Program Analysis Verification And Testing

(CS 639)

ASSIGNMENT - 1

Ruby Prajapati
231110042
Dept - CSE (M.tech)

Objective

Mutate the inputs in a way that it `maximizes` coverage within a small time budget.

Solution :

Implementation :

For achieving maximum coverage we have to implement three functions, one for comparing matrices , one for extending coverage matrices and other one for mutating input,.

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

def compareCoverage(curr_metric, total_metric:

The function compareCoverage appear to be a method of a class (indicated by self parameter). This method is designed to compare two sets of matrices , curr_matrix and total_matrix and determine if the current set of matrices represents an improvement in coverage compared to the total set of matrices .It returns True if there is an improvement in coverage and false otherwise.

def updateTotalCoverage(curr_metric, total_metric)

The function updateTotalCoverage appears to be a method of a class (indicated by the self parameter). This method is designed to update the total_metric by merging it with the curr_metric and returning the updated total_metric as a list. It ensures that there are no duplicate entries in the total_metric list by converting it to a set and then back to a list.

def mutate(input_data):

The function mutate appears to be a method of a class (indicated by the self parameter). This method is designed to mutate the input_data by applying random mutations to its values while keeping track of code coverage information. It operates on a dictionary of variable names and their corresponding integer values in input_data.data. Here's a step-by-step explanation of what this function does:

It takes three arguments as input: input_data, coverageInfo, and irList. input_data is a dictionary of variables and their integer values, coverageInfo is of a certain type (presumably related to code coverage), and irList is a list of IR (Intermediate Representation) statements.

Inside the function, it creates a deep copy of the input_data dictionary, which is stored in the variable mutated_data. Using a deep copy ensures that the original input_data is not modified.

It then iterates through the keys (variable names) in the mutated_data.data dictionary.

For each variable, it generates a random number between 0 and 3 (inclusive) using random.randint(0, 3) to determine the type of mutation to apply.

Depending on the random number generated, it applies one of the following mutations:

If random_number is 0, it performs a bitwise XOR operation (^) with a randomly generated number (num) between -128 and 128.

If random_number is 1, it shifts the bits of the variable to the left by one position (<< 1) and then applies a bitwise XOR (^) with num.

If random_number is 2, it shifts the bits of the variable to the right by one position (>> 1) and then applies a bitwise XOR (^) with num.

If random_number is 3, it directly applies a bitwise XOR (^) with num.

Finally, it updates the input_data variable with the mutated_data and returns the mutated input_data.

Limitations:

1. **Range of inputs :**

   Main limitation of this mutation function is that inputs are bounded in some range , input will not increase or decrease after some range.

2. **No Control Over Specific Variables:**

   The function randomly selects variables to mutate. If you need to target specific variables or types of variables for mutation, additional logic would be required.

3. **Randomness :**

   We can not predict the next input data because of the randomness of the program

4. It can not generate fractional input values.