Institute of Information Technology
University of Dhaka

Topic: BVC and Equivalence Class Testing

Software Testing and Quality Assurance (SE-605)

Submitted by:

Mashiat Amin Farin - 1202

Sampad Sikder - 1219

Mohammad Momenuzzaman - 1227

Monayem Sarker - 1228

Ahnaf Mubashshir Mobin - 1232

Submitted to:

Kishan Kumar Ganguly
Assistant Professor
Institute of Information Technology
University of Dhaka

Submission Date: 11 November, 2023

Introduction

Boundary Value Testing and Equivalence class are Black-Box testing techniques used to test the validity of outputs in a system, given a set of inputs.

Boundary Value Testing: BVA is considered a technique that uncovers the bugs at the boundary of input values. Here, a range is defined for the input domain and test cases are designed in such a way so that functions can be evaluated at the boundaries of that input domain. There are a few techniques for boundary value testing:

1. Boundary value checking
2. Robustness testing method
3. Worst case testing method

Worst case testing method considers all test cases and is considered as the superset of boundary value checking and worst case testing method. It generates a total of **5^n test cases.**

Equivalence Class Testing: Equivalence class testing divides the test cases into classes based on the validity of outputs. It is used so that for each specific class, there is no redundancy of checking and only one test case per equivalence class is executed. Equivalence class should ensure that there is no redundancy of test cases.

User Manual

Installation:
In the project directory:
Run the following command to install all dependencies:
pip install -r requirements.txt

Then use the following command from root location of the project to run:

python main.py
Or
python3 main.py

Based on your python version
And if you install any new library just run this command after installing:

pip freeze > requirements.txt

Equation parsing:
User is prompted to enter an equation at the beginning of the code. The equation can contain multiple variables.
The input equation is then parsed by the equation parser. Equation parser separates the left-hand side and right-hand side of the equation and then finds the unique symbols in the equation.

For example:

4*x + 6*y + 8*z = 54. This equation has three unique symbols.

Equation parser also looks for trigonometric or logarithmic functions in the equation. There are separate methods for handling these functions and these methods are checked using BVC and Equivalence class testing.

```
Enter an equation (e.g., '4*x + x*x + tan(x) + log(10) + 1 = 13'): 4*x + 6*y + 8*z = 54


Enter the low boundary value for symbol 'x': 3
Enter the high boundary value for symbol 'x': 10


Enter the low boundary value for symbol 'y': 20
Enter the high boundary value for symbol 'y': 30


Enter the low boundary value for symbol 'z': 10
Enter the high boundary value for symbol 'z': 20


Boundary Value Table:
{'Symbol': 'x', 'Min': 3.0, 'Max': 10.0, 'Min+': 4.0, 'Max-': 9.0, 'Max+': 11.0, 'Min-': 2.0, 'Nominal': 6.5}
{'Symbol': 'y', 'Min': 20.0, 'Max': 30.0, 'Min+': 21.0, 'Max-': 29.0, 'Max+': 31.0, 'Min-': 19.0, 'Nominal': 25.0}
{'Symbol': 'z', 'Min': 10.0, 'Max': 20.0, 'Min+': 11.0, 'Max-': 19.0, 'Max+': 21.0, 'Min-': 9.0, 'Nominal': 15.0}
```

Figure: Equation and boundary value input with generation of Boundary Value Table

Boundary value input:

For each unique variable, the user is prompted to enter the boundary values. The test cases are generated by the **Worst Case Testing** method since it is the superset of all testing methods.

After that **Robust Testing method** is used to generate the invalid test cases.

The valid and invalid test cases are shown separately in the output.

```
Test-Case-id 123: x=6.5 y=25.0 z=11.0 , result: 264.0
Test-Case-id 124: x=6.5 y=25.0 z=19.0 , result: 328.0
Test-Case-id 125: x=6.5 y=25.0 z=15.0 , result: 296.0


Invalid  cases:
Test-Case-id 1: x=3.0 y=20.0 z=21.0 , result: Out of Range Error
Test-Case-id 2: x=3.0 y=20.0 z=9.0 , result: Out of Range Error
Test-Case-id 3: x=3.0 y=30.0 z=21.0 , result: Out of Range Error
Test-Case-id 4: x=3.0 y=30.0 z=9.0 , result: Out of Range Error
Test-Case-id 5: x=3.0 y=21.0 z=21.0 , result: Out of Range Error
Test-Case-id 6: x=3.0 y=21.0 z=9.0 , result: Out of Range Error
Test-Case-id 7: x=3.0 y=29.0 z=21.0 , result: Out of Range Error
Test-Case-id 8: x=3.0 y=29.0 z=9.0 , result: Out of Range Error
Test-Case-id 9: x=3.0 y=31.0 z=10.0 , result: Out of Range Error
Test-Case-id 10: x=3.0 y=31.0 z=20.0 , result: Out of Range Error
Test-Case-id 11: x=3.0 y=31.0 z=11.0 , result: Out of Range Error
Test-Case-id 12: x=3.0 y=31.0 z=19.0 , result: Out of Range Error
Test-Case-id 13: x=3.0 y=31.0 z=21.0 , result: Out of Range Error
Test-Case-id 14: x=3.0 y=31.0 z=9.0 , result: Out of Range Error
```

```
Valid  cases:
Test-Case-id 1: x=3.0 y=20.0 z=10.0 , result: 212.0
Test-Case-id 2: x=3.0 y=20.0 z=20.0 , result: 292.0
Test-Case-id 3: x=3.0 y=20.0 z=11.0 , result: 220.0
Test-Case-id 4: x=3.0 y=20.0 z=19.0 , result: 284.0
Test-Case-id 5: x=3.0 y=20.0 z=15.0 , result: 252.0
Test-Case-id 6: x=3.0 y=30.0 z=10.0 , result: 272.0
Test-Case-id 7: x=3.0 y=30.0 z=20.0 , result: 352.0
Test-Case-id 8: x=3.0 y=30.0 z=11.0 , result: 280.0
Test-Case-id 9: x=3.0 y=30.0 z=19.0 , result: 344.0
Test-Case-id 10: x=3.0 y=30.0 z=15.0 , result: 312.0
Test-Case-id 11: x=3.0 y=21.0 z=10.0 , result: 218.0
Test-Case-id 12: x=3.0 y=21.0 z=20.0 , result: 298.0
Test-Case-id 13: x=3.0 y=21.0 z=11.0 , result: 226.0
Test-Case-id 14: x=3.0 y=21.0 z=19.0 , result: 290.0
Test-Case-id 15: x=3.0 y=21.0 z=15.0 , result: 258.0
Test-Case-id 16: x=3.0 y=29.0 z=10.0 , result: 266.0
Test-Case-id 17: x=3.0 y=29.0 z=20.0 , result: 346.0
Test-Case-id 18: x=3.0 y=29.0 z=11.0 , result: 274.0
```

Figure: Generation of Valid and Invalid Cases

Custom test case:

Users can input a custom test case to check for the validity of any set of inputs.

```
Do you want to try a custom test case? (Y/n): y
Enter value for symbol 'x': 2
Enter value for symbol 'y': 10
Enter value for symbol 'z': 20
The case is Invalid.
The error is:  Out of Range Error
Try another one? (Y/n):
```

Figure: Input Custom Test Case

Trigonometric and Logarithmic function handling:

The program has the functionality for testing trigonometric and logarithmic functions. There are separate methods for these functions whose input and output is tested for validation.

To trigger this checking, simply input trigonometric and logarithmic functions in the prompt where it asks you to input an equation. For trigonometric functions, **boundary values must be provided in degrees. It is later converted to radians.**

```
Enter an equation (e.g., '4*x + x*x + tan(x) + log(10) + 1 = 13'): tan(x) + cot(y) = 0


Enter the low boundary value in degrees for symbol 'cot_y': 0
Enter the high boundary value in degrees for symbol 'cot_y': 180


Enter the low boundary value in degrees for symbol 'tan_x': 0
Enter the high boundary value in degrees for symbol 'tan_x': 180


Boundary Value Table:
{'Symbol': 'cot_y', 'Min': 0.0, 'Max': 3.141592653589793, 'Min+': 0.017453292519943295, 'Max-': 3.12413936106985, 'Max+': 3.1590459461097367, 'Min-': -0.017453292519943295, 'Nominal'
: 1.5707963267948966}
{'Symbol': 'tan_x', 'Min': 0.0, 'Max': 3.141592653589793, 'Min+': 0.017453292519943295, 'Max-': 3.12413936106985, 'Max+': 3.1590459461097367, 'Min-': -0.017453292519943295, 'Nominal'
: 1.5707963267948966}
```

Figure: Input Trigonometric or Logarithmic Function

```
Test-Case-id 8: cot_y=1 tan_x=179 , result: 57.27250656583121
Test-Case-id 9: cot_y=179 tan_x=0 , result: -57.28996163075968
Test-Case-id 10: cot_y=179 tan_x=180 , result: -57.28996163075968
Test-Case-id 11: cot_y=179 tan_x=1 , result: -57.272506565831456
Test-Case-id 12: cot_y=179 tan_x=179 , result: -57.3074166956879
Test-Case-id 13: cot_y=90 tan_x=0 , result: 6.123233995736766e-17
Test-Case-id 14: cot_y=90 tan_x=180 , result: -6.123233995736766e-17
Test-Case-id 15: cot_y=90 tan_x=1 , result: 0.017455064928217648
Test-Case-id 16: cot_y=90 tan_x=179 , result: -0.017455064928217447


Invalid  cases:
Test-Case-id 1: cot_y=0 tan_x=0 , result: Math Domain Error
Test-Case-id 2: cot_y=0 tan_x=180 , result: Math Domain Error
Test-Case-id 3: cot_y=0 tan_x=1 , result: Math Domain Error
Test-Case-id 4: cot_y=0 tan_x=179 , result: Math Domain Error
Test-Case-id 5: cot_y=0 tan_x=181 , result: Out of Range Error
Test-Case-id 6: cot_y=0 tan_x=-1 , result: Out of Range Error
Test-Case-id 7: cot_y=0 tan_x=90 , result: Math Domain Error
Test-Case-id 8: cot_y=180 tan_x=181 , result: Out of Range Error
Test-Case-id 9: cot_y=180 tan_x=-1 , result: Out of Range Error
Test-Case-id 10: cot_y=180 tan_x=90 , result: Math Domain Error
```

Figure: Boundary Value Checking and Equivalence Class Testing for Trigonometric Functions

Example function for which assertion is checked is as follows:

```
`
    def replace_tan_function_variables(self, match):
        variable = match.group(1)
        new_name = f"tan_{variable}"
        return f"tan({new_name})"

    def replace_sin_function_variables(self, match):
        variable = match.group(1)
        new_name = f"sin_{variable}"
        return f"sin({new_name})"

    def replace_cos_function_variables(self, match):
        variable = match.group(1)
        new_name = f"cos_{variable}"
        return f"cos({new_name})"
```

Figure: Functions for which output is checked

Conclusion

In conclusion, our project focuses on the development of a robust Boundary Value and Equivalence Class Testing Tool for software testing and quality assurance. The tool is designed to enhance the testing process by effectively identifying bugs and validating the output of a system based on a set of inputs.

Github repository link:

https://github.com/AhnafMubashshir/Boundary-Value-Equivalence-Testing-Tool