CS 701

Final Exam

Monday, December 20, 1999

10:00 a.m. — 12:00 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)
   What are the **top**-most and **bottom**-most values in a lattice called?

2. Recall that by using constant propagation we can determine that at a particular point in a program a variable v must contain a constant value c. Let us say that at a particular point in a program a variable is **N-limited** if we know that it must contain one of N constant values, c_1, c_2,..., c_n.
   Note that in doing constant propagation, we establish that a variable is 1-limited (limited to one possible value). Boolean variables are naturally 2-limited, since they must always be either true or false.
   (a) (10 points)
   Give examples that illustrate how knowing that a particular variable is N-limited can be used to optimize a program.
   (b) (23 points)
   We can compute N-limited variables by generalizing constant propagation (tracking sets of N values rather than a single value).
   Give a data flow framework for determining N-limited variables for a given N. That is, give the solution lattice, direction, transfer functions and meet operation necessary to compute NLimited(b), where b is a basic block and NLimited(b) is a function that maps a variable name to the set of N values the variable is limited to, or an indication that the variable is not limited to N values.

3. Assume we have a Java-style try statement of the form

```
try
    body
catch
    handler;
```

In this try statement `body` is executed. If any statement in `body` executes a **throw** statement, execution transfers immediately to the statements in `handler`. If no throw is executed within `body`, then the statements in `handler` are ignored. Thus either all of `body` is executed or a prefix of `body` is executed, followed by the statements in `handler`.

(a) (11 points)
Assume we limit our try statement to the case in which `body` contains only one top-level throw controlled by a predicate. That is, the try is now of the form

```
try
   body1
   if (pred)
           throw;
   body2;
catch
   handler;
```

Explain how to compute the transfer function of this form of try statement given the transfer functions of $\texttt{body}_1$, $\texttt{body}_2$, `pred` and `handler`.

(b) (11 points)
Extend your solution to (a) to the case in which body contains more than one top-level throw, each controlled by its own predicate. That is, the try is now of the form

```
try
   body1
   if (pred1)
           throw;
   body2;
   if (pred2)
           throw;
   ...
   if (predn)
           throw;
   bodyn+1;
catch
   handler;
```

Explain how to compute the transfer function of this form of try statement given the transfer functions of $\texttt{body}_1$, $\texttt{body}_2$,..., $\texttt{body}_{n+1}$, $\texttt{pred}_1$,..., $\texttt{pred}_n$ and `handler`.

(c) (11 points)

Extend your solution to (a) to the case in which a try statement contains one top-level throw, controlled by a predicate, nested within a top-level do-while loop. That is, the try is now of the form

```
try
   body1
   do
         body2
         if (pred1)
              throw;
         body3;
   while (pred2)
   body4;
catch
   handler;
```

Explain how to compute the transfer function of this form of try statement given the transfer functions of $\texttt{body}_1$,..., $\texttt{body}_4$, $\texttt{pred}_1$, $\texttt{pred}_2$ and `handler`.

4. (a) (11 points)

We showed in class that if a value lattice has a finite height, then our iterative algorithm for solving data flow equations must terminate with a solution (since on each iteration one or more In_b or Out_b values must change, and each value can change only a bounded number of times because the lattice has finite height).

Recall that some lattices that we used had finite height but infinite width (e.g., the lattice used for constant propagation). If we have such a lattice, explain why our iterative algorithm can't loop forever moving "sideways" across the lattice. For example, given the statement `a=a+1` in a loop body, why can't our algorithm, when doing constant propagation, start with value v for `a`, then try v+1, then v+2, etc., never terminating?

(b) (11 points)

Show that if we use a lattice that has infinite height, then our iterative data flow algorithm can fail to halt. Can we sidestep this problem by using a structured solution in which a transfer function for a whole program is created by composing transfer functions for each expression and statement in the program?

(c) (11 points)

Recall that DFO order (depth-first order) is recommended in solving data flow equations using an iterative algorithm. Why is DFO order better than an arbitrary visit order for nodes within the control flow graph? Is DFO order always best, or is a non-DFO order sometimes preferable (in reducing the number of iterations needed to solve a set of data flow equations)?

5. This question involves partial redundancy elimination. The data flow equations that define partial redundancy are listed at the end of the question.
   (a) (11 points)
   Assume an expression, e, has been placed at the end of block b using the Insert_b rule defined below. If b has one or more immediate successors that neither use nor kill e, under what circumstances can the computation of e be sunk to each of b's successors?
   (b) (11 points)
   Assume an expression, e, is invariant in a do-while loop, but appears within a conditional (an if or switch statement) within the loop. Under what circumstances will the partial redundancy equations move the computation of e outside the loop?
   (c) (11 points)
   Assume that b_1 is a ancestor (either direct or indirect) of b_2. Is it ever possible for the Insert_b rule of the partial redundancy equations to specify that the same expression is to be placed in both bocks b_1 and b_2? If so, give a simple example. If not, explain why this can't happen.

$\text{PPOut}_b = 0$ for all exit blocks

$$= \mathop{\text{AND}}_{k \in \text{succ}(b)} \text{PPIn}_k$$

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

$\text{PPIn}_b = 0$ for b_0 (the start block)

$$= \text{Const}_b \text{ AND } (\text{AntLoc}_b \text{ or } (\text{Transp}_b \text{ AND } \text{PPOut}_b) \mathop{\text{AND}}_{p \in \text{pred}(b)} (\text{PPOut}_p \text{ OR } \text{AvOut}_p)$$

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

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

Partial Redundancy Equations