SE 421: Assignment #5

Due on October 8, 2018 at 12:00 PM (noon)

Instructor: Ben Holland

https://github.com/SE421/assignment5

Student Name:

Problem 1

(14 points)

Review the class lecture notes and the papers [1, 2, 3] and then answer the questions below.

1. *(10 points)* For each analysis technique below clearly and concisely describe how the algorithm works in at most five sentences of *your own words*.
   (a) Rapid Type Analysis (RTA)
   (b) Reachability Analysis (RA)
   (c) Class Hierarchy Analysis (CHA)
   (d) 0-CFA Andersen-style Variable Type Analysis (VTA)
   (e) Hybrid Type Analysis (XTA)

2. *(2 points)* Arrange the algorithms above in order of least expensive to most expensive. Justify your ordering.

3. *(2 points)* Arrange the algorithms above in order of least precise to most precise. Justify your ordering.

References

[1] J. Dean, D. Grove, and C. Chambers, "Optimization of object-oriented programs using static class hierarchy analysis," in *European Conference on Object-Oriented Programming*. Springer, 1995, pp. 77–101.

[2] D. F. Bacon and P. F. Sweeney, "Fast static analysis of c++ virtual function calls," *ACM Sigplan Notices*, vol. 31, no. 10, pp. 324–341, 1996.

[3] F. Tip and J. Palsberg, *Scalable propagation-based call graph construction algorithms*. ACM, 2000, vol. 35, no. 10.

Problem 2

(6 points)

(*5 points*) Call Graph Construction

In this problem, you are required to modify the provided skeleton code (`ReachabilityAnalysis.java`) to implement a reachability analysis (RA) call graph. Complete the task described by the commented "TODO". 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 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. You may wish to use tools or utilities from your previous assignment.

(*1 points*) Experimentation

What call graph construction algorithm is Atlas likely using? Gather evidence and support your answer by constructing test programs, mapping the test programs with Atlas, and comparing Atlas' `XCSG.Call` edges in mapped programs with your RA call graph implementation.

Using your RA call graph implementation run your plugin project as an Eclipse Plugin application. Eclipse will launch a new runtime copy of Eclipse. The skeleton project includes a source file called `CallGraphConstructionCodemapStage.java`, which configures your call graph construction code as well as an example implementation of a Class Hierarchy Analysis (CHA) implemented in `ClassHierarchyAnalysis.java` to run automatically after Atlas produces a codemap for a project. Finally, the skeleton project includes two custom Atlas "Smart View" defined in `RACallGraphSmartView.java` and `CHACallGraphSmartView.java` that allows you to select a method node and view the corresponding call graph from the selected node.

(*0 points*) Unit Testing

An empty JUnit test framework has been provided. To run the unit tests, right click on the `com.se421.cg.regression` project and select `Run As` → `JUnit Plug-in Test`. To fully test your code you should extend these unit tests with your own test cases! 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 (30 points)

For any of the following call graph construction techniques, 1) implement the technique in Atlas, 2) document your implementation, and 3) provide example code snippets and call graphs that compare an contrast the technique to a class Hierarchy analysis. Note that no points will be given without an accompanying implementation.

A. (*10 points*) Rapid Type Analysis (RTA)
Implement RTA as described by David Bacon and Peter Sweeney in *Fast Static Analysis of C++ Virtual Function Calls*.

B. (*10 points*) Rapid Type Analysis Variants (FTA, MTA, or XTA)
Implement FTA, MTA, or XTA as described by Tip, Frank, and Jens Palsberg in *Scalable propagation-based call graph construction algorithms*.

C. (*10 points*) Variable Type Analysis (VTA)
Implement VTA using your points-to analysis with type inference from assignment 4.