Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.035, Fall 2005 Handout 10 — Practice Quiz 2 Friday, November 11

---

1. **Name That Optimization**

   For the basic block:

```
a = 1
b = 2
c = 3
d = a + x
e = b + c
f = e
g = f
g = d + y
a = b + c
```

   State for each of the basic blocks on the following page which optimization was performed on the above:

   - Constant Propagation/Folding
   - Copy Propagation
   - Common Subexpression Elimination
   - Dead Code Elimination.

(a)
```
a = 1
b = 2
c = 3
d = a + x
e = b + c
f = e
g = d + y
a = b + c
```

(b)
```
a = 1
b = 2
c = 3
d = a + x
e = b + c
t1 = e
f = e
g = f
g = d + y
a = t1
```

(c)
```
a = 1
b = 2
c = 3
d = 1 + x
e = 5
f = 5
g = 5
g = d + y
a = 5
```

(d)
```
a = 1
b = 2
c = 3
d = a + x
e = b + c
f = e
g = e
g = d + y
a = b + c
```

2. **Dead Variable Analysis**

Dead Variable Analysis determines, for each program point, the set of variables whose values are no longer needed by the computation enclosed within a given control flow graph. A variable is dead if there are no further uses of the variable before it is reassigned a new value. You may assume all variables are local and therefore dead at exit from the CFG.

(a) Is Dead Variable Analysis a forward or backward dataflow problem?

(b) What do the members of the IN and OUT sets represent? What does it mean for a set member to be a zero or one?

(c) What are the initialization conditions? (The initial values of IN or OUT where appropriate)

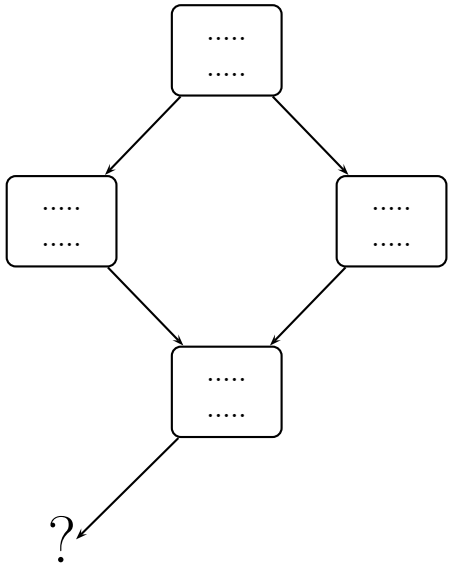
(d) What is the confluence operator?

(e) What are the GEN and KILL sets for the basic blocks in Figure 1?

	GEN	KILL
B1		
B2		
B3		
B4		
B5		
B6		

3. **Fixed Points**

You are on a deadline. The first release of FrobozzCo. Inc's compiler, FlatheadCC, is due out next week. Fortunately for him, Ben Bitdiddle somehow managed to get this week off (the Flatheads never were very good managers) and is sunning himself in Hawaii. Unfortunately for you, his code to build the control flow graph for a procedure is having some problems. Occasionally, you discover an edge which is dangling; you know where the edge comes from but you can't tell which basic block it is supposed to point to.



How would you adapt the following procedure for Reaching Definition analysis to compensate for this problem? You may assume that only one such edge exists in any particular CFG. Note that the following procedure takes the node from which the edge is dangling as a parameter. This parameter is called `dangle`. You may also assume that a preprocessing stage has modified the `predecessors` and `successors` functions to ignore the dangling edge.

```
// N is the set of nodes in the control flow graph
// E is the set of edges in the control flow graph
// Entry is the entry node of the control flow graph
// Exit is the exit node of the control flow graph
// D is the set of definitions in the control flow graph
// gen[n] is the set of definitions that the node n generates
// kill[n] is the set of definitions that the node n kills
// predecessors[n] is the set of predecessors of the node n in the
//   control flow graph
// successors[n] is the set of successors of the node n in the
//   control flow graph

// dangle is the node with dangling outgoing edge

ReachingDefinitions(N,E,Entry,Exit,D,gen,kill,predecessors,successors,dangle)
  // in[n] is the set of definitions that reach the point before
  //   the execution of the node n
  // out[n] is the set of definitions that reach the point after
  //   the execution of the node n
  in[Entry] = emptyset;
  out[Entry] = gen[Entry];
  for all nodes n in N - { Entry }
    out[n] = emptyset; // Can also set out[n] = gen[n];
  Changed = N - { Entry };
  while (Changed != emptyset)
    choose a node n in Changed;
    Changed = Changed - { n };
    in[n] = emptyset;
    for all nodes p in predecessors(n)
      in[n] = in[n] U out[p];
    oldout = out[n];
    out[n] = gen[n] U (in[n] - kill[n]);
    if (oldout != out[n])
      for all nodes s in successors(n)
        Changed = Changed U { s };
```

4. **Global Dataflow and Code Hoisting**

An expression e is said to be *very busy* at a program point p if no matter what path is taken from p, the expression e will be evaluated before any of its operands are defined. Furthermore:

- It is a reverse dataflow analysis.
- The members of the bit sets represent the possibly busy expressions.
- The OUT sets should be initialized to all zeroes.
- The confluence/meet operator is intersection.
- The GEN and KILL sets for Figure 1 are as follows if you assign the expressions to bits in the following order: $(a + b)$, $(c - a)$, $(b * d)$, $(e + 1)$, and $(a - d)$.

	GEN	KILL
B1	00000	11101
B2	10000	01101
B3	00100	00101
B4	10010	00111
B5	11000	10110
B6	00100	11101

Perform global very busy expression analysis on Figure 1. What are the final values of IN for each block once the analysis is complete?

	IN
B1	
B2	
B3	
B4	
B5	
B6	

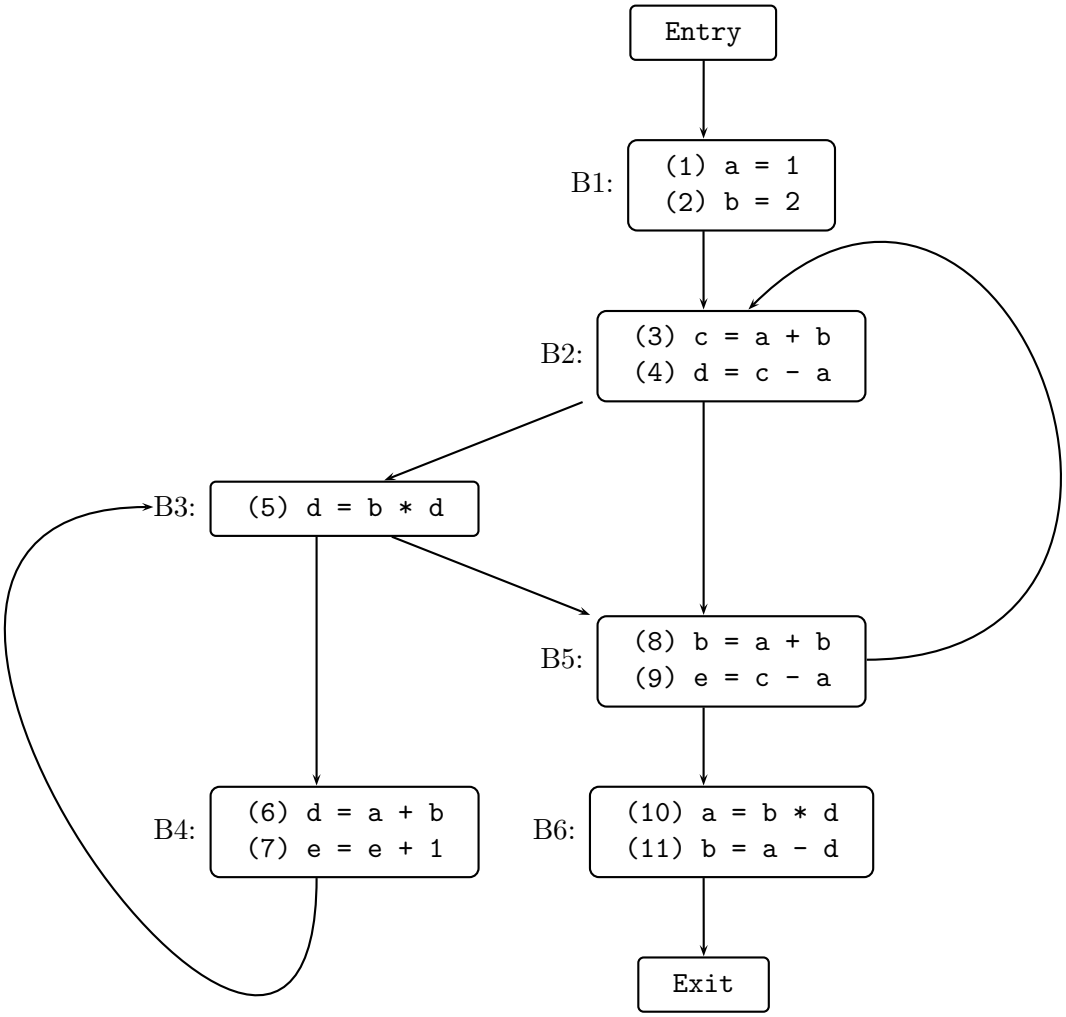


Figure 1: A procedure CFG