ASSIGNMENT 6: SPECTRUM BASED FAULT LOCALIZATION

KACHUA VERSION: 5.1

EXECUTION FLOW OF THE TOOL:

- User inputs- the turtle file path of correct program and suspected buggy program, timeout in seconds to run each test case , list of variables which are initialized by user input, no of initial test cases to generate, and genetic algorithm parameters such as population size, no of times genetic algorithm has to be run, crossover and mutation probability , verbose output to be displayed or not as command line argument to the ***Kachua.py***, both the files are then opened and parsed to make the ***Intermediate Representation (IR) ir1 and ir2.***
- Except turtle file paths, all command line arguments and generated ir1 and ir2 are passed to **testsuiteGenerator()** as parameters to generate initial test suite, optimal test suite on which outputs of correct and suspected buggy program outputs are compared to generate a spectrum which is a combination of activity matrix and error vector .
- **testsuiteGenerator()** uses the fitness function (**fitnessScore()**)defined in **sbflSubmission.py** to determine the quality / goodness of the test suite based on the spectrum generated. Based on the fitness score of each test suite the best test suite is returned as optimized test suite and spectrum corresponding to it is returned.
- **fitnessScore()** uses either **ddumetric()** or **ulysis()** to compute the fitness score of a test suite. Here traditional DDU metric or Ulysis have not been used but a slightly modified version is used.
- **Kachua.py** using the returned spectrum as argument for **computeRanks()** function in **sbflSubmission.py** to determine the rank of each component of buggy program based on suspiciousness score in descending order that is component with highest suspiciousness score gets rank 1 and so on.
- **computeRanks()** first bakes a **SpectrumBug** type object and calls **getRankList()**
- **getRankList()** passes the spectrum to **suspiciousness()** to calculate suspiciousness score of each component.
- **suspiciousness()** defined here uses '***Ochiai Score***' to calculate suspiciousness score of each component of buggy program. And returns it as a list to **getRankList().**
- **getRankList()** sorts the suspiciousness score list in descending order and assigns appropriate rank to each component and returns the list to **computeRanks().**
- **computeRanks()** outputs the returned rank list to a csv file and returns control to Kachua.py
- Kachua.py then writes all other data variables like original test suite and its activity matrix, optimized test suite and its activity matrix and the spectrum as csv files.

HARSH AGARWAL
21111030
MTECH CSE 2021-23
PROGRAM ANALYSIS, VERIFICATION AND TESTING

STEPS TO RUN THE TOOL:

1. Open any terminal (***Command Prompt, Windows PowerShell, Git Bash***, etc.) and move into the directory where **k*achua.py*** is present
2. Type " ***python <space> kachua.py <space> --SBFL <space> file path/turtle file name.tl <space> --buggy <space> file path/ buggy turtle file.tl <space> -vars <space> list of user initialized variables <space> --timeout <space> time in seconds to run each test case <space> --ntests <space> no of initial test case to generate <space> --popsize <space> population size for genetic algorithm <space> --cxpb <space> crossover probability <space> --mutp <space> mutation probability <space> --ngen <space> no of iterations to run of genetic algorithm <space> --verbose <space> Either True of False to whether display verbose output in terminal ".***

   For eg, to run one of the test cases provided type: "**python kachua.py --SBFL ./tests/mytest5.tl --buggy ./tests/mytest5_bug.tl -vars [':a']" --timeout 10 --ntests 20 --popsize 100 --cxpb 1.0 --mutpb 1.0 --ngen 10 --verbose True**"
3. Press Enter
4. **Spectrum, Suspiciousness Score and Rank List of components is displayed in Terminal.**

IMPLEMENTATION LOGIC:

- **fitnessScore() : Takes IndividualObject (test suite) as input parameter and returns its fitness score.**
  - Derives the activity matrix from the test suite spectrum by removing error vector column
  - Calls either ulysis() or ddumetric() with parameter as activity matrix calculated in previous step to calculate fitness score of test suite
  - Returns the fitness score

- **ddumetric():** Function to find out fitness score of a test suite using metrics such as its density diversity and uniqueness as generally known. A slight modification to the formula is made such that for each component which was not activated for any test case, we decrease the score by -1.
  - Calculate no of rows and columns in activity matrix and initialize fscore to 0
  - Density is calculated as total no of 1's in activity matrix divided by dimension of activity matrix. Optimal value is 0.5
  - Then calculates no of unique rows in the activity matrix and using this calculates the diversity of the activity matrix. If no of rows is 1 then diversity is assumed to be 1. Optimal diversity value is 1
  - Then calculates no of unique columns in the activity matrix and calculates uniqueness of test suite as no of unique columns divided by total no of columns. Optimal value is 1

- fscore is calculated as product of diversity , uniqueness and modified diversity. Optimal value is 1.
- Then for every zero column in activity matrix we subtract 1 from f score so as to penalize test suite and therefore promote test suite where each component is activated at least once.
- Return -1 * fscore

- **ulysis():** Function to find out fitness score of a test suite. A slight modification to the formula discussed in class has been made such that for each component which was not activated for any testcase, we increase the wasted effort by no of columns in test suite.
  - Find out no of columns in activity matrix and initialize sum1 to 0
  - Then for every component we assume it to be buggy and try to find out wasted effort till we reach that component. To do so, first we check whether it is a zero column, ie component is not activated in any test case:
    - If yes, then we increase sum1 value by no of columns in activity matrix to penalize the test suite and encourage selection of test suite in which every component is activated at least once.
    - If no, then we check no of components having same activity pattern as current assumed buggy component and we increase sim1 by value ratio of no of components with same pattern to total no of components – 1
    - To calculate final fitness score we divide by no of columns or components to find average wasted effort.
    - Returns the fitness score.

- **suspiciousness() : Takes component no as parameter and returns suspiciousness score using Ochiai formula.**
  - Initialize sus_score, Cf (No of failing cases executing component C), Cp (No of passing cases executing component C), Nf (No of failing cases not executing component C), Np (No of passing cases not executing component C) to 0
  - get activity pattern for component C using getActivity().
  - For each test case, we determine in which above mntioned category does component C fall under and we increment appropriate variable by 1
  - Check if Cf is 0 then return suspiciousness score as 0
  - Else calculate suspiciousness score as per Ochiai formula
  - Return suspiciousness score.

- **getRankList() : Returns rank list of components based on their suspiciousness score.**
  - Define a empty list called rankList
  - Print the spectrum passed on Terminal
  - For each component, calculate its suspiciousness score by calling suspiciousness() with component no and store it in a list.
  - Sort the score list in descending order and print it on termninal.
  - Using the sorted suspiciousness score list, calculate rank of each component and add it to list rankList as [component no , rank] where rankList is sorted in ascending order by component no.
  - Return rankList.

LIMITATIONS & ASSUMPTIONS:

- There are no errors raised during execution of the input turtle program.
- Parameters passed through command line are good enough to produce a test suite which has most of possible paths covered and the activity matrix generated as a result is decent enough to give good fault localization.
- Timeout in seconds for running each test case is enough for most of the test cases to complete execution successfully.
- Suspicion score and rank of components are directly dependent on test suite quality and amount of coverage ie it is preferable that all components are executed at least once in test suite.

NOTE:

Both Ulysis metric and Density Diversity Uniqueness Metric have been implemented with slight modifications but for submission purpose ddumetric() has been called. To run with Ulysis as metric change functional call in **sbflSubmission.py** in fitnessScore() from ddumetric() to ulysis(). Performance in terms of test suite selection is same for both the metrics.

HARSH AGARWAL
21111030
MTECH CSE 2021-23