CS639A: Program Analysis, Verification And Testing
Users Online : 1

Assignment-6B: Spectrum Based Fault Localization

Submitted on 16/11/2021 19:41

Instructions

- You are given an extra 10 minutes after due time to submit your assignment.
- However, please note that any submissions made after the due time are marked as late submissions.

Assignment-6B: Spectrum Based Fault Localization

Question:

Problem Statement:

In this assignment, we will explore Spectrum-based Fault Localization (**SBFL**) to identify buggy line/lines in a program i
a regression testing scenario. Assume that, a team of developers have written a program **P**, which is known to be corre
A new developer has recently joined the team and made some changes to **P**. Let us refer to this changed version as **P2**.
The team of developers want your help to figure out whether **P2** is buggy. If **P2** is found to be buggy, they would also be
grateful if you can tell them which lines are most suspicious, i.e., most likely to contain the bug.

One possible way, by which you can help them, is by employing the **SBFL** framework in Kachua. The steps that you nee
to follow are:

Generate a test-suite **T** on **P** using the evolutionary search-based test-suite generation framework in Kachua. You will need to provide a fitness function for the evoulutionary search to the test-suite generator such that it may come with a optimal test-suite for fault localization. Test-suite **T** will be generated in the form of an activity matrix **A**.
Next, execute **T** on **P2** and compare the output of **P2** against that of **P** for each test-case t in the Test-suite T. If the outputs of **P2** matches with that of **P**, then the corresponding test-case is a passing test-case. Otherwise, it is a failing test-case. Executing all the test-cases will generate the error vector **E**. *The error vector **E** will be generated by the SBFL framework itself.*
Next, armed with the spectrum consisting of A and E, you will try to find the bug in the program. This step would require for you to implement a fault-localization metric. The SBFL framework will utilize your metric to compute the suspiciousness score of each program component and rank them in order of their suspiciousness. Observe the rank of the buggy component in the ranked list. The closer it is to the top, the better is your fitness function and fault-localization metric.

Assumptions :

IR of a turtle program is consider as components.
if **len(IR)** = 5 then components are 0, 1, 2, 3, 4 i.e c_0, c_1, c_2, c_3 and c_4.
A test case fails if turtle location after completing execution of correct and buggy turtle program are not same, otherw
passes.

A SBFl framework has been implemented for you in sbfl.py file. please go through it to get the better understanding of the framework.

Implement the marked functions and class interfaces in Submission/sbflSubmission.py file for this assignment.

def fitnessScore(IndividualObject): # input_data type Individual()
This function will returns the fitness of the test-suite.

CS639A: Program Analysis, Verification And Testing
Users Online : 1

*please don't modify function '**def computeRanks(spectrum,outfilename)**'*

Version:

Minimum Kachua version to use is v5.0 .

Running the SBFL : (From KachuaCore folder)

```
$ ./kachua.py --SBFL ./example/sbfl1.tl --buggy ./example/sbfl1_buggy.tl -vars '[":x", ":y", ":z"]' --
timeout 10 --ntests 20 --popsize 100 --cxpb 1.0 --mutpb 1.0 --ngen 100 --verbose True
```

Arguments :

--*buggy* : Buggy turtle program.
-*vars* : Input variables to the program. if program does not have any input variable, pass **-vars '[]'**.
--*timeout* :Time budget for to run each test cases.
--*ntests* : Intitial test-suite size.
genetic algorithm parameters:
--*popsize* : Population size.
--*cxpb* : Cross-over probability.
--*mutpb* : Mutation probability.
--*ngen* : Number of times GA to iterate.
--*verbose* : To show GA output to console.

Deliverables

The source code of your implementation. Only submission.py and any other files created by you must be submitted.
A brief report (less than 5-pages) describing your implementation, assumptions and limitations. (Understand the difference between the tool's limitation and a bug: any error or missing feature that is caught during evaluation is a bu
unless it is listed under the "limitations" section of your tool.)
A set of test cases (at least 5) with the expected output. (tests folder, KachuaCore).

The quality of all the above would affect your marks. The quality of all the above would affect your marks.

Submission Format

Your submission **MUST** be in the following format:

The submission should be a **zip** file.
The zip file should be named as **assignment_**"**number**"_"**Roll-of-student**".
Zip the content of the source as is and submit. **Don't refactor the base code or move the files around**. (KachuaCore Folder, Submission Folder, README)

Please note that your submission will **NOT** be graded if you do not follow the format. Furthermore, we will use the **Readme** file provided by you to build and run your code. Therefore, please make sure that the Readme is clear. We cannot grade your submission if we cannot run it on our system.

Some important comments

CS639A: Program Analysis, Verification And Testing
Users Online : 1

tool that simply spits out a bunch of numbers. So, display the results from your tool suitably.
Discussion is healthy, copying is not. You are encouraged to discuss the assignments, but you must implement the assignments individually. If any two students are found with "similar" pieces of code, both of them will be failed (with no concern as to who was the source).

Uploaded Files:

assignment_6_21111037.zip

Grades:

Marks: 90
(-5) : minor error in the implementation of fitness function.
Feedback: (-5) : minor error in implementation of suspiciousness function.