CS 701
Final Exam

Thursday, December 11, 2003

11:00 a.m. — 1:00 p.m.

2321 Engineering Hall

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 a microprocessor, an I-cache holds:
   (a) Ink.
   (b) Ice Cream.
   (c) Irish coffee.
   (d) Instructions.

2. (a) (5 points)
   In the following loop, live variable analysis would determine the assignment to `x` in the loop body to be live, even though `x` is not used outside the loop. Explain why.

```
while (...) {
   x = x+1;
}
x = 100;
```

   (b) (28 points)
   A variable that is not live is considered dead. A concept related to dead variables is that of a *faint* variable. At any point faint variables are a superset of dead variables. A variable `x` at a particular point is faint if it is dead at that point or if it a live only because it is used in an assignment to a faint variable. A variable used in a control predicate or a print statement is never faint at the point at which it is used.
   Call a variable that is not faint *truly live*. True-liveness analysis identifies the minimal set of truly-live variables at each basic block. Give a data flow framework for determining true-liveness, That is, give the solution lattice, direction, transfer functions and meet operation necessary to compute `trulyLiveIN(n)` and `trulyLiveOut(n)` for `n`∈`basicBlocks`. Illustrate your truly live analysis on the code fragment of part (a).

3. (a) (23 points)

   A key step in putting a program in SSA (static single assignment) form is placement of the ϕ functions. Assume that all assignments to variables have been rewritten so that each variable is now assigned to exactly once. Give an algorithm to determine, for a set of variables $x_0, \ldots, x_n$ derived from a single variable `x`, where to place ϕ functions and the appropriate arguments for each ϕ. How are the arguments (the x_i values) of each ϕ determined?

   (b) (10 points)

   Show that once a program is in SSA form, reaching definition analysis is greatly simplified. In particular no analyses beyond those needed to put a program in SSA form are needed to determine which definitions to a variable reach a given use of that variable.

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

   (a) (10 points)

   Assume that expression e is computed and used in block b. However, e is neither computed nor used again on any path from b to an exit node. Show that partial redundancy elimination will never add a computation of e into any block from b to an exit node (including b).

   (b) (23 points)

   Assume that expression e is computed and used in block b. However, e is neither computed nor used on any path from b_0 to b. Show that partial redundancy elimination will never add a computation of e into any block from b_0 to b (excluding b).

$$\mathrm{PPOut}_b = 0 \text{ for all exit blocks}$$
$$= \underset{k \in \mathrm{succ}(b)}{\mathrm{AND}} \mathrm{PPIn}_k$$

$$\mathrm{Const}_b = \mathrm{AntIn}_b \text{ AND } [\mathrm{PavIn}_b \text{ OR } (\mathrm{Transp}_b \text{ and } \neg \mathrm{AntLoc}_b)]$$

$$\mathrm{PPIn}_b = 0 \text{ for } b_0 \text{ (the start block)}$$
$$= \mathrm{Const}_b \text{ AND } (\mathrm{AntLoc}_b \text{ or } (\mathrm{Transp}_b \text{ AND } \mathrm{PPOut}_b) \underset{p \in \mathrm{pred}(b)}{\mathrm{AND}} (\mathrm{PPOut}_p \text{ OR } \mathrm{AvOut}_p)$$

$$\mathrm{Insert}_b = \mathrm{PPOut}_b \text{ AND } (\neg \mathrm{AvOut}_b) \text{ AND } (\neg \mathrm{PPIn}_b \text{ OR } \neg \mathrm{Transp}_b)$$

$$\mathrm{Remove}_b = \mathrm{AntLoc}_b \text{ AND } \mathrm{PPIn}_b$$

Partial Redundancy Equations

5. The languages C and C++ allow function pointers. A function pointer points to a function rather than an area of memory. It is used to indirectly access and call a function. Function parameters may be simulated using function pointers. For example, the declaration

   ```
   int (*fp)();
   ```

   declares that `fp` may point to any integer valued function of no arguments.
   It is often the case that an optimization requires an accurate estimate of what functions a function pointer might access. That is, if we see a call

   ```
   (*fp)();
   ```

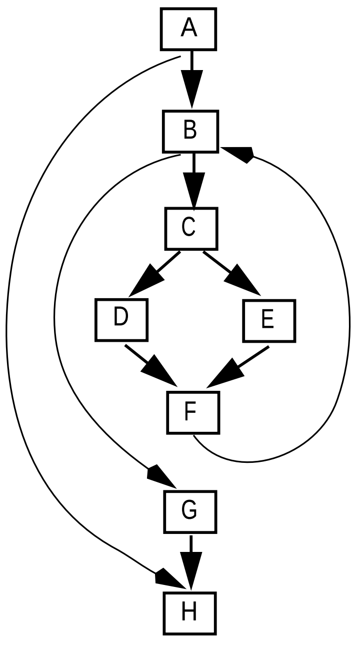
   we want to know what functions might actually be activated.
   We will focus on two kinds of assignments: `fp` = `function;` and `fp1` = `fp2`;
   The first assignment binds `fp` to point to a particular known function, while the second kind of assignment copies the function pointed to by `fp2` into `fp1`.
   To simplify our analysis, we will assume only one assignment to a pointer variable occurs in each basic block (blocks can be split, as needed, to guarantee this).
   We wish to formulate a data flow problem that will determine, for a given function pointer, the set of functions it might access in each basic block.

   (a) (13 points)
   Give a data flow framework (solution lattice, direction, transfer function and meet operation) that can be used to solve the "points to" problem described above.

   (b) (10 points)
   Is the data flow problem you formulated distributive? If it is, explain carefully why. If it is not, give a *simple* counter example.

   (c) (10 points)
   Is the data flow problem you formulated rapid? If it is, explain carefully why. If it is not, give a *simple* counter example.

6. This question involves a variety of trees and graphs used in data flow analysis and optimization. You will use the following control flow graph to illustrate the answers to parts (a) to (c):



(a) (11 points)
Explain how a DFST (depth-first spanning tree) is computed for a control flow graph. Illustrate your solution on the above control flow graph.

(b) (11 points)
Explain how a postdominator tree is computed for a control flow graph. Illustrate your solution on the above control flow graph.

(c) (11 points)
Explain how a control dependence graph is computed for a control flow graph. Illustrate your solution on the above control flow graph.