Assignment 2 (Written/Programming): Dataflow Analysis

15-819M: Program Analysis
Jonathan Aldrich (jonathan.aldrich@cs.cmu.edu)

Due: Monday, February 1, 2010 (1:30 pm)

100 points total

Turn in a zip file electronically in the Blackboard drop box. The zip file should contain the following files:

- answers.xxx - the answers to text questions in txt, pdf, or Word (doc) format. At the top of the document, state your name and andrew ID.
- output.xxx - either analysis output in .txt format or a screenshot in some common graphics format.
- project.zip - a zipped-up Eclipse project with your analysis implementation.

Assignment Objectives:

- Precisely define an analysis using a lattice and flow functions.
- Simulate analysis execution on a program using the worklist algorithm.
- Implement a dataflow analysis in a code framework built based on the concepts of flow functions and lattices.

1 Sign Analysis Definition (60 points)

Integer sign analysis tracks whether each integer in the program is positive, negative, or zero. The results of this analysis can be used to optimize a program or to circumvent errors like using a negative index into an array. The analysis is broadly similar to the zero analysis discussed in class. For the purposes of this assignment, we will ignore the possibility of integer overflow (i.e. consider mathematical integers).

Question 1.1 (10 points).

> Design a "precise" lattice for a single variable. Your lattice should track whether a value is less than zero, greater than zero, equal to zero, greater than or equal to zero, less than or equal to zero, non-zero, or unknown. Define the lattice by giving (a) the set of lattice elements and (b) the ordering relation between them, (c) the top element and (d) the bottom element.

Question 1.2 (6 points).

> Design a "less precise" lattice for a single variable. This lattice should only track whether a value is less than zero, greater than zero, equal to zero, or unknown (which in this case will include cases like greater than or equal to zero). Define the lattice by giving (a) the set of lattice elements and (b) the ordering relation between them, (c) the top element and (d) the bottom element.

Question 1.3 (3 points).

> What is the initial analysis information before the first statement of each function? Justify your choice (more than one answer may be correct, so long as it is justified).

Question 1.4 (15 points).

> Define a flow function for a multiplication of two variables assigned to a third variable, i.e. of the form $x = y * z$. Your flow function should be based on the second, simpler lattice. It should be as precise as possible given the analysis information available. You may define the flow function in any notation you like (e.g. mathematics, code, pseudo-code) as long as it is unambiguous.

Question 1.5 (6 points).

Assume you had an implementation of your sign analysis, as specified above. Explain how you would errors due to a negative array index. Specifically, assume a 3-address code operation of the form $x = y[z]$, and describe what condition on the analysis results just before such an operation would yield (a) a definite negative array index error and (b) a possible negative array index error (e.g. in cases where the analysis is too imprecise to tell if there is definitely an error).

Question 1.6 (20 points).

Simulate your analysis on the following program using a table as done in class. Your table should have a column for the program point, a column for the worklist, and a column for the abstract value of each variable. Each row should track the value after the execution of the corresponding statement. The rows should show how the analysis executes, examining one statement at a time:

```
1: x = 0;
2: y = 5;
3: z = -3;
   if (...)
4:     w = y * x;
   else
5:     w = x * z;
   while (...)
6:     z = y * z;
7:     y = w;
```

2 Sign Analysis Implementation (40 points)

Next, you will implement your valid pointer analysis for the Java programming language using Crystal's dataflow analysis capabilities.

In Java, integer variables are separate from variables that hold references, booleans, floating point values, etc. Your implementation need only track information for variables of type int. Your analysis only needs to track the sign of local variables. Any use of fields, arrays, method parameters or results should be considered to have unknown sign.

You should implement your analysis by defining a class to represent your simple lattice (i.e. the sign of a single variable), then use this with the TupleLatticeElement to build a tuple lattice. You will need to define operations on your lattice in a subclass of SimpleLatticeOperations (there is a corresponding class TupleLatticeOperations for your tuple lattice). You should define your transfer functions using a subclass of AbstractingTransferFunction. Finally, you should write a subclass of AbstractCrystalMethodAnalysis that creates a SimpleTACFlowAnalysis with your lattice and transfer function. An example of all of this is the edu.cmu.cs.crystal.analysis.npe.simpleflow package from the Crystal tutorial.

In order to drive the flow analysis, create a visitor that visits array indexing operations and produces a message using the text "error" or "warning" if the array index will definitely, or might possibly, be less than zero.

Your analysis should produce exactly one warning in the Eclipse errors window for each of the errors marked with comments in the test file TestSign.java and no additional warnings.

Your analysis should cover variable copies, integer constants, addition, subtraction, and multiplication as precisely as possible given your lattice. You are not required to correctly analyze other operations, though your analysis should not crash.

Question 2.1 (10 points).

Run your analysis on TestSign.java. Turn in a screenshot of the problems window, or the text produced by your analysis if you wrote to the Crystal console. When capturing the screenshot, resize the window if necessary to show all the errors. Do **not** change TestSign.java.

Question 2.2 (30 points).

Turn in your analysis code. Your code should follow the basic design described above.

We reserve the right to run your analysis code on examples other than TestSign.java, looking both for accuracy of the analysis and its robustness (i.e. it should not throw unexpected exceptions when run on a larger codebase).