SE 421: Assignment #3

Due on September 17, 2018 at 12:00 PM (noon)

Instructor: Ben Holland

https://github.com/SE421/assignment3

Student Name:

Problem 1

(10 points)

A. (*2 points*) How many paths exist in a control flow graph with `n` *non-nested* branches? Explain.

B. (*2 points*) How many paths exist in a control flow graph with `n` *nested* branches? Explain.

C. (*2 points*) How many paths could `n` logical short circuiting operators (`&&` or `||`) introduce if each logical short circuiting operator is part of an expression to a branch condition that consists only of logical short circuiting operators and variable values (examples: `if(c1 && c2 && c3)` or `if(c1 && c2 || c3)`)? Explain.

D. (*4 points*) The following statements each describe a node in a control flow graph. Answer with justification whether a node which fits the description can or can not exist in a control flow graph.

   (a) A node with one incoming edge and one outgoing edge

   (b) A node with two incoming edges and two outgoing edges.

   (c) A node with one incoming edge and three outgoing edges.

   (d) A node with three incoming edges and two outgoing edges.

Problem 2

(10 points)

A. (*8 points*) Path Enumeration

In this problem, you are required to modify the provided depth-first search (DFS) implementation (`PathEnumerator.java`) to count the number of control flow paths in an acyclic control flow graph (CFG) for C functions. You will modify the DFS algorithm to actually enumerate the paths by producing the program trace (the sequence of line numbers corresponding to program statements) along each path starting from the CFG root to a CFG leaf. Your implementation should adhere to the following requirements:

- Your implementation should be a modification to the provided `DFSPathEnumerator.java`.
- Your implementation should extend the `PathEnumerator` class and implement all abstract methods. Note that an `EnumerationResult` contains a `List<List<Long»` (paths), which represents a list of paths where each path is a list containing the line number trace. The `EnumerationResult` also contains a `CountingResult` that has a value `additions`, which is the number of addition operations performed during the counting.
- A trace should be computed as the set of line numbers from the CFG root to a given CFG leaf in the DFS. If a node in the graph is not explicitly represented by a line number in the source then report the line number as `-1`.
- The modified program *must* explore `TRUE` edges before `FALSE` edges (your implementation should traverse paths through switch statements in a natural ordering corresponding to the switch primitive type).
- Your implementation does not need to consider inter-procedural paths, implicit paths introduced due to logical short circuiting, or ternary operators. You do not need to consider path feasibility.

Commit your changes to the code on your GitHub assignment repository and briefly explain your modifications. Remember to provide a direct link to your source code on GitHub in your report!

The provided code is an Eclipse plugin (`vogella.com/tutorials/EclipsePlugin/article.html`) that depends on Atlas (`http://www.ensoftcorp.com/atlas`). You will be using the Atlas SDK (`https://www.ensoftcorp.com/atlas/sdk`) and its XCSG graph schema (`http://ensoftatlas.com/wiki/Extensible_Common_Software_Graph`) to extend the functionality of the provided plugin project. Example code for detecting true and false edges can be found in the `CFGPrinter.java` along with other resources that may be useful in the `com.se421.paths.support` package.

B. (*2 points*) Unit Testing

You are expected to think about and document a good testing strategy to validate the correctness of your implementation. A JUnit test framework with some example unit tests have been provided. To run the unit tests, right click on the `AllTests.java` file in the `com.se421.paths.regression` project and select `Run As` → `JUnit Plug-in Test`. You should extend the `com.se421.paths.regression` project by creating at least three new unit tests. Briefly explain your testing strategy for each unit test in your report and provide a GitHub link to each unit test. Note that you are allowed to share and discuss test cases with other students as long as you cite your sources and explain the included content in your own words!

Problem 3

Extra Credit (10 points)

A. (*10 points*) `O(n)` Path Counting

The number of paths in an acyclic CFG may be exponential. The provided DFS implementation counts paths incrementally, one at a time, from 1 to `n`, which will be very expensive for graphs where `n` is an exponential number of paths. In this problem you should implement the multiplicities algorithm outlined during the lectures to count paths in linear time. Note that you do not need to adhere to the requirements of Problem 2.

Modify the `MultiplicitiesPathCounter.java` source with your solution. Briefly describe your implementation and provide a direct link to your source code on your GitHub assignment repository.