DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KANPUR

Report: Assignment 1
CS639: Program Analysis Verification and Testing

Name: Prashant Kumar
Roll No: 231110036

Abstract

Fuzzing is a software testing approach that involves feeding the target software with random/semi-random inputs to achieve the maximum coverage of software (i.e. a program in this case). We try to achieve maximum coverage by mutating the seeding inputs such that they may cover different sets of IRs present in the program resulting in an increase in coverage.

Objective

The task in assignment 1 is to implement a **mutate** method (function inside CustomMutator class) that performs the mutation on inputs that may increase the coverage of the program also implementation of another method that identifies the improved coverage and updates the total coverage of the program is to be done.

Detailed understanding of the assignment

An input (to a program/software) covers some specific path during the execution of the program, and to find the bugs in a program (i.e. testing) we need to pass some set of inputs to the program that cover maximum (or possibly all) possible paths of the program to ensure that program is working correctly, this is useful also useful to analyze the behavior of a program on some critical/boundary case inputs.

One of the methods to do this type of testing is **fuzzing** which is performed by a fuzzer, A fuzzer is a program that feeds the inputs to target software and records the coverage matrix on that input then it mutates the input and repeats the same process to update the coverage matrix again and again upto a predefined time. The central task in implementing a fuzzer is to implement a cleaver mutation function that increases the chance of getting more coverage by mutating inputs in each iteration.

Assumptions

To implement the function for mutating the input and to record the coverage the assumptions are-

- The program (that is to be tested) has multiple branch conditions resulting true on the different values of inputs (i.e. negative/positive).
- The program is written in a language supported by **Kachua** (i.e. Kachua syntax[1] is followed and the extension of the file is **.tl**)

Implementation

The basic template for a function that is to be implemented is provided in path 'Submission/fuzzSubmission.py' the implementation details of each function are as follows-

A. Mutation function:

There are several approaches used for mutation (e.g. Simple arithmetic, byte flip, known integers, test case splicing), I tried implementing mutation by assigning known integers values, simple arithmetic, and also by assigning random numbers but finally in the submitted implementation I have used following 3 approaches to mutate the inputs, One of these approach is randomly decided to mutate the input in each run of mutate method.

- Left shifting: In this approach value of each input variable is multiplied by 4.

- BytesFlip: Assume an int number is of 4 bytes, select a random byte and perform shift operation by shifting it to 8 bits. Also can use 8 bits,16 bits with a constant stepover of one byte.

- In this approach, each value in input is assigned a random number within the range of 50 above and below that number itself(i.e. current_num-50 and current_num+50) this mutation function gives very good coverage, it also gives a twist in mutation by taking a random choice that is also dependent on previous random choice hence this mutation gives very good distribution of inputs that results in good coverage.

B. Compare Coverage Function:

Compare coverage function compares the current coverage matrix with the total coverage matrix. In the CompareCoverage method, curr_metric is the coverage done by input of that instance and total_metric is the total coverage of the fuzzer program.

This function returns **True** if the union of curr_metric and total_metric is having more coverage than total_metric, **False** otherwise.

C. Update Total Coverage Function:

If the program coverage including the coverage of the current input instance is greater than that of before(i.e. CompareCoverage methods returns True) then this function updates the value of total_metric indicating that total coverage is increased on this input instance.

Results

There are 5 sample examples (Programs) in the path: '\KachuaCore\example' (refer to "Assignment_1_231110036.zip" file) to run the fuzzer and record coverage of these programs. The **Interesting results**(increased coverage) as well as **Interesting inputs** for this coverage and corpus for these programs are as follows.

Note- (i) For the first example, the 5 Interesting inputs and their results are given and for other subsequent examples, 1 interesting input and corresponding example are given.

(ii) These results can vary on executing these programs again as mutation function mutates inputs randomly.

First example Results-
For the first example 'fuzz_program1.tl' **the 5 Interesting inputs and their results are-**

(1) with initial input: {':x': 2, ':y': -5} -

Coverage	[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, 28, 29, 33],

Corpus	Input 0 : {':x': 2, ':y': -5}
	Input 1 : {':x': 1, ':y': -39}
	Input 2 : {':x': -17, ':y': -39}
	Input 3 : {':x': 36, ':y': 13}
	Input 4 : {':x': -40, ':y': -2}

(2) with initial input: {':x': 2, ':y': 0} -

Coverage	[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, 28, 29, 30, 31, 32, 33],
Corpus	Input 0 : {':x': 2, ':y': 0}

	Input 1 : {':x': -28, ':y': 18}
	Input 2 : {':x': 8, ':y': 49}
	Input 3 : {':x': -6, ':y': 12}
	Input 4 : {':x': 17, ':y': -20}
	Input 5 : {':x': -19, ':y': -9}

(3) with initial input: {':x': 50, ':y': -10} -

Coverage	[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, 28, 29, 33],

Corpus	Input 0 : {':x': 50, ':y': -10}
	Input 1 : {':x': 47, ':y': -1}
	Input 2 : {':x': 24, ':y': 18}
	Input 3 : {':x': -1, ':y': 29}

(4) with initial input: {':x': -10, ':y': -7} -

Coverage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 23, 24, 25, 26, 27, 28, 29,31,32, 33],

Corpus	Input 0 : {':x': -10, ':y': -7}
	Input 1 : {':x': 2, ':y': -53}
	Input 2 : {':x': -3, ':y': -212}
	Input 3 : {':x': -40, ':y': -35}

(5) with initial input: {':x': 0, ':y': 23}-

Coverage	[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, 28, 29, 33],

Corpus	Input 0 : {':x': 0, ':y': 23}
	Input 1 : {':x': -36, ':y': 92}
	Input 2 : {':x': 0, ':y': -15}
	Input 3 : {':x': 36, ':y': 23}
	Input 4 : {':x': -22, ':y': -15}

Second example Results-
For the example 'fuzz_program2.tl' **the 1 Interesting input and corresponding result is-**

(1) with initial input: {':x': 0, ':y': -1} -

Coverage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 24, 25, 26],

Corpus	Input 0 : {':x': 0, ':y': -1}
	Input 1 : {':x': -39, ':y': -42}
	Input 2 : {':x': -36, ':y': 4}
	Input 3 : {':x': 43, ':y': -5}
	Input 4 : {':x': 8, ':y': 39}
	Input 5 : {':x': 0, ':y': 25}

Third example Results-
For the example 'fuzz_program3.tl**' the 1 Interesting input and corresponding result is-**

(1) with initial input: {':x': -10, ':y': -7, ':z': 17} -

Coverage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22],

Corpus	Input 0 : {':x': -10, ':y': -7, ':z': 17}
	Input 1 : {':x': 19, ':y': -22, ':z': 66}
	Input 2 : {':x': -57, ':y': -27, ':z': 19}
	Input 3 : {':x': 16, ':y': 41, ':z': -31}
	Input 4 : {':x': -8, ':y': 5, ':z': 55}

Fourth example Results-
For the example 'fuzz_program4.tl**' the 1 Interesting input and corresponding result is-**

(1) with initial input: {':x': -10, ':y': -7, ':z': 17} -

Coverage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

Corpus	Input 0 : {':x': -10, ':y': -7, ':z': 17}
	Input 1 : {':x': -59, ':y': 22, ':z': -1}
	Input 2 : {':x': 27, ':y': 25, ':z': 31}
	Input 3 : {':x': -6, ':y': 20, ':z': 4}
	Input 4 : {':x': 65, ':y': -6, ':z': 17}
	Input 5 : {':x': 9, ':y': -2, ':z': -20}

Fifth example Results-
For the example 'fuzz_program5.tl' **the 1 Interesting input and corresponding result is-**

(1) with initial input: {':x': -10, ':y': -7, ':z': 17} -

Coverage	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 63, 64, 65, 66, 67, 68, 69, 70, 78, 79, 80, 81, 82, 83, 84, 85],

Corpus	Input 0 : {':x': -10, ':y': -7, ':z': 17}
	Input 1 : {':x': -38, ':y': 15, ':z': 7}
	Input 2 : {':x': 18, ':y': -55, ':z': 44}
	Input 3 : {':x': -35, ':y': 23, ':z': 29}
	Input 4 : {':x': -27, ':y': -26, ':z': 38}
	Input 5 : {':x': 18, ':y': -76, ':z': 77}
	Input 6 : {':x': -78, ':y': 64, ':z': 34}
	Input 7 : {':x': -34, ':y': 3, ':z': 62}

References

[1]: https://moodle.cse.iitk.ac.in/mod/resource/view.php?id=4564