program with time
# You can use 60 as a default Value
timeout = int(sys.argv[1])

# Determines the random seed for testing. This should be assigned 0 when using the
MEMORY/WIDTH
# You can use 12 as a default Value
seeds = int(sys.argv[2])

# TEST_LENGTH or Depth
# You can use 100 as a default Value
depth = int(sys.argv[3])

# MEMORY or Width, the number of "good" tests to store
# You can use a 100 as a default Value when testing combination lock faults
length = int(sys.argv[4])

# Enable/Disable Faults
# You can use 1 as a default Value
FaultsEnabled = int(sys.argv[5])

# Enable/Disable Coverage
# You can use 1 as a default Value
CoverageEnabled = int(sys.argv[6])

# Enable/Disable Running
# You can use 1 as a default Value
RunningEnabled = int(sys.argv[7])
```
Figure 2