# CS 701

## Final Exam

Monday, December 17, 2007

10:00 a.m. — noon

1257 CSST

**Instructions**

Answer question #1 and any three others. (If you answer more, only the first four will count.) Point values are as indicated. Please try to make your answers neat and coherent. Remember, if we can't read it, it's wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

1. (1 point)
   In Madison, on a typical December day, you can expect:
   (a) Snow.
   (b) Sleet.
   (c) Freezing rain.
   (d) All of the above.

2. (a) (10 points)
   Recall that the solution to a data flow problem need not be unique. Give a *simple* example of an instance of a data flow problem that has two or more solutions.
   (b) (10 points)
   Assume that a Control Flow Graph is a DAG or Tree (it has no cycles). Is it possible for a forward data flow analysis performed on this CFG to have a non-unique solution? If so, give a *simple* example. If not, explain carefully why.
   (c) (13 points)
   Let us represent a solution to a forward data flow problem as a vector of the Out values for each block in the CFG being analyzed. That is, solution = $(Out_0,Out_1,...,Out_n)$ where the CFG has blocks numbered 0 to n. We will say data flow solution A is ≥ Data Flow solution B (for the same CFG and the same analysis) if $A_0 \geq B_0$, $A_1 \geq B_1$, ..., $A_n \geq B_n$. That is, for each basic block, A's solution must be ≥ B's solution. Let S be any valid solution to a particular forward data flow problem for a given CFG, and let T be the solution computed by the iterative DFO solution algorithm presented in class. Show that T ≥ S. (That is, the iterative algorithm's solution is always as good as, or better than, any other valid solution.) You may assume, as usual, that all transfer functions are monotone, and that the standard lattice axioms (on ≥ and $\wedge$) hold.

3. In project 3 we investigated removing loop-invariant expressions from loops. Removing loop-invariant assignments is trickier, because of the side-effects of assignments. Consider the following two program fragments.

```
a=0;                          a=0;
b=0;                          b=0;
do                            do
   a=1;                          if (f(b) == 0)
   b++;                             a=1;
while (b<10);                    b++;
                              while (b<10);
```

(a) (5 points)
Explain carefully why moving `a=1` to the loop's preheader is correct in the left program, but is incorrect in the right program.

(b) (14 points)
Explain how to put these two programs into SSA (static single assignment) form. Show each of the two programs after they have been put into SSA form. (Show the programs in source form, with variables rewritten and phi functions added as needed.)

(c) (14 points)
Assuming a program is in SSA form, what rules are sufficient to determine if an assignment can be moved from within a loop to the loop's preheader?

4. This question involves points-to analysis in a language like C or C++. Assume the following pointer manipulation statements appear in a subprogram (points-to analyses are flow-insensitive so non-pointer statements are irrelevant):

```
r  = &p3;
s  = &p2;
p4 = *s;
p1 = &a;
p2 = p1;
p1 = &b;
p3 = &c;
p2 = &d;
*r = p4;
```

(a) (11 points)
Show the points-to graph that Andersen's Algorithm would compute for these statements.

(b) (11 points)
Show the points-to graph that Steengaard's Algorithm would compute for these statements. How does it compare to that of Andersen's Algorithm?

(c) (11 points)
Show the points-to graph that Horwitz's Algorithm would compute for these statements. Assume one run and two categories are used. The category assignments are:
Category 1: `p1`, `p4`, `a`, `b`, `r`
Category 2: `p2`, `p3`, `c`, `d`, `s`
How does this points-to graph compare with those produced in parts (a) and (b)?

5. (a) (17 points)

Some language designers have suggested the following form of loop, which unifies while and do-while loops into a single more general structure:

```
loop
   body1
   exit when (pred);
   body2;
end;
```

In this loop, the termination condition is in the "middle" of the loop. $body_1$ is executed. Then pred is tested. If it is true, the loop is terminated. Otherwise, $body_2$, followed by $body_1$, are executed, and pred is tested again. Iteration continues until pred becomes true.

Explain how to compute the transfer function of this form of loop given the transfer functions of $body_1$, $body_2$, and pred.

(b) (16 points)

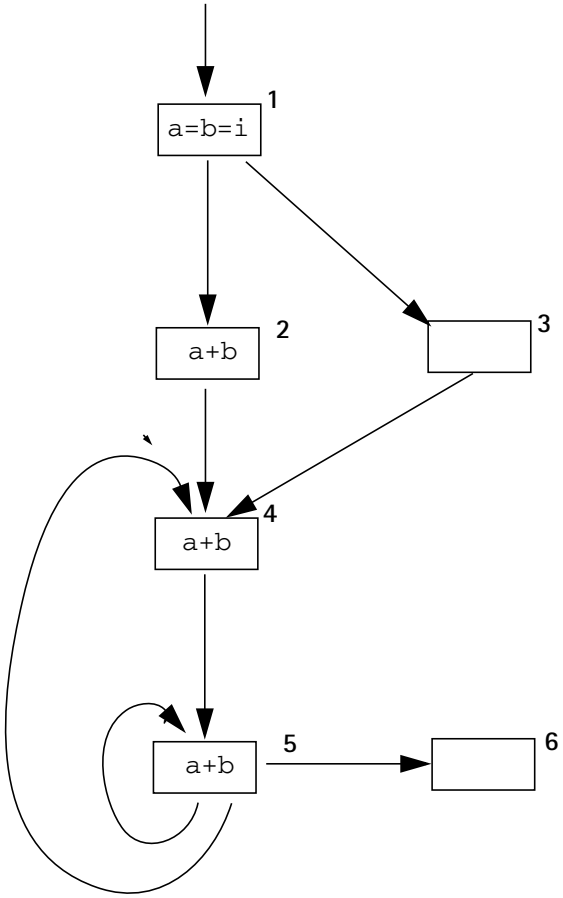
A sequence of if-then-else statements can be combined into a single if-then-elsif statement:

```
if (pred1)
   body1;
 elsif (pred2)
   body2;
 ...
 elsif (predn)
   bodyn;
 else
   bodyn+1;
```

Explain how to compute the transfer function of this form of if statement given the transfer functions of $body_1$, $body_2$,..., $body_{n+1}$, $pred_1$,..., $pred_n$.

6. This question involves partial redundancy elimination. The data flow equations that define partial redundancy are listed at the end of this question.

   Consider the following control flow graph. What are the PPIn and PPOut values for each block? Where should computations of `a+b` be added and where should existing computations of `a+b` be deleted? Explain why your solutions are correct.




$PPOut_b$ = 0 for all exit blocks

$= \underset{k \in succ(b)}{AND} PPIn_k$

$Const_b = AntIn_b$ AND [$PavIn_b$ OR ($Transp_b$ and ¬ $AntLoc_b$)]

$PPIn_b$ = 0 for $b_0$ (the start block)

$= Const_b$ AND ($AntLoc_b$ or ($Transp_b$ AND $PPOut_b$)

$\underset{p \in pred(b)}{AND}$ ($PPOut_p$ OR $AvOut_p$)

$Insert_b = PPOut_b$ AND ( ¬ $AvOut_b$) AND ( ¬ $PPIn_b$ OR ¬ $Transp_b$)

$Remove_b = AntLoc_b$ AND $PPIn_b$

**Partial Redundancy Equations**