Assignment 3
Program Analysis

15-411: Compiler Design
Miguel Silva(`miguels@andrew`)

Due: Thursday, October 8, 2009 (1:30 pm)

Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own.

You may hand in a handwritten solution or a printout of a typeset solution at the beginning of lecture on Thursday, October 8. Please read the late policy for written assignments on the course web page. If you decide not to typeset your answers, make sure the text and pictures are legible and clear.

Problem 1

[20 points]

As discussed in class, constant propagation can be performed by first computing reaching definitions. However, it is also possible to perform constant propagation directly as a forward dataflow analysis. In this problem, we will explore a sort of constant propagation for boolean values, keeping track of some information about the value that a boolean variable x may have. We write T for the value true, F for the value false, and $!\,x$ for the negation of x. We assume variables are typed as containing either booleans or integers.

We write $\mathsf{val}(l, x, V)$ if the value of the boolean variable x at line l in the program may be V, where V is either T or F. This means we don't track the precise value (which would be impossible), but only a sound approximation of the possible values of x at line l. As in Lectures 4 and 5, we mean the value of x *before* the instruction at line l is executed.

The val predicate is seeded at assignments according to the following rules.

$$\frac{l : x \leftarrow T \quad \mathsf{succ}(l, l')}{\mathsf{val}(l', x, T)} \qquad \frac{l : x \leftarrow F \quad \mathsf{succ}(l, l')}{\mathsf{val}(l', x, F)} \qquad \frac{l : x \leftarrow y \odot z \quad \mathsf{succ}(l, l')}{\mathsf{val}(l', x, T) \quad \mathsf{val}(l', x, F)}$$

In the last rule, $\odot$ stands for some binary logical operator such as conjunction or disjunction. Here, we have two conclusions, because we cannot predict the value of x in this simple analysis and so x may be T or F.

a. Write the rule for $l : x \leftarrow !\, y$.

b. Complete the specification of the analysis by writing rules to propagate value information throughout the program. You may assume the language and predicates are as in notes to Lecture 5 on dataflow

analysis with the addition of function calls, except that conditionals now take boolean variables.

$l : x \leftarrow c$	constant load
$l : x \leftarrow y$	move
$l : x \leftarrow y \oplus z$	binary operation
$l : x \leftarrow f(x_1, \ldots, x_n)$	function call
$l :$ goto l'	jump
$l :$ if (x) goto l'	conditional, for boolean x
$l :$ return x	return

You may use predicates $\mathsf{use}(l, x)$, $\mathsf{def}(l, y)$, and $\mathsf{succ}(l, l')$, but not live, nec, or reaches.

c. If x is a function parameter, how do we define val of x at l_0, the beginning of the function body?

d. On the other hand, how do we define val of x at l, if x was not initialized before line l?

We can use the approximate value information for optimization in the following way:

- If $\{V \mid \mathsf{val}(l, x, V)\} = \{b\}$ (where b is either F or T) we can replace uses of x at line x by b, because that is the only value x can have at line l.

Now consider the program (assume g is an unknown function):

l_0 : $z \leftarrow F$
l_1 : $y \leftarrow !\, z$
l_2 : if (y) goto l_4
l_3 : $y \leftarrow z$
l_4 : $u \leftarrow g(y)$
l_5 : return u

e. Compute the possible values V of y and z at each line.

f. Using these values, perform constant propagation

g. After constant propagation, could the program be further simplified?

Problem 2

[20 points]

Consider the following program:

l_0 : $x \leftarrow g()$
l_1 : if (x) goto l_3
l_2 : $x \leftarrow !\, x$
l_3 : return x

a. Is it possible, using constant propagation, to simplify this program?

b. In the previous problem, we were not fully exploiting our knowledge about the values of the variables because the conditional branch remains. More generally, for common tests such as

$$l : \text{if } (x) \text{ goto } l'$$

we should be able to conclude that x is T at l' and x is F at $l + 1$. An analysis that captures this phenomenon is called *path sensitive*.

i. Extend and/or modify your set of rules from Problem 1 so that the analysis is now path sensitive.

ii. Compute the possible approximate values V of x in the program above with your analysis. Could the program be simplified now?

Problem 3

[20 points]

In this question you will write the rules for an analysis that determines the *available expressions* at any program point.

We say that an expression $x \oplus y$ is available at line l if $x \oplus y$ is computed along every path from the start of the program (or function) to line l. On the other hand, if x or y are modified at line l, then $x \oplus y$ is no longer available at any successor of l.

a. Write the rules for this analysis on the language for dataflow analysis from Problem 1. You may use the use, def, and succ predicates, but no others.

b. Calculate the available expressions at each line for the following program:

$$
\begin{array}{lcl}
l_0 & : & x \leftarrow 3 \\
l_1 & : & y \leftarrow 4 \\
l_2 & : & z \leftarrow x + y \\
l_3 & : & w \leftarrow x + y \\
l_4 & : & x \leftarrow 1 \\
l_5 & : & w \leftarrow x + y
\end{array}
$$

c. Describe an optimization that can be performed using the information from this analysis.