Introduction to Compilers: Midterm 3
December 10, 2020

Name: _____

Purdue Email: _____

Please sign the following:

I affirm that the answers given on this test are mine and mine alone. I did not receive help from any person or material (other than those explicitly allowed).

X _____

Part 1:	/15
Part 2:	/30
Part 3:	/20
Part 4:	/20
Total:	/85

Part 1: Dataflow analysis (15 pts)

Suppose you want to write a dataflow analysis that computes the *ranges* of variables. You can represent the value of a variable by the minimum and maximum value that the variable can have at a particular program point. Thus, a variable x might be represented by the range $[a, b]$, meaning that $a \leq x \leq b$.

Problem 1 (5 points): For the following two statements, assume that x has the range $[a, b]$ immediately before the statement, and y has the range $[c, d]$ immediately before the statement. Give the range of x immediately *after* the statement.

(2 points) `x = x + 2;`

(3 points) `x = x - y;`

Problem 2 (5 points): At a merge point, x has the range $[a, b]$ coming in on one branch, and $[c, d]$ coming in on the other. What should the range of x be *immediately after the merge*?

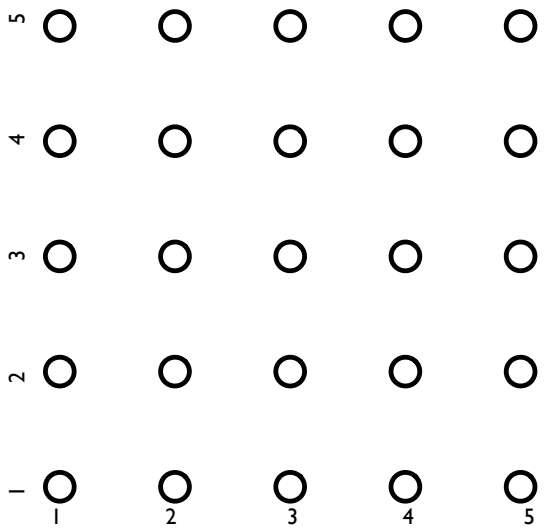
Problem 3 (5 points): Suppose you are analyzing a language that uses BigInts (integer representations that do not have a maximum or minimum expressible value). Is your analysis guaranteed to terminate? Why or why not?

Part 2: Dependence analysis and high-level loop optimization (30 pts)

For the next three problems, consider the following loop:

```
for (int i = 1; i < 6; i++) {
  for (int j = 1; j < 6; j++) {
     A[i+1][j+2] = A[i+2][j-1] + A[i+4][j-1];
  }
}
```

Problem 1 (9 pts): Below is the iteration space graph for the loop nest (i along the horizontal axis, j along the vertical axis). Draw *all of* the dependence arrows that arise from analyzing the loop. Use solid arrows for flow dependences, arrows with a slash through them (as in class) for anti-dependences, and arrows with a circle over them for output dependences.



Problem 2 (2 pts): List the distance vectors that arise from this loop nest, and mark whether they are flow, anti or output.

Problem 3 (2 pt): List the direction vectors that arise from this loop nest, and mark whether they are flow, anti or output.

Problem 4 (5 pts): How would your answers to questions 2 and 3 be different if the code was as follows:

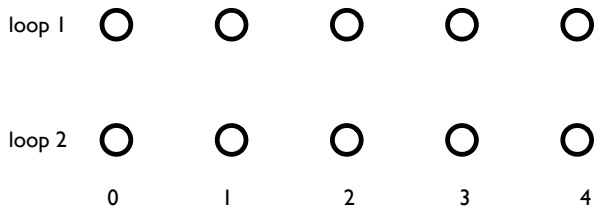
```
for (int i = 1; i < 6; i++) {
  for (int j = 1; j < 6; j++) {
     if (i != 3)
       A[i+1][j+2] = A[i+2][j-1] + A[i+4][j-1];
  }
}
```

For the next two problems, consider the following loops:

```
for (int i = 0; i < 5; i++) {
  A[i+3] = B[i+1];
}

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

Problem 5 (10 pts): Given the iteration space for the two loops, draw the dependences that exist between various iterations using the same kinds of arrows as in Problem 1.



Problem 6 (2 pts): Can the loops be fused? Why or why not?

Part 3: Low-level loop optimization (20 pts)

Problem 1 (15 pts): Show the effects of performing strength reduction on the following loop (for partial credit, identify any induction variables or mutual induction variables)

```
1.  READ(a); //A = read from stdin
2.  i = 0;
3.  j = 0;
4.  b = 0;
5.  c = 0;
6.  if (i > a) goto 12
7.    b = i * 10;
8.    c = j * 15 + 5;
9.    i = i + 1;
10.   j = j + 3;
11.   goto 6;
12. WRITE(b + c);
```

Problem 2 (5 pts): How would line 6 (from the original program) change if you applied linear test replacement?

Part 4: Pointer Analysis (20 pts)

Problem 1 (10 points): Draw the points-to graph you get at the *end* of analyzing the following code:

```
a = read(); //reads input from user
b = read(); //reads input from user
c = read(); //reads input from user
x = & a;
y = & b;
w = & a;

if (a < b) {
  z = & y;
} else {
  z = & x;
}

w = & c;

* z = tmp;
```

Problem 2 (10 points): Draw the points-to graph you would get for the code using a *flow-insensitive* analysis.