Assignment 2:
Locks, Lattices, and Pointers

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

Out: Thursday, January 27, 2005
Due: Thursday, February 3, 2003 (11:59 pm)

100 points total

1 Lock Analysis (30 points)

After graduating from CMU, you have been hired by FluidSoft, a (fictional) company devoted to bringing the benefits of the Fluid Java 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 associated with `x`
- `unlock(x)` unlocks the variable associated with `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).

> Draw the lattice for a single variable. Your drawing should clearly show the top and bottom elements of the lattice, as well as symbols for any interior elements of the lattice. The ordering of the lattice should be shown with relative position of the lattice elements and lines between them, as in class.

Now, define your analysis more precisely. Specifically:

Question 1.2 (4 points).

> Is your analysis forward or backward?
>
> What is the initial analysis information for the entry nodes E of the analysis? You should assume that all variables are initially unlocked.

Question 1.3 (10 points).

> Define the transfer functions for your analysis. Because your lattice is not based on sets, you will probably not formulate your transfer functions in terms of gen and kill sets; use any notation you wish as long as it is clear. Don't forget transfer functions for the new `lock(x)` and `unlock(x)` statements.

Question 1.4 (6 points).

> Explain how to interpret your analysis results in terms of deadlock errors (locking a lock twice). Specifically, explain under what precise conditions your analysis detects that the user has definitely tried to lock a lock twice (a double-locking error), and under what conditions the user may have tried to lock a lock twice (a double-locking warning). Your answer should be expressed in general terms, so that it applies to analysis of any program.
>
> There is a corresponding double-unlocking bug, but finding it is not required for this assignment.

2 Pointer Analysis (70 points)

After designing your lock analysis in the WHILE language, you begin to implement it for C. You quickly realize, however, that WHILE isn't a very realistic model of C in one key respect: it doesn't have pointers. Pointers cause trouble for the analysis because when you lock an expression like `*p`, you don't know which variable was locked because you don't know what variables `p` may point to.

Your next task, therefore, is to design a pointer analysis that will tell you, for each variable at each program point, what variables that variable *may* point to. You decide to model pointers by adding two new expressions and one new statement to the WHILE language:

- `*x` is an expression that returns the value in the variable that `x` points to.
- `&x` is an expression that returns the address of the variable `x`.
- `[*x := a]` is a new assignment statement that assigns the value computed by `a` to the variable that `x` points to.

A senior engineer at FluidSoft suggests that your lattice should be the powerset of $\{(x, y) \mid x, y \in FV(S_*)\}$, i.e. you pass around sets of pairs (x, y) meaning that variable x may point to variable y. You first go about determining some basic questions about your analysis.

Question 2.1 (20 points).

What is the ordering operator $\sqsubseteq$ for your lattice?

What is the join operator $\sqcup$ for your lattice?

What is the top element $\top$ of your lattice?

What is the bottom element $\bot$ of your lattice?

Is your analysis forward or backward?

What is the initial analysis information for the entry nodes E of the analysis? Assume all variables are initialized to NULL (i.e., 0) so that they don't point to any other variable.

You discuss your analysis with the senior engineer at FluidSoft, and she gets you started on defining the transfer functions for your analysis. She suggests using gen and kill sets, so that your transfer function for the points-to (PT) analysis will be of the form:

$$PT_{exit}(\ell) = (PT_{entry}(\ell) \setminus kill_{PT}(B^\ell)) \cup gen_{PT}(B^\ell)$$

The senior engineer also gives you a head start on a function PTE that calculates the variables pointed to by a computed expression. Here is the partial definition of PTE:

$PTE(n) = \emptyset$
$PTE(a_1\ op_a\ a_2) = \emptyset$
$PTE(x) = PT_{entry}(\ell)(x)$
$PTE(\&x) = \{x\}$

Question 2.2 (10 points).

Define the remaining case for the PTE function:

$PTE(*x) =$

Question 2.3 (10 points).

Define the gen functions for the two forms of assignment. Hint: you should use the PTE function.

$gen_{PT}(x := a) =$

$gen_{PT}(*x := a) =$

Question 2.4 (10 points).

Now define the kill functions for the two forms of assignment.

$kill_{PT}(x := a) =$

$kill_{PT}(*x := a) =$

Question 2.5 (10 points).

Now, check the correctness of your analysis by simulating it on the following example program using chaotic iteration. Use the format from lecture to show your work:

$[x := \&y]^1;$
$[w := \&z]^2;$
$if\ [c]^3$
$\quad then\ [y := \&x]^4;$
$\quad else\ [y := w]^5;$
$[z := *x]^6;$
$[*y := \&r]^7$

The data flow value at the end of statement 7 after your analysis has reached a fixed point should be:

$\{(w, z), (x, y), (x, r), (y, x), (y, z), (z, x), (z, z), (z, r)\}$

Now, let's look at how pointer analysis can help us do better lock analysis in languages like C. Let's assume that both pointers and locks have been added to WHILE. We've already run a pointer analysis, and we have the results $PT_{enter}(\ell)$ and $PT_{exit}(\ell)$ for all statements in the program.

Question 2.6 (10 points).

Write an additional case for your lock analysis transfer function to handle lock statements of the form $[lock(*x)]^\ell$. You may use $PT_{enter}(\ell)$ and/or $PT_{exit}(\ell)$ if you need to.

Hint: try out your new transfer function on the following program to make sure you got it right.

$[lock(y)]^1;$
$if\ [c]^2$
$\quad then\ [x := \&y]^3;$
$\quad else\ [x := \&z]^4;$
$[lock(*x)]^5;$