ASSIGNMENT 4: COVERAGE GUIDED FUZZING

KACHUA VERSION: 2.3

EXECUTION FLOW OF THE TOOL:

- User inputs the turtle file path, initial seed inputs and timeout in seconds as command line argument to the ***Kachua.py***, file is then opened and parsed to make the ***Intermediate Representation (IR).***

- ***IR*, timeout,** and **initial seed input**. are passed as arguments to the ***fuzzmain()*** in the ***fuzzer.py by Kachua.py*** to fuzz the turtle program

- Then **fuzzmain()** repeats below steps until timeout occurs:

  - Mutate the randomly selected input from Corpus and execute given turtle program with mutated input
  - Coverage is calculated as all the **IR** statements executed in the current execution.
  - **compareCoverage()** in **Submission.py** is called with current execution coverage and total coverage as arguments. If coverage has increased, that is new **IR** statement(s) executed then it returns **True** else **False**
  - If **True,** returned by **compareCoverage(),** then append mutated input to Corpus and total coverage is updated by calling u**pdateTotalCoverage()** in **Submission.py** with arguments current coverage and total coverage.
  - Checks if timeout has occurred then stops Fuzzer loop and returns the Corpus and Total Coverage to **kachua.py** else it repeats above steps

- At the end total coverage, time exhausted and list of interesting inputs (Corpus) is displayed in terminal. Also, a line graph showing progress of coverage with each iteration is stored as a .png image file

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

STEPS TO RUN THE TOOL:

1. Check if python packages like ***graphviz, kturtle, numpy, matplotlib*** and other packages and dependencies mentioned in ***Readme.md*** are installed properly, if not you can install it by running command “***pip <space> install <space> <package name>***” in any terminal or appropriate commands from ***Readme.md***

2. Open any terminal (***Command Prompt, Windows PowerShell, Git Bash***, etc.) and move into the directory where **k*achua.py*** is present

3. Type “***. /kachua.py <space> -t <space>’timeout in seconds’<space> --fuzz <space> file path/turtle file name.tl***” **<space> -d <space> ‘seed input’** to run the fuzzer

   For eg, to run one of the testcases provided type: “**. /kachua.py -t 100 –fuzz ./tests/mytest6.tl -d ‘{“: b”: 50, “: a”: 5}’** ”

4. Press Enter

5. ***Time Exhausted, Total Coverage and Corpus (list of interesting inputs) is displayed in the terminal***

6. Also, a line graph showing progress of coverage with each iteration is stored as a .png image file in **Submission** folder with filename ‘graph.png’.

IMPLEMENTATION LOGIC:

- **compareCoverage():** Takes current coverage metric and total coverage metric as arguments.
  - Increments counter by 1, to count no of iterations.
  - Prints the Coverage Percent after executing mutated input.
  - For every **IR** id in current coverage metric, we check if that **IR** id is in total coverage metric, if there **continue** else return **True**
  - Add length of total coverage metric – 1 to len_tomet with key as iteration no.
  - Call **graphplotter()** to plot the Coverage progress against no of iterations.
  - Return **False.**

- **updateTotalCoverage():** Called only if **compareCoverage()** returned **True.** Takes current coverage metric and total coverage metric as arguments.
    - For every **IR** id in current coverage metric, we check if that **IR** id is in total coverage metric, if not there add it to total coverage metric else **continue**
    - Add length of total coverage metric – 1 to len_tomet with key as iteration no.
    - Prints the Total Coverage Percent after executing mutated input.
    - Call **graphplotter()** to plot the Coverage progress against no of iterations.
    - Return total coverage metric**.**

- **graphplotter():** Plots the Coverage progress against no of iterations.
    - With Coverage progress on y axis with label 'Length of Total Metric'
    - With Iteration no on x axis with label 'Iteration'
    - Sets graph title as 'Length of Total Metric V/S Iteration'.
    - Saves the graph in **Submission** folder with filename 'graph.png' without white space around the graph

- **mutate():** Mutates the given input and takes input_data, coverageInfo and irList as arguments.
    - Prints the **IR** just once at beginning.
    - For every variable being fuzzed:
        - First convert into binary format
        - Split the binary on 'b' so bin_val contains only the binary digits
        - Check if bin_val is either 0 , 1 ,-1 then add '0000000' to the beginning of bin_val
        - Generates a random number between 0 and 1000 and takes modulus 10 and checks if:
            - It is 1, then again generates a random number between 0 and 100 and takes modulus 4. if it is:
                - 0, then assigns variable value as 300
                - 1, then assigns variable value as -300
                - 2, then assigns variable value as 0
                - 3, then assigns variable value as 1

    - It is 2, then we multiply current variable value by -1

    - Else,
        - Generates a random integer between 1 and length of bin_val and then takes that many no of samples from a list containing integers from 0 to length of bin_val – 1 and stores it in a list index.
        - Complements all the bits at index j where j is in index list
        - Converts bin_val back to integer and assigns as variable value.
- Returns mutated input to **fuzzer.py**

LIMITATIONS & ASSUMPTIONS:

- There are no errors raised during execution of the input turtle program.

- ‘==’ condition have very less probability of being satisfied as fuzzer randomly mutates input, so a particular integer value being generated is highly unlikely.

- As mutator module has a randomizer involved to choose mutation method, for two executions of same turtle program with same initial seed inputs, coverage metric might differ if very small timeout value is decided.

- Timeout in seconds is at least enough for turtle program to execute about 8-10 times, so fuzzer can get good coverage.

- A timeout of 200 seconds gives 100% coverage majority of times if there are no dead statements. For example, in mytest1.tl there are 2 dead statements, that is they can never be reached and therefore are not executed so fuzzer coverage is less than 100%.