PROGRAM ANALYSIS, VERIFICATION AND TESTING

(CS639)

ASSIGNMENT - 3

Submitted by:

PRATIBHA GUPTA
231110038

In the given task we are supposed to implement Spectrum Based Fault Localization. For which we need to complete three functions in the 'sbflSubmission.py' file which is at the location 'Chiron-Framework\Submission'. The details of implementation of the functions are given as:

IMPLEMENTATION -

1) **def fitnessScore(IndividualObject):**

In this function we are supposed to implement fitness computation that quantifies how good a test suite is. I have implemented Uylsis - Multiverse Analysis for the same. Ulysis computes the average worst-case wasted effort over all imaginary universes (multiverse). Ulysis score is the fitness_score that the function returns. So, the formula for ulysis score is:

$$\mathcal{L}_i = \begin{cases} c_j \mid c_j \in C, j \neq i, & \text{if } c_i = \vec{0} \\ c_j \mid c_j \in C, c_j = c_i, j \neq i, & \text{otherwise.} \end{cases}$$

$$\mathcal{W}_i = \frac{|\mathcal{L}_i|}{m-1}$$

$$\mathcal{W}_{Ulysis} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{W}_i$$

So, for Li I have directly counted the number of components with the same activity pattern instead of storing the complete set. Then add it to wi after dividing by m-1 (where, m in number of components). Finally, fitness_score = W_ulysis.

2) **def suspiciousness(self, comp_index):**

In this function we are supposed to calculate the suspiciousness for the comp_index in the activity matrix. For doing the same I have used the Ochiai metric:

C_f	Number of failing tests that execute C
C_p	Number of passing tests that execute C
N_f	Number of failing tests that do not execute C
N_p	Number of passing tests that do not execute C

$$Ochiai(C) = \frac{C_f}{\sqrt{(C_f+N_f).(C_f+C_p)}}$$

I have compared the activity_met for given component index (stored in a list "component") and errorVec and increment the value of cf, cp and nf respectively as:

Increment cf : when errorVec[i]==1 and component[i]==1.

Increment cp : when errorVec[i]==0 and component[i]==1.

Increment cp : when errorVec[i]==1 and component[i]==0.

Then used the metric formula to compute the suspiciousness.

3) **def getRankList(self):**

This is the calling function of function suspiciousness. To generate the list with suspiciousness of each component I have called the function self.suspiciousness(i) for all the components. Then, the list is sorted in descending order of suspiciousness values. Now, for ranking I have assigned the worst case rank to components with the same suspiciousness score as components with the same score can't have different rank and returned the rank list.

ASSUMPTION -

1- In one run, only one component of the program is executed.

LIMITATIONS -

1- Fails to locate error if more than one component is run for an execution.
2- For large codebases with many test cases, it can become computationally expensive.
3- Since SBFL relies on runtime execution data. This means that the results may vary between test runs, depending on the specific test cases executed and the input data. This variability can make it difficult to consistently locate faults.