Problem Set 6

Fall 22

Due: Sunday, Dec 4th.

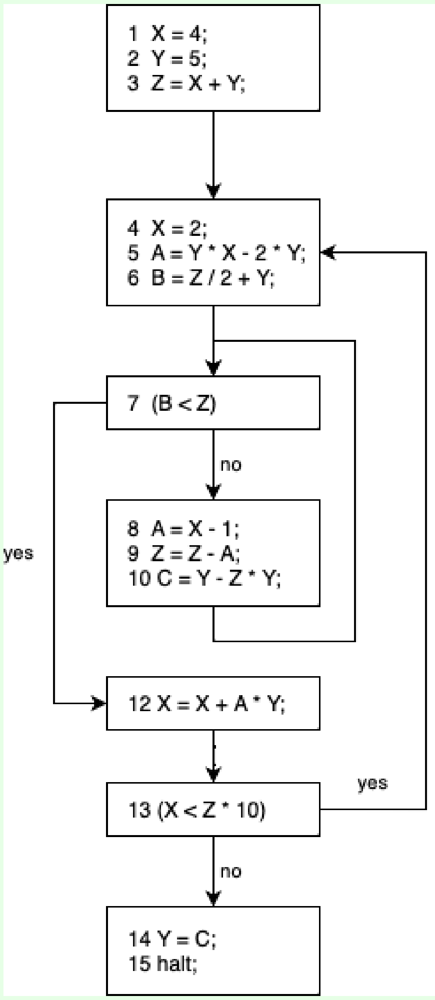
1. Loop Optimization

   For the following problems, consider the code below:

```
1   X = 4;
2   Y = 5;
3   Z = X + Y;
4   X = 2;
5   A = Y * X − 2 * Y;
6   B = Z / 2 + Y;
7   if (B < Z) goto 12
8   A = X − 1;
9   Z = Z − A;
10  C = Y − Z * Y;
11  goto 7;
12  X = X + A * Y;
13  if (X < Z * 10) goto 4;
14  Y = C;
15  halt ;
```

   (a) Draw the CFG for the code above. Identify the loops in the code.

Solution:



The loops are lines 4–13 and lines 7–11.

(b) Which statements are loop invariants? Can they be moved outside their enclosing loop? Show the code that results after hoisting any loop invariant code outside the loop.

Solution:

Statements 4, 5, and 8 are loop invariants. Statements 4 and 5 are loop invariants of the outer loop, and cannot be hoisted outside of their loop because X and A are defined more than once within the loop. Statement 8 is a loop invariant of the inner loop. It cannot be hoisted outside of its loop because it does not dominate all exits of the inner loop (The 'B<Z' check can succeed the first time, causing it to never be reached).

The resulting code is:

```
1   X = 4;
```

```
2   Y = 5;
3   Z = X + Y;
4   X = 2;
5   A = 0;
6   B = Z / 2 + Y;
7   if (B < Z) goto 12
8   A = 1;
9   Z = Z − A;
10  C = Y − Z * Y;
11  goto 7;
12  X = X + A * Y;
13  if (X < Z * 10) goto 4;
14  Y = C;
15  halt;
```

(c) Identify the induction variables in this code. Show the code that results after performing any possible strength reduction.

Solution:

There is one basic induction variable: Z on line 9, because A is loop invariant. However, there is no way to perform strength reduction on Z, as it is already in its simplest form. C is a mutual induction variable, as it relies on the value of Z, which is a basic induction variable. It is possible to perform strength reduction on C.

The rewritten code looks like:

```
1   X = 4;
2   Y = 5;
3   Z = X + Y;
4   CC = Y − Z * Y;
5   X = 2;
6   A = 0;
7   B = Z / 2 + Y;
8   if (B < Z) goto 14
9   A = 1;
10  Z = Z − A;
11  CC = CC + 5;
12  C = CC;
13  goto 8;
14  X = X + A * Y;
15  if (X < Z * 10) goto 5;
16  Y = C;
17  halt;
```

Solution:

There is no opportunity for linear test replacement in this code. The loop conditions are
if (B < Z) goto 14 and
if (X < Z * 10) goto 5.
Both conditions depend on variables modified within the loop they control, and no simpler linear condition that preserves the loop's semantics is possible.

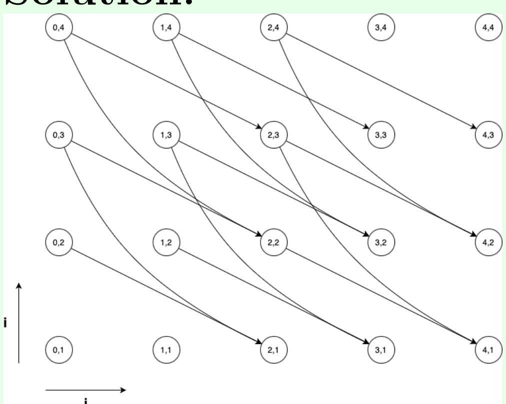
2. Dependence Analysis

For the following problems, consider the code below:

```
for (j = 0; j < 5; j++) {
    for (i = 1; i < 5; i++)
        A[i-1][j+1] = A[i+1][j-1] + A[i][j-1];
}
```

(a) Draw the iteration space graph for the following piece of code (be careful about the index expressions and the loop order!).

Solution:



(b) What are the distance vectors? The direction vectors?

Solution:

Distance vectors: $(2, -1)$ and $(2, -2)$. Direction vectors: $(+, -)$ and $(+, -)$.

(c) Can the loops be interchanged? Why or why not?

Solution:

No, the loops can't be interchanged because the resulting dependency vectors will be negative, signalling that both dependencies will be violated.

(d) Can the following two loops be fused? Why or why not? Explain your answer in terms of dependencies between the loops.

```
for (i = 1; i < 10; i++)
    A[i+1] = B[i+1];
for (i = 2; i < 10; i++)
    B[i-2] = A[i+1];
```

Solution:

The two loop *can* be fused. The data dependencies exist between the i-th iteration of the first loop and the i-th iteration of the second loop (flow dependence), for any $1 < i < 10$, and between the i-th iteration of the first loop and the $i+3$-th iteration of the second loop (anti-dependence), for any $2 < i < 10$. If we fuse the two loops, the flow dependencies will be preserved.

3. Pointer Analysis

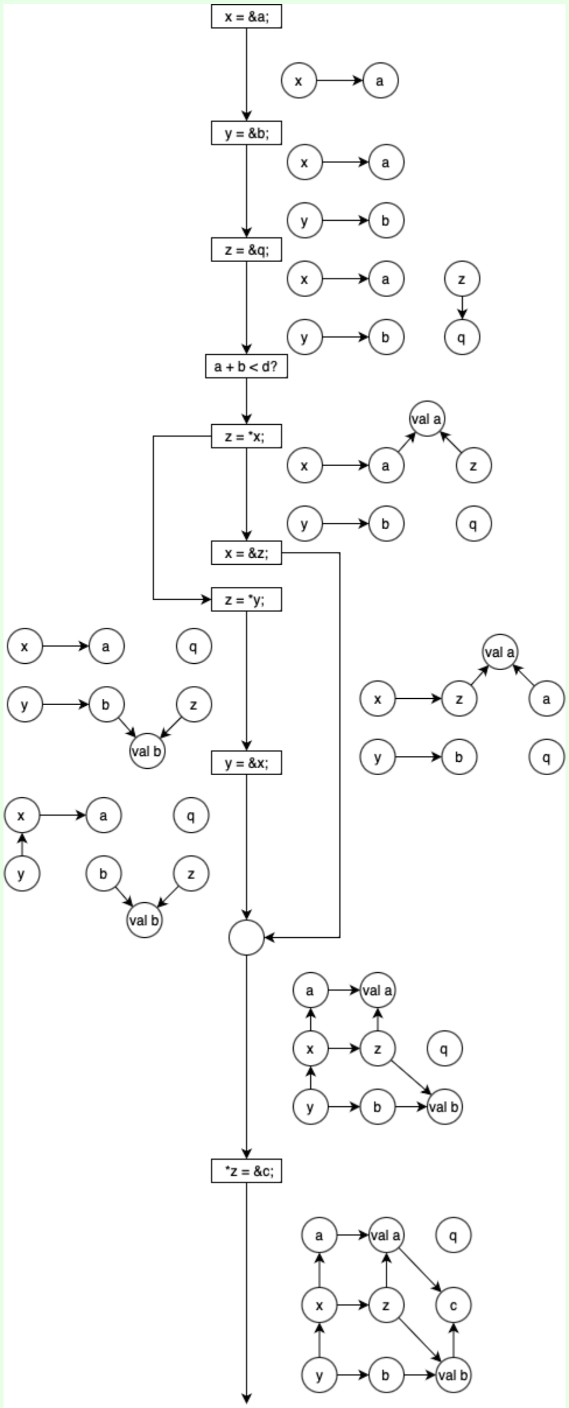
For the following problems, consider the code below:

```
x = &a;
y = &b;
z = &q;
if (a + b < d)
    z = *x;
    x = &z;
else
    z = *y;
    y = &x;
*z = &c;
```

(a) Draw the points-to graph at the end of this piece of code for a flow-sensitive pointer analysis (assume the variables have all been declared appropriately beforehand).

Solution:

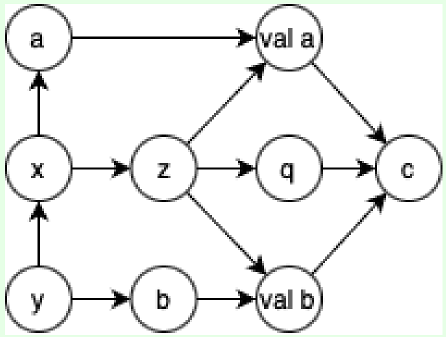
The entire flow-sensitive analysis result is shown below:



(b) Draw the points-to graph you would get if you ran a flow-insensitive pointer analysis on the same code.

Solution:

The flow-insensitive analysis result is shown below:


a
val a
x
z
q
c
y
b
val b