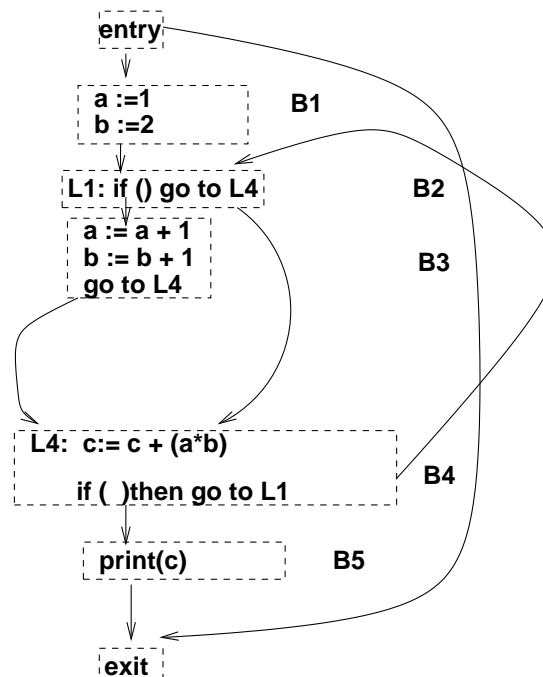


CSE 231 Fall 2003 HW 2 Due Mon Oct. 20

Please check class web page for homework policy.

Problems:

1. Consider the following CFG:



- (a) From the CFG, construct the Dominance Frontier set for every basic block node.
 - (b) Construct the SSA form for the program, and indicate the Def-use chains for this.
 - (c) Construct the Control Dependence graph for the CFG.
2. Consider the following program fragment with references to array A:

```
<s0>      do i = 1, 100
<s1>          A[2*i] := ...
<s2>          ..... := A[4*i - 2] .....
<s3>          A[3*i - 5] := ...
<s4>      enddo
```

What data dependences hold between the references according to the GCD test? Classify the data dependences as *flow* or *storage-related*. Also indicate which dependences are *loop-carried*.

3. *Extra Credit: Recurrent Pointer Update Problem:* This question is optional.

This problem asks you to help define a new dataflow problem and to find the dataflow equations needed to solve it. Suppose that we want to implement some optimization that works on linked lists. (For example, the optimization might make an effort to store the nodes sequentially in memory to improve cache performance.) Our goal is to find instances in the program where there is a loop that performs "recurrent updates" to a pointer, which we will use to trigger the optimization. I hope the following examples will make the motivation clear.

This first example shows a loop that we want to say involves a recurrent update to p.

```
typedef struct node *link;
struct node { int data; link next; };
link p;
...
for (p = firstNode; p!=NULL; p = p->next) {
    ... loop body with no assignments to p ...
}
```

The second example is a tree search, which is NOT a recurrent update.

```
typedef struct node* link;
struct node { int data; link left, right; };
link p;
...
p = treeRoot;
for (i = 0; i<N; i++) {
    if (x<p->data) p = p->left;
    else p = p->right;
}
```

The third example is a more complicated case, which (I think) could qualify as a loop that performs recurrent updates. However, it might be too complicated to recognize because of the conditional updates (which may or may not advance the pointer), the use of temp (whose value might come from outside or inside the loop), and the multiple assignments to the pointer x.

```
typedef struct node* link;
struct node { int data; link left, right; };
link p;
...
p = treeRoot->right;
```

```

temp = treeRoot->right;
/* Loop invariant: temp will always be either p or p->left */
for (i = 0; i<N; i++) {
    if (x<p->data) p = temp;
    else temp = p->left;
    if (random()>.5) p = temp;
}

```

Your goal is to devise the dataflow sets and equations needed to recognize examples of recurrent updates. It need not necessarily find ALL recurrent updates (for instance, it might miss the third), but you should briefly describe what cases your algorithm will catch and what it will miss.

IMPORTANT: Your dataflow analysis should only err on the side of not finding recurrent updates (i.e., it should never mistakenly say a loop recurrently updates a pointer when in fact it doesn't), since triggering the optimization on a non-example might introduce a bug. For instance, this could happen if the optimization converted the linked list to an array data structure.

- (a) Define the sets *Gen*, *Kill*, *RPUIn*, and *RPUOut* needed to solve this problem. To define the *Gen* and *Kill* sets, you need to consider the different ways a pointer can be assigned in a basic block.
- (b) Give the data flow equations to compute the *RPUIn* and *RPUOut* sets.