Problem Set 7

(10 points) Cohort Exercise 1:

Test the new istd.sutd.edu.sg webpage by randomly clicking links. The process may go on forever.

(10 points) Cohort Exercise 2:

Write a test to do the following: Send N invalid usernames to Google login form (be innovative about the invalid user names) before the captcha appears and then login with your user name and password. For the purpose of the submission, of course, you do not need to show your user name and password.

(10 points) Cohort Exercise 3:

Write a test to check whether all web pages *directly* reachable from a given webpage have titles. The test will fail if any such directly reachable webpage has an empty title. Hint: Use getTitle() from the web driver.

(20 points) Cohort Exercise 4:

Use any programming language to implement a fuzzer that will randomly generate inputs to the calculator conforming to the grammar. **For simplicity, you can hardcode the expression grammar.**

Hint: Start with the initial rule S := Expr and at each point, apply a rule at random. For example, randomly choose any of the rule Expr := Term, Expr := Expr + Term or Expr – Term in the next step. Continue until a valid expression for the calculator is obtained. **Make sure you do not expand the rules forever to avoid infinite loop .**

(5 points) Cohort Exercise 5:

Assume that we need to instrument the code in slide 44 [week10slides(PartII)] to obtain statement coverage measure. Instrument the code in a way such that we can obtain the statement coverage of any test case while keeping the overhead minimum. It is enough to show the control flow graph and the point of insertions of instrumentation code (you do not need to write code for the instrumentation).

(10 points) Cohort Exercise 6:

1. Study the implementation of the Genetic Algorithm provided.

2. Modify the provided genetic algorithm so that it generates any palindrome string with 64 characters.

- How do you define the fitness function?
- How do you define the selection/crossover/mutation operator?

(15 points) Cohort Exercise 7:

Write two example classes both of which have the same number of conditional branches as the Example.java, however, one of them is easier to test and the other is harder to test as compared to Example.java.

1. Compare the performance of Evosuite for both the examples (the one in the previous example and the ones you come up with in this exercise). Concretely, compare the conditional branch coverage and the average coverage over all coverage criterions. Also compare the time taken for Evosuite for all cases.

2. Argue why your examples take less/more time (they should!!!!) to test as compared to Example.java.

(20 points) Homework:

Implement a generalized fuzzer that will take a file, read each line of the file, randomly choose a mutation operator (swap, bit flip or trim) and produce a different file with the modified lines. This means for each input file, the fuzzer will produce one output file, where each line is modified with a random mutation operator. Choose any programming language. Make your program modular so that more mutation operators can be added easily.

(10 points) Cohort Exercise 8:

Remove the "repeated code smell" from BrokenLinkFinderSmell.java

(10 points) Cohort Exercise 9:

Write a CreditTransaction() method in the AccountSmell class which also records the last credit time. Refactor the code to remove the smells.

(15 points) Cohort Exercise 10:

In ShootTheAccount.java, assume we wish to add a new feature that only personal accounts will become dysfunctional when balance drops below 500. All other type of accounts (say "business" account) are exempted from this restriction. Change the file to add this feature.

Remove the shot gun surgery code smell from ShootTheAccountPlus.java.

(10 points) Cohort Exercise 11:

Consider XSSFixed.java. Discuss whether the cross-site scripting (XSS) style of attacks can be prevented by the code. If your answer is yes, then justify your answer. If your answer is no, then fix the code.

(30 points) Cohort Exercise 12:

The program exercise4.java overrides the methods after() and compareTo() of java.util.Calendar. The programmer wishes to extend this functionality so that the after() method returns true even when the two objects represent the same date. The programmer also overrides the method compareTo() to provide a “comparisons by day” option. Compile and run the program. Discuss the observed problems.

Provide a solution to fix the observed problem.

(15 points) Homework:

Write a loop-free C program that has one symbolic character input and eight (8) different paths. Test your program with KLEE and produce all the SMT2 files. For each SMT2 file, compute the model (i.e. the test or the concrete value of the symbolic input) using Z3 web-service. Submit your program, all SMT2 files and the computed models by Z3.