

# COMP 621 Program Analysis and Transformations Assignment #2

Using the Soot Framework for Analysis and Profiling

Due: Monday Feb 16, 2009

## Overview:

The purpose of this assignment is to give you some practice designing flow analyses, and to familiarize you with the Soot framework, so you will be comfortable with it for your course project. You have been given an introduction to the system during the lecture, and there is also documentation and tutorials on the web site [www.sable.mcgill.ca](http://www.sable.mcgill.ca), on Eric Bodden's blogs at <http://www.bodden.de/tag/soot-tutorial/> and the very useful "A Survivor's Guide to Java Program Analysis with Soot" found at <http://www.brics.dk/SootGuide/>.

Please feel free to ask questions of the graduate students in my research lab, the TA during his office hours, and also feel free to share information with your classmates on the google group. The objective of this assignment is to get everyone comfortable with the environment - so don't spend a lot of time getting stuck on some small technical issue.

## Question 1 - *Implementing an Analysis in Soot*

The Soot framework has a Flow Analysis Class that can be extended to implement any standard flow analyses. For this question you can choose one of the following analyses to implement. However, you are strongly encouraged to suggest your own analysis, just give a clear definition of it. You may find it useful to look at the parity or nullness analysis already in Soot for examples of analyses that have tags that can be viewed in Eclipse.

- **Must Reaching Definitions:** Implement the data flow analysis for the **must** reaching definition problem. In this problem a definition reaches, only if it reaches along **all** paths.
- **Faint Variable Analysis:** A variable is a faint variable if it is dead or if it is only used to calculate new values for faint variables; otherwise it is strongly live. As an example, in `x := 1; x := x-1; x := 2`, `x` will be faint after each of the three assignments (whereas it is dead after second and third assignment but it is live after the first assignment).
- **Signs Analysis:** A detection of signs analysis will model all the negative numbers by the symbol "-", zero by the symbol "0", and the positive numbers by the symbol "+". So the set `{-2,-1,1}` will be modeled by the set `{-,+}`, that is an element of  $P(\{-,0,+ \})$ . Specify a constant-propagation-like analysis to perform a detection of signs analysis.

You should implement the analysis and also use the tagging mechanism to allow you to visualize the results of your analyses using the Soot Eclipse framework.

For the purposes of this question you should submit a short description of your approach to implementing the analysis, the source code for your implementation (only those files you implemented), and some example screenshots of your analysis information displayed in the plug-in.

## Question 2 - *Profiling*

It is sometimes useful to insert instructions to profile the behaviour of programs. For example, it might be interesting to know how many virtual method calls a program makes. One way to accomplish this is to insert new statements in the Jimple representation of the program, where these statements increment the appropriate counters. Another way is to use JVMPI and a profiler such as \*J. A third way is to use Aspect Oriented Programming and AspectJ.

For this question, you will try profiling using Soot and also using AspectJ.

(a) **Using Soot:**

Implement profiling of some feature of your choice (for example, virtual method calls, allocations, array accesses, etc.). See the tutorial at <http://www.sable.mcgill.ca/soot/tutorial/profiler/> for guidance.

You should submit a short description of your approach to profiling, your source code, and some examples showing the Jimple before and after inserting the profiling statements. You should also try your profiler on your benchmark program (from assignment #1), and report the results of your profiler.

(b) **Using AspectJ:**

Implement profiling of some feature of your choice using Aspects. This does not need to be the same feature you profiled in (a), but it would be interesting to do the same feature in order to compare the results and the ease of implementing the profiling.

Write a brief comparison of profiling using AspectJ compared to a bytecode rewriting tool such as Soot. In particular, what would be needed to make AspectJ as flexible as bytecode rewriting? On the other hand, what are some advantages of AspectJ over a bytecode rewriting tool?

Apply your AspectJ profiler to your benchmark program from Assignemnt #1 and report the results. (For students who did a Matlab benchmark, you may borrow a Java benchmark program from another member of the class.)

You should submit a short description of your approach to profiling, your source code, your comparison and evaluation of using AspectJ vs. Soot, and the results of applying your profiler.