Assignment 11 (Written): Dataflow Analysis

17-654/17-754: Analysis of Software Artifacts
Jonathan Aldrich (jonathan.aldrich@cs.cmu.edu)

100 points total

Turn in a file named <username>-17654-A11.{txt,pdf,doc}, where username is your Andrew id. At the top of the file, state your name, Andrew id, and how long you spent on the assignment.

Assignment Objectives:

- Precisely define two analyses using lattices, abstraction functions, and flow functions.
- Simulate analysis execution on a program using the worklist algorithm.
- Demonstrate understanding of analysis soundness principles.

1 Lock Analysis (45 points)

After graduating from CMU, you have been hired by FlowSoft, a (fictional) company devoted to bringing the benefits of static dataflow analysis tools to C programmers. Your first task is to get a simple lock analysis up and running.

You quickly observe that C presents a harder problem than Java, because there's no built-in synchronization statement. Thus it's easy to make simple errors that can't happen in Java, like locking a lock when you enter a function and forgetting to unlock it when you return from that function. Your first task, therefore, is to design an analysis that can detect simple errors like deadlock, which will occur if the programmer tries to lock the same lock twice. For this assignment, you need only consider one thread running at a time–believe it or not, double-locking errors due to a single thread that forgets to unlock a lock have been found in the Linux kernel, causing the system to hang.

You study the problem first in the context of the WHILE language. You model locks with two new kinds of statements:

- lock(x) locks the variable x
- unlock(x) unlocks the variable x

For the purposes of data flow analysis over the control flow graph, you can assume these statements turn into similar three address code statements, lock(x) and unlock(x). You decide you will base your analysis on a tuple lattice, with one element of the tuple for each lock variable in the program.

Question 1.1 (10 points).

> Design a lattice for a single variable. Your lattice should be able to represent both locked and unlocked states. 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 (5 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 1.3 (10 points).

Define the flow functions for your analysis, using the notation given in class. Naturally, you will need to include flow functions for the new `lock(x)` and `unlock(x)` statements.

Question 1.4 (10 points).

Assume you had an implementation of your lock analysis (from above). Explain how you would identify deadlock errors based on the results of this analysis. A deadlock occurs when a program locks a variable that is already locked. Specifically, describe (a) the While AST element that is associated with the error, (b) what condition on the analysis information at that location indicates a definite deadlock error, and (c) whether your condition is based on the analysis information immediately before or after the AST element. Finally, (d) explain what you would change about the condition to find a *possible* deadlock error (e.g. in cases where the analysis is too imprecise to tell if there is definitely an error).

There is a corresponding double-unlocking error that could be found, but finding it is not required for this assignment.

Question 1.5 (10 points).

> Simulate your analysis on the following program, using the worklist algorithm. Use the notation from the lecture 19, slide 7 ("Example of Worklist"), so you have 4 columns: the first describing to which statement you are applying the flow function (with 0 at the beginning to show the dataflow values at the entry of the CFG), the second column showing the statements on the worklist after analyzing this statement, and the last two columns showing lock lattice values for each variable (x and y—the variable b is not relevant since it is not a lock) after the analysis of this statement.

[lock(x)]$_1$;
if [$b > 0$]$_2$
  then [lock(y)]$_3$
  else [skip]$_4$;
[lock(x)]$_5$;
if [$b > 0$]$_6$
  then [unlock(y)]$_7$
  else [skip]$_8$;
[unlock(x)]$_9$;

2 Valid Pointer Analysis Specification (35 points)

Now it is time to precisely define a valid pointer analysis for WHILE. A valid pointer is a pointer which it is safe to dereference (i.e. it is not null and does not point to garbage). Assume that WHILE has been extended with the following syntactic construct:

$$\begin{array}{rcl} a & ::= & ... \\ & | & \&x \end{array}$$

The $\&x$ expression takes the address of some variable x, resulting in a valid pointer. For the present, we will assume WHILE has a C-like pointer model, where regular integer variables can hold pointer values. We represent null with the constant 0. We also assume that integer constants–including zero–are not valid pointers. For the purposes of data flow analysis over the control flow graph, you can assume that this new expression type turns into a three address code statement of the form $y = \&x$.

Question 2.1 (15 points).

Design a lattice for a single variable. Your lattice should be able to represent both definitely valid and definitely not valid states, as well as possibly valid. 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 2.2 (5 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 2.3 (15 points).

Define the flow functions for your analysis, using the notation given in class. Naturally, you will need to include flow functions for the new three address code statements of the form $y = \&x$, as well as for statements like $y = 0$ which loading integer constants. You should assume that an integer constant is an invalid pointer; the only valid pointers are those that come from the address-of operator.

3 Local Soundness (20 points)

Consider the following hypothetical flow function for constant propagation:

$$f_{\text{CP}}(\sigma,\ x{=}y{+}z) = [x \mapsto (\sigma(y) -_{\top} \sigma(z))]\sigma$$

Here the operator $-_{\top}$ is like $op_{\top}$ defined in lecture: the same as $-$ for integers, but yeilding $\top$ if either of its operands is $\top$.

This flow function is "obviously" incorrect. However, to prove that it is incorrect, you must show that it violates the criterion of local soundness defined in class. That is, you must find an environment η and a statement S such that $\eta \mapsto_S \eta'$ (executing S on the memory η yields new memory values η') and $\alpha_{CP}(\eta') \not\sqsubseteq f_{CP}(\sigma, S)$ with $\sigma = \alpha_{CP}(\eta)$.

Question 3.1 (5 points).

Find a pair of η and S that illustrates the local unsoundness. What are η and S?

Question 3.4 (2 points).

What is $\sigma = \alpha_{CP}(\eta)$?

Question 3.3 (3 points).

If $\eta \mapsto_S \eta'$, what is η'?

Question 3.5 (2 points).

What is $\alpha_{CP}(\eta')$?

Question 3.6 (4 points).

What is $\sigma' = f_{CP}(\sigma, S)$?

Question 3.7 (4 points).

Show that $\alpha_{CP}(\eta') \not\sqsubseteq \sigma'$