Program Analysis, Verification, and Testing

Assignment 1

By: Dhiraj Pareek

231110012

Date: September 16, 2023

Introduction

This report describes the implementation, assumptions, and limitations of three functions:

- `mutateInput`
- `compareCoverage`
- `updateTotalCoverage`

`mutateInput` Function

Implementation

- This function is responsible for mutating the input data based on a randomly chosen mutation type.
- It supports three mutation types: "fliping_bits," "random_multiplication," and "swapping_variables."
- For "fliping_bits" mutation, it iterates through the keys of the input data dictionary, flips random bits within the values, and introduces randomness by occasionally making values negative.
- For "random_multiplication" mutation, it multiplies the values in the input data by random factors.
- For "swapping_variables" mutation, it shuffles the values in the input data dictionary while keeping the keys intact.

Assumptions

- The `mutateInput` function assumes that `input_data` is an object with a `data` attribute that is a dictionary.
- The function assumes that the mutation types "fliping_bits," "random_multiplication," and "swapping_variables" are applicable to the input data.

Limitations

- The `mutateInput` function introduces randomness in the mutation process, but it may not guarantee meaningful or context-aware mutations for all input data types.
- The mutation types used in `mutateInput` are relatively simple and may not cover all possible mutation scenarios, depending on the nature of the input data.

`compareCoverage` Function

Implementation

- This function calculates the new coverage percentage based on the current metric and the total metric.
- It checks if the total metric is empty. If it is, it returns True, indicating an improvement in coverage.

Assumptions

- The `compareCoverage` function assumes that the metrics (`curr_metric` and `total_metric`) are iterable collections, such as lists.
- It calculates the new coverage percentage by finding the difference between the current metric and the total metric and dividing it by the size of the current metric.
- It uses an improvement threshold of 1% and returns True if the new coverage percentage is greater than or equal to this threshold, indicating an improvement in coverage.

Limitations

- The `compareCoverage` function relies on the size of the metric collections, which may not provide a comprehensive measure of coverage improvement in all cases.
- The improvement threshold of 1% (0.01) is a fixed value and may not be suitable for all applications. Different scenarios may require different thresholds.

`updateTotalCoverage` Function

Implementation

- This function updates the total metric by adding the current metric's elements to it.
- It creates a new set containing elements from both the current metric and the total metric, ensuring that there are no duplicate elements.
- It returns the updated total metric.

Conclusion

In conclusion, the described functions provide essential functionality for mutation, coverage comparison, and updating coverage metrics. However, they have certain assumptions and limitations that should be considered in their usage.

Command

python kachua.py -t 100 –fuzz example/example1.tl -d {':x':5,':y':100, ':z':20, ':a':25}