Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.035 Fall 2017

Test II Solutions

I Movable Expressions

For some reason, you decide that you want to place the evaluation of as many expressions as you can in the first (start) block of your control flow graph. To that end, you decide to develop a *movable expressions* analysis that computes, for each expression in the program, whether it can be moved into the first block of the control flow graph. Note that the compiler can only put an expression in the first block if it will always be evaluated in every program execution.

You therefore define a movable expression as an expression that 1) is evaluated on all paths from the first block to an exit block of the control flow graph and 2) has none of the operands of the expression redefined between the end of the first block and the evaluation of the expression.

Inspired by your experience in 6.035, you decide to define a backward GEN/KILL dataflow analysis to detect movable expressions. You recall that, in lectures, you heard about four sets for each basic block b – GEN[b], KILL[b], IN[b], and OUT[b]. You decide to use these sets for your analysis. Here GEN[b] is the set of movable expressions generated by the block b, KILL[b] is the set of movable expressions killed by the block b, IN[b] is the set of movable expressions at the start of block b, and OUT[b] is the set of movable expressions at the end of block b.

You decide to use a bit-vector representation of the sets, so that each set is represented by a bit vector, with a position in the bit vector for each expression in the program.

1. [2 points]: What is the least upper bound $\vee$ for two bit vectors e_1 and e_2?

Solution:

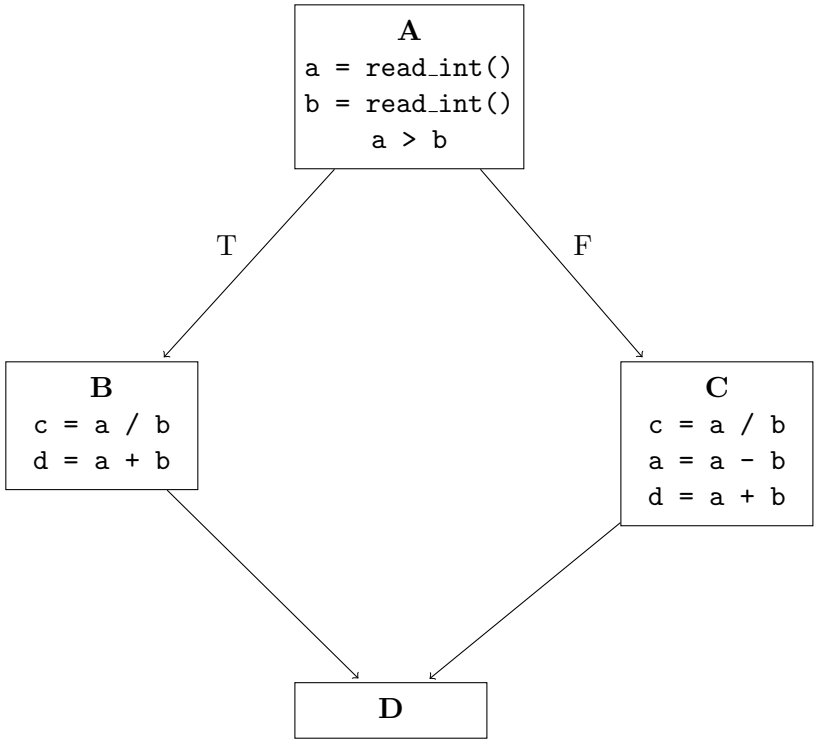
$$e_1 \vee e_2 \iff e_1 \& e_2$$

In the following code each statement is labeled with a number. Assume all variables have been declared.

```
1:  a = read_int();
2:  b = read_int();

3:  if (a > b) {
4:    c = a / b;
5:    d = a + b;
    } else {
6:    c = a / b;
7:    a = a - b;
8:    d = a + b;
    }
```

2. [4 points]: Draw the control flow graph (CFG) for this program. Label each basic block with a letter.



3. [16 points]: Present the analysis result that the movable expressions analysis generates. Use the bit-vector representation, representing {`a/b,a-b,a+b`} as the first, second, and third bit respectively.

A. [8 points]: Compute GEN[b] and KILL[b] for each basic block b. Fill in your bit vectors in the table below, each row for a block b. (If the definitions for GEN and KILL permit multiple answers, write down any such answer that ensures correct analysis for IN and OUT.)

b	GEN[b]	KILL[b]
A	000	111
B	101	000
C	110	111
D	000	000

B. [8 points]: Compute the solution to the dataflow analysis problem, specifically, IN[b] = ... and OUT[b] = ... for each basic block b. Fill in your bit vectors in the table below, each row for a block b.

b	IN[b]	OUT[b]
A	000	100
B	101	000
C	110	000
D	000	000

4. [4 points]: After performing the analysis, what expressions can be moved to the first block in the control flow graph?

`a / b`

II Loop Optimizations

In the following program, `j` is a derived induction variable in the family of the base induction variable `k`.

```
j = 0;
k = 0;
sum = 0;
while (j < 512){
  k = k + 4;
  j = k*4 + 16;
  sum = sum + j;
}
```

5. [7 points]: Rewrite the program after induction variable recognition and induction variable strength reduction (and no other optimizations):

Solution:

```
j = 16;
k = 0;
sum = 0;
while (j < 512) {
  k = k + 4;
  j = j + 16;
  sum = sum + j;
}
```

6. [7 points]: Rewrite the program after induction variable recognition, induction variable strength reduction, and induction variable elimination (and no other optimizations):

Solution:

```
j = 16;
sum = 0;
while (j < 512) {
  j = j + 4;
  sum = sum + j;
}
```

III Maximum Analysis

In this question, you will perform an analysis on programs with multiple whole number (non-negative integers) variables, `x`, `y`, and `z`, that determines the maximum value of each variable. Programs written in this language have four kinds of statements:

- `v1 = c;`
- `v1 = v2 + v3;`
- `if (...) { ... } else { ... }`
- `while (...) { ... }`

In these statements, `c` is some integer constant and `vi` is a given variable.

At each program point, the dataflow analysis maintains a maximum value (specifically, a non-negative integer) for each variable. Before the analysis, we initialize the dataflow information for each variable to 0. At merge points, we update the dataflow information by computing the join of information from all incoming edges. The join operator $\vee$ computes the least upper bound of the lattice elements. For example, for a program of 3 variables `x`, `y`, and `z`, a join is defined by the following:

$$[x \to c_1, y \to c_2, z \to c_3] \vee [x \to c_4, y \to c_5, z \to c_6] = [x \to \max(c_1, c_4), y \to \max(c_2, c_5), z \to \max(c_3, c_6)]$$

7. [4 points]: Given the join defined as above, define the partial order:

$$[x \to c_1, y \to c_2, z \to c_3] \leq [x \to c_4, y \to c_5, z \to c_6] \iff ??$$

Solution:

$$c_1 \leq c_4 \&\& c_2 \leq c_5 \&\& c_3 \leq c_6$$

8. [4 points]: Is the lattice complete? Explain your answer.

Solution: No, since for the set of non-negative integers there is no least upper bound for the entire set.

9. [4 points]: What is the transfer function $f_n(e)$ for a statement n of the form `x = c`? Here e is the incoming dataflow lattice element. Your transfer function should correctly model the semantics of the program and be as precise as possible.

$f_n(e) =$ `e[x → c]`

10. [4 points]: What is the transfer function $f_n(e)$ for a statement n of the form `z = x + y`? Here e is the incoming dataflow lattice element. Your transfer function should correctly model the semantics of the program and be as precise as possible.

$f_n(e) =$ `e[z → e[x] + e[y]]`

11. [8 points]: Consider the following program.

```
x = 0;
y = 0;
z = 0;
if (...) {
  x = 4;
  y = 7;
} else {
  x = 6;
  y = 4;
}
z = x + y;
```

A. [4 points]: What is the analysis result at the end of this program?
Solution: `e[x` $\rightarrow$ `6, y` $\rightarrow$ `7, z` $\rightarrow$ `13]`

B. [4 points]: What is the meet-over-paths result at the end of this program?
Solution: `e[x` $\rightarrow$ `6, y` $\rightarrow$ `7, z` $\rightarrow$ `11]`

12. [**4 points**]: Does the analysis always terminate? If so, explain why. If not, give an example program or control flow graph where it fails to terminate.

Solution: NO

```
x = 1;
while (x > 0) {
  x = x + 1;
}
```

IV Register Allocation

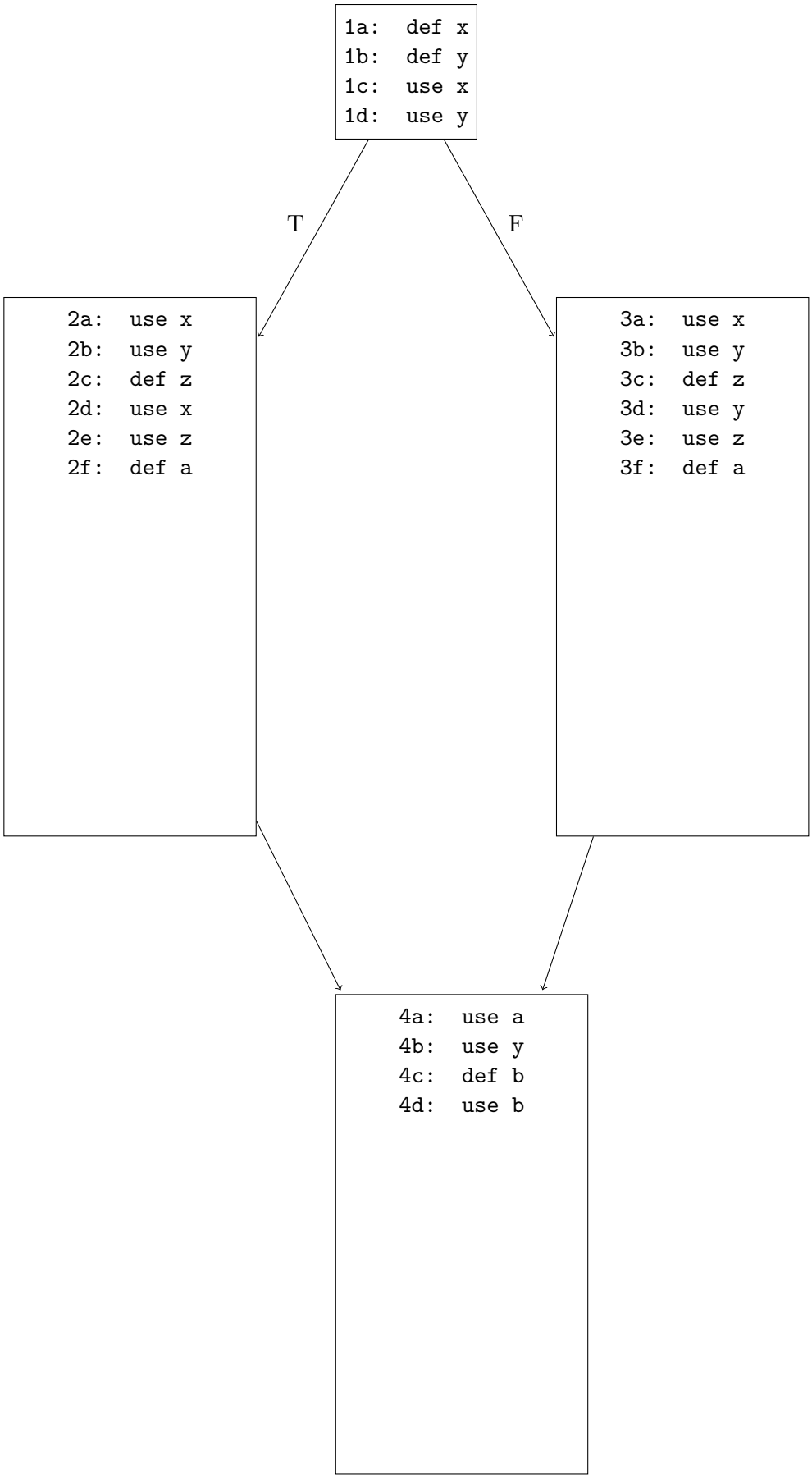
In this problem, you will perform register allocation for the following code. Each instruction is labeled with a number. Assume that you do not perform any other optimizations, and none of the variables is subsequently used.

```
1:  x = read_int();
2:  y = read_int();
3:  if (x < y) {
4:      z = x + y;
5:      a = x + z;
    } else {
6:      z = x * y;
7:      a = y + z;
    }
8:  b = a + y;
9:  print (b);
```

13. [4 points]:

On the next page, translate this program into a CFG of def and use statements. Be sure to include labels for each statement (block number followed by letter).

DO NOT DRAW THE CFG HERE, GO TO THE NEXT PAGE.


1a: def x
1b: def y
1c: use x
1d: use y
T
F
2a: use x
2b: use y
2c: def z
2d: use x
2e: use z
2f: def a
3a: use x
3b: use y
3c: def z
3d: use y
3e: use z
3f: def a
4a: use a
4b: use y
4c: def b
4d: use b

14. [4 points]: Write the set of def-use chains for each variable in the program. Write each def-use chain as a number pair (d, u) where d is the label of an instruction that defines the variable and u is the label of an instruction that uses that definition (include a pair for each use for any given definition). Use labels from your CFG in problem 13.

Solution:

`x`: (1a, 1c) (1a, 2a) (1a, 2d) (1a, 3a)

`y`: (1b, 1d) (1b, 2b) (1b, 3b) (1b, 3d) (1b, 4b)

`z`: (2c, 2e) (3c, 3e)

`a`: (2f, 4a) (3f, 4a)

`b`: (4c, 4d)

15. [4 points]: Write the set of webs in the program. Write each web as the set of instructions that belong to the web, in terms of the labels from your CFG in problem 13. We have given you names `w1-w7` for the webs, use only as many names as you need.

Solution:

`w1`: {1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d, 3a}

`w2`: {1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d, 2e, 2f, 3a, 3b, 3c, 3d, 3e, 3f, 4a, 4b}

`w3`: {2c, 2d, 2e}

`w4`: {3c, 3d, 3e}

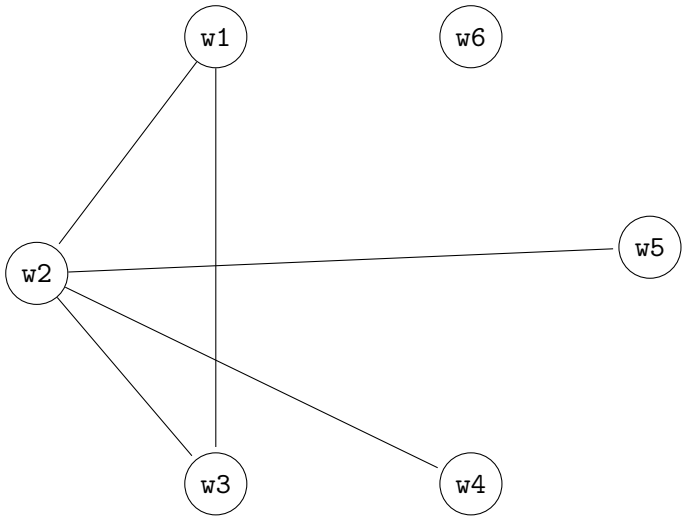
`w5`: {2f, 3f, 4a}

`w6`: {4c, 4d}

`w7`:

16. [4 points]: Draw the interference graph for the webs. Each node in the interference graph should represent one web. There should be an edge between two nodes if the two webs interfere. Label each node with the name (`w1-w7`) of the corresponding web.

Solution:



17. [4 points]: Suppose that the architecture we are targeting for compilation has two general purpose registers, `r1` and `r2`. Assume that the register allocater allocates at the granularity of variables.

A. [1 points]: Can we place all the variables in this code in registers (ignore any constraint due to calling convention)?

Solution: No, 3-clique between w1, w2, w3

B. [3 points]: If yes, specify the mapping of variables to registers. If no, specify where in the def-use CFG in 13 you would introduce spills and loads of variables to maximize performance.

Solution: Spill `y` at the beginning of block 2, load y back at the end of block 2.

Partial credit given to correct, but not optimium strategies.

V Parallelization

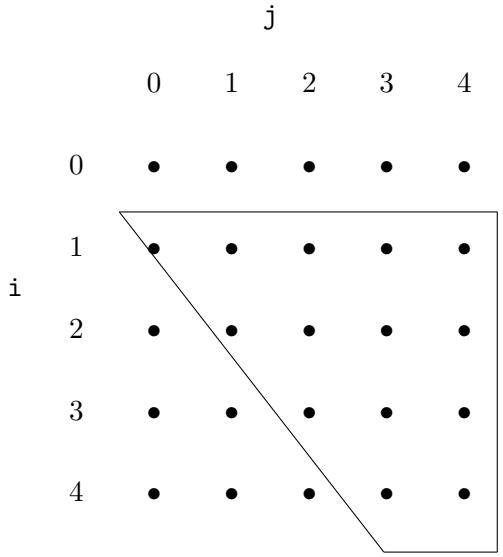
Consider the following program:

```
for (i = 1; i < n; i += 1) {
  for (j = i - 1; j < n; j += 1) {
    A[i, j] = A[i - 1, j + 3] - 2;
  }
}
```

`A[i, j]` means the element at the `i`-th row and `j`-th column in the 2-dimensional array `A`. Ignore out-of-bound array access.

18. [4 points]: Assume that `n = 5`. In the grid below, circle the dots that represent the iteration space for this loop and draw the distance vectors. Ignore out-of-range cases. Each dot represents the values of `i` and `j` for an iteration.

Solution:



19. [4 points]: What is the distance vector for these loops?

Solution:

$$\begin{pmatrix} 1 \\ -3 \end{pmatrix}$$

20. [4 points]: Without any other optimizations or transformations, is either loop fully parallelizable as a "FORALL" loop? If so, is it the outer loop (i), the inner loop (j), or both, that can be parallelized? Explain your answer.

Solution: Only the inner loop, due to $d_1 > 0$