Homework 5 (Programming): Context-Sensitive Interprocedural Analysis

17-355/17-665/17-819: Program Analysis
Claire Le Goues
clegoues@cs.cmu.edu

Due: Thursday, March 5, 2020 (11:59 PM) 200 points total

Assignment Objectives:

- Implement a context-sensitive, interprocedural dataflow analysis.
- Handle the complexities of analyzing a real programming language.
- Make use of a real framework for analyzing Java code.

Handin Instructions. Submit your *entire* GITHUB repository for this assignment following the instructions on GradeScope. Note that this is different from the last coding assignment, where we asked you to submit a single file. Your grade will be based on a combination of the autograder tests your code passes, and certain manual considerations, described at the end of this document.

Note that we do *not* plan to look directly at your GITHUB repository for this assignment. You *must* submit your code via Gradescope. Gradescope does not automatically pull new versions of your code, so you must resubmit whenever you have a new version that you would like graded.

1 Context-Sensitive Interprocedural Analysis Implementation

In this assignment, you will implement (and test!) a context-sensitive interprocedural integer sign analysis for Java, using the simpler/less precise domain we implemented in homework 3.[1] Implement context-sensitivity using the *call string approach* with a maximum depth of 2.

We provide starter code for this assignment based on the Soot framework. We have provided a bit less scaffolding than we did last time, but there are still clear TODOs in comments indicating where you should begin your implementation. In particular, you need to implement:

- In `Context`, the **`getCtx`** method.
- In `Sigma`, the **`equals`** and **`hashCode`** methods(s).
- The majority of `IntraSignAnalysis`, which implements the intra-procedural part of the analysis. Note that you do *not* need to explicitly implement Kildall's, because Soot provides it at the backend; familiarize yourself with the framework, and look over the starter code to

[1] If you would prefer to implement a different analysis, or target a different (real!) programming language, contact the course staff with a concrete proposal regarding analysis/language/framework and we will try to accomodate you.

see what you *do* need to implement (e.g., flow functions, join, etc). *Unlike in homework 3*, you *do* need to implement a **`reportWarnings`** method, see below.

- The majority of `InterSignAnalysis`.

On language. In Java, integer variables are separate from variables that hold references, booleans, floating point values, etc. Your implementation need only track information for variables corresponding to type `int`, for local variables and method parameters. Your analysis should cover variable copies, integer constants, addition, subtraction, multiplication, and division as precisely as possible. You are not required to correctly analyze other operations, though your analysis should not crash on code that includes them. Your analysis should reason about local variables and method parameters; you may assume that globals, fields, or array accesses, are unknown.

Expected analysis output. At a high level, we expect the analysis to issue warnings when it identifies an array access that may involve a negative array index. This is the primary output we expect from your analysis implementation: a set of warnings for the code. We provide a standard method `Util.reportWarning` for reporting warnings, and there are example usages in `IntraSignAnalysis.java` that show how to call it. To see an example of how we will test this, you may look at the sample tests we provide in the starter code. The `src/test/inputs/` directory contains test inputs to test both intra- and interprocedural analyses; these also are commented where errors should be reported. The files `IntraAnalysisTest.java` and `InterAnalysisTest.java` show how we test the analysis results, and will be informative when you are writing your own.

Tests. We do provide sample tests, and we have a set of held-out autograder tests, as we did for homework 3. For full credit on the assignment, you must *also* implement additional tests for your analysis, covering the key implementation considerations you are tackling. One or more test cases should require context sensitivity—i.e., the test case would fail if your analysis were interprocedural not context-sensitive. Your tests should use JUnit and be automatically run with `gradlew test`. We will download your Gradescope submission and run your tests.

2 Setup and Tool Information

Go to https://classroom.github.com/a/9R7r-N4v to clone the starter code into your own private GITHUB account. Ensure you have installed the Java Development Kit version 8 or later.

This assignment uses the Gradle build automation system. Gradle generates wrapper scripts that automatically download any dependencies that are needed to run a project, including Gradle itself. To build the source code, you just need to run `./gradlew build` on *nix systems, or `gradlew.bat build` on Windows.

We have provided test cases, which you can run with `gradlew test`; passing the test cases is an indication that you are on the right track, though earning credit for the assignment requires implementing your own analysis in a general way so that it will also work correctly with other test programs.

For getting started with Soot's dataflow framework, have a look at the Github wiki page, here:
https://github.com/Sable/soot/wiki/Implementing-an-intra-procedural-data-flow-analysis-in-Soot

3 Grading

This assignment is worth 200 points in total. Grading will involve both testing, and some manual assessment. The rough expected distribution of maximum available points for the automatically graded components is:

- Correct implementation of **`getCtx`**: 20 pts
- Correct implementation of **`Sigma`**: 10 pts
- Correct implementation of **`IntraSignAnalysis`**: 60 pts
- Correct implementation of **`InterSignAnalysis`**: 80 pts

The rough expected distribution of points for the manually graded components is:

- Original tests that roughly cover the implemented functionality and run with `gradlew test`: 20 pts
- Good coding practices, including code structure and commenting: 10 pts

Partial credit will of course be available as well, for all of the above.