Contents

1 Problem Statement

It was fun to learn about fuzzing. Clever mutation operators along with nifty coverage metrics makes a naive fuzzer great. In this assignment, you are to implement a custom mutation operator along with a coverage metric operator (this operator will determines if there is a change/improvement in coverage metric when a turtle program is executed with mutated inputs.) for the fuzzer loop in `fuzzSubmission.py` file.

A basic `coverage guided fuzzer` loop has been implemented for you in `fuzzer.py` file.

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

```
- def compareCoverage(curr_metric, total_metric)
- def mutate(input_data) # input_data type is InputObject(...)
- def updateTotalCoverage(curr_metric, total_metric)
```

Points to note.

- Fuzzing needs initial seed values. Specify using ‘-d’ or ‘–params’ flag.
- Refer to `fuzzer.py` (KachuaCore/fuzzer.py) and `fuzzerInterface.py` (KachuaCore/interfaces/fuzzerInterface.py) file for better understanding. (This is optional)
- The last entry in the `Coverage List` is to be ignored if the program terminated successfully on the input run. https://github.com/CS639A-PAVT/BugTracker/issues/6

Running the `Fuzzer` Loop: (From `KachuaCore` folder)

```
$ ./kachua.py -t 100 --fuzz example/fuzz2.tl -d '{":x": 5, ":y": 100}'


# on termination. sample output
...
...
forward :move MoveCommand 1
  MoveCommand
...
[fuzz] Program took too long to execute. Terminated
[fuzz] Coverge for execution : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
[fuzz] Time Exhausted : 10.10099196434021
[fuzz] Terminating Fuzzer Loop.
Coverage : [],
Corpus:
Input 0 : {':x': 5, ':y': 100}
```

- Parameters: -t: Time budget for fuzzing in seconds. -d: Specify initial parameters.

1.1 Objective

Mutate the inputs in a way that it `maximizes` coverage within a small time budget. (Eg: Covers as many IR statements in the Turtle program as possible). We will use a timeout of 60 seconds atleast. Submit 5 interesting testcases on which you were able to maximize the coverage.

1.2 Deliverables

The source code of your implementation. 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 bug 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

1.3 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 Before doing anything “extra” (which might fetch bonus marks), first, complete the basic expectations from your implementation. Program analysis tools are expected to display their results in a user-friendly manner; a user would never like to use a 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).