CSI 5137A – AI-enabled Software Verification and Testing
Assignment 2, Autumn **2024**

Due date: **Nov 11th, 2024**

The assignment files have to be submitted on BrightSpace by Nov 11th, 2024 midnight.

1 Background

In this assignment, you investigate and extend an implementation of a search-based test input generation approach. We use the AVMFramework (`http://avmframework.org`) for this assignment. The actual implementation of the framework is available at this github project page (`https://github.com/AVMf/avmf`). AVMf is a framework and Java implementation of the Alternating Variable Method (AVM), a heuristic local search algorithm that has been applied to several important software engineering problems. The framework includes several example engineering problems that can be automated using AVM. You can find the problem instantiations in the `examples` directory of the project. Among the different examples in the `examples` directory, we are interested in the test input generation problem implemented in `GenerateInputData.java`.

To become familiar with Alternating Variable Method (AVM), read the github project page and clone/download the project, and further, read the slides (avm-test generation.pdf) and the paper (avm-paper.pdf) that are attached to this assignment handout. In addition, to understand how the test input generation works, you can try to execute `GenerateInputData.java` for the three example input Java classes provided in the `inputdatageneration` directory: `Line`, `Triangle` and `Calendar`. Note that the instrumented versions of these three Java classes are provided in three classes named respectively `LineBranchTargetObjectiveFunction`, `TriangleBranchTargetObjectiveFunction` and `CalendarBranchTargetObjectiveFunction`. The instrumented classes help compute the standard fitness function for branch coverage for any branch in the original class. Further, the test inputs for these classes are, respectively, specified in `LineTestObject`, `TriangleTestObject` and `CalendarTestObject`. The information on how `GenerateInputData.java` can be used to generate test inputs for these three Java classes are available on the github page (`https://github.com/AVMf/avmf`) under the **Test Data Generation** section.

2 Your Tasks

For this assignment you need to answer the following questions and extend the above framework with a new search algorithm. The answer to questions should be submitted as a .pdf file. The required deliverables for the implementation task are specified below.

2.1 Question 1.

How does the Alternating Variable Method (AVM) work? How is this algorithm different from the Hill Climbing or Simulated Annealing algorithms we discussed in the class? Please try to describe the main ideas and the working of AVM in less than one page (using a few paragraphs).

2.2 Question 2.

Show the control-flow graph for the classify method of the Triangle class and label each branch with an id such that your branch ids are consistent with those specified in `TriangleBranchTargetObjectiveFunction`. Note that each branch id should be a number followed by T (for the true branch) or F (for the false branch).

2.3 Question 3.

Provide a test suite that achieves branch coverage for the classify method of the Triangle class.

2.4 Question 4.

Provide a smallest test suite that can achieve statement coverage for the code below. Does this test suite achieve branch coverage as well? If yes, for each test case in your test suite, specify the branches covered by the test case. Otherwise, provide a smallest test suite that can achieve branch coverage for this code.

```
Add (int a, int b) {
        if (b > a) {
           b = b - a
           Print b
        }
        if (a > b) {
           b = a - b
           Print b
        }
        if (a = b) {
           if (a = 0) {
              Print '0'
           }
        }
}
```

2.5 Question 5.

Provide a test suite for the intersect method of the Line class that achieves statement coverage, but not branch coverage. Explain which branches of the intersect method are not covered by your test suite.

2.6 Question 6.

The fitness function to compare different candidate test inputs is defined based *approach level* and *branch distance* metrics. Explain how these two metrics are combined to compare different test input vector candidates and specify in which method of which Java class in the AVMf framework, this comparison is implemented.

2.7 Question 7.

As discussed in the class, one way to combine approach level and branch distance metrics is to first nomralize branch distance and then add it to approach level. Why do we need to normalize the branch distance metric but not the approach level?

2.8 Implementation Task.

Implement one of the local search algorithms we learned in the class (e.g., Hill Climbing and Simulated Anneal) in the AVMf framework and modify `GenerateInputData.java` to use your new local search algorithm to generate test input data. Do not modify the input interface of `GenerateInputData.java` though. That is, your modified version of `GenerateInputData.java` should still generate test inputs for branch "1T" of Triangle by passing `1T Triangle` to it as input. Note that you do not need to use the optional `search` as input to determine the type of search algorithm since `GenerateInputData.java` should call the new search algorithm that you have implemented by default.

For the implementation task, you should submit the following deliverables by the submission deadline:

- **Implementation:** The modified AVMf framework that generates test inputs using a new local search algorithm that you have implemented. Make sure that your submission is self-contained and compiles and executes without any problem.
- **Report:** A written report that contains a detailed description of your new search algorithm and how it is used for test generation.