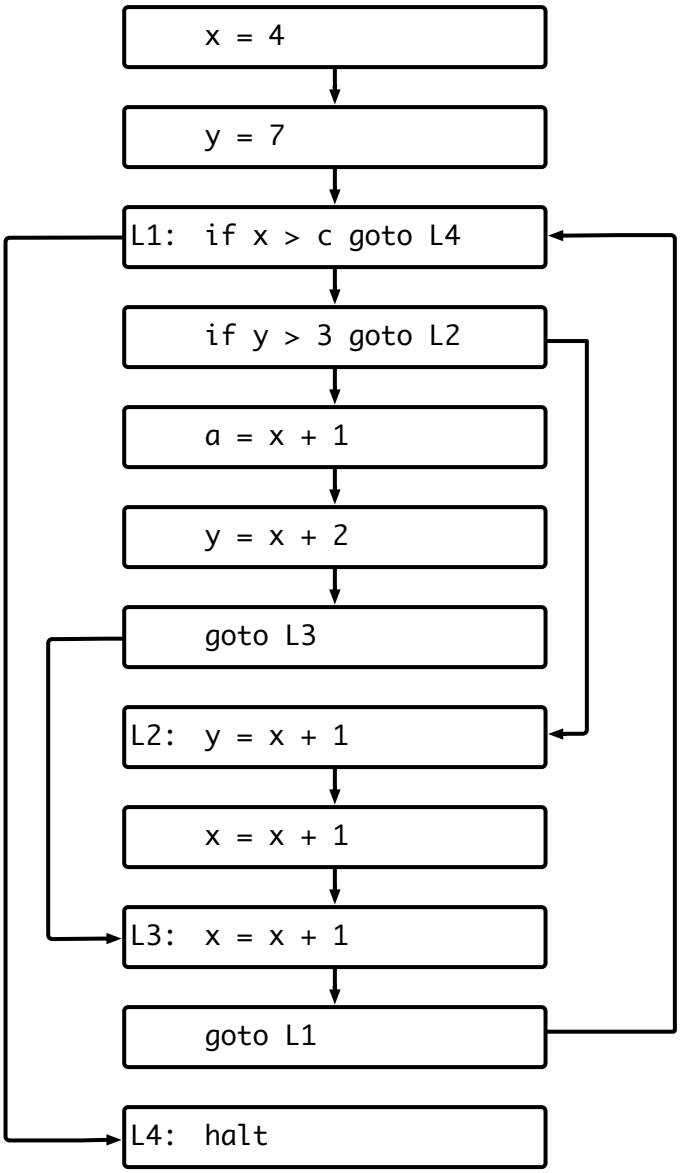
ECE 468
Problem Set 7: Dataflow analysis

1. Show the results of running a *reaching definition* analysis on the following piece of code: For each line of code, show which definitions reach that line of code by indicating the line number the definition occurred in.

```
    1: x = 4;
    2: y = 7;
L1  3: if (x > c) goto L4
    4:   if (y > 3) goto L2
    5:     a = x + 1;
    6:     y = x + 2;
    7:     goto L3
L2  8:     y = x + 1;
    9:     x = x + 1;
L3 10:   x = x + 1;
   11:   goto L1;
L4 12: halt
```

**Answer:** To begin, let us build the statement-level CFG for this program:




For each statement, we begin by calculating the GEN and KILL sets. In the case of reaching definitions, the GEN set for a statement is the set of definitions (line number & variable) created in that statement. The KILL set is all other definitions of that variable.

| Line # | GEN | KILL |
|---|---|---|
| 1 | [x, 1] | [x, 9], [x, 10] |
| 2 | [y, 2] | [y, 6], [y, 8] |
| 3 | | |
| 4 | | |
| 5 | [a, 5] | |
| 6 | [y, 6] | [y, 2], [y, 8] |
| 7 | | |
| 8 | [y, 8] | [y, 2], [y, 6] |
| 9 | [x, 9] | [x, 1], [x, 10] |
| 10 | [x, 10] | [x, 1], [x, 9] |
| 11 | | |
| 12 | | |

We now start calculating IN and OUT for each statement, according to dataflow equations. Reaching definitions is a forward analysis (because we care about what happened in the past), so the IN set for a statement is based on the *predecessors'* OUT sets. Reaching definitions is also an any-path analysis, so at merge statements, we combine the incoming sets using set union:

$$\begin{aligned} IN(s) &= \bigcup_{t \in pred(s)} OUT(t) \\ OUT(s) &= GEN(s) \cup (IN(s) - KILL(s)) \end{aligned}$$

All the sets start as empty (no reaching definitions), and we start with the first statement in the program:

$$\begin{aligned} IN(1) &= \{\} \\ OUT(1) &= \{[x, 1]\} \end{aligned}$$

Because changing OUT(1) might change the value of its successors, we next process statement 2:

$$\begin{aligned} IN(2) &= OUT(1) = \{[x, 1]\} \\ OUT(2) &= \{[x, 1], [y, 2]\} \end{aligned}$$

which then changes statement 3. Note that statement 3 has two predecessors, but OUT(11) is currently empty:

$$\begin{aligned} IN(3) &= OUT(2) \cup OUT(11) = \{[x, 1], [y, 2]\} \\ OUT(3) &= \{[x, 1], [y, 2]\} \end{aligned}$$

Statement 3 has two successors, statements 4 and 12, so we need to update both of their sets:

$$\begin{aligned} IN(4) &= OUT(3) = \{[x,1],[y,2]\} \\ OUT(4) &= \{[x,1],[y,2]\} \\ IN(12) &= OUT(3) = \{[x,1],[y,2]\} \\ OUT(12) &= \{[x,1],[y,2]\} \end{aligned}$$

Statement 4 has two successors, so we need to update statements 5 and 8:

$$\begin{aligned} IN(5) &= OUT(4) = \{[x,1],[y,2]\} \\ OUT(5) &= \{[x,1],[y,2],[a,5]\} \\ IN(8) &= OUT(4) = \{[x,1],[y,2]\} \\ OUT(8) &= \{[x,1],[y,8]\} \end{aligned}$$

Note that OUT(8) killed the [y, 2] definition because of its KILL set. Statement 5 has one successor, statement 6, and statement 8 has one successor, statement 9, so we need to update them:

$$\begin{aligned} IN(6) &= OUT(5) = \{[x,1],[y,2],[a,5]\} \\ OUT(6) &= \{[x,1],[y,6],[a,5]\} \\ IN(9) &= OUT(8) = \{[x,1],[y,8]\} \\ OUT(9) &= \{[x,9],[y,8]\} \end{aligned}$$

We now update the successors of statements 6 and 9:

$$\begin{aligned} IN(7) &= OUT(6) = \{[x,1],[y,6],[a,5]\} \\ OUT(7) &= \{[x,1],[y,6],[a,5]\} \\ IN(10) &= OUT(7) \cup OUT(9) = \{[x,9],[y,8],[x,1],[y,6],[a,5]\} \\ OUT(10) &= \{[x,10],[y,6],[y,8],[a,5]\} \end{aligned}$$

Note that multiple definitions of x and y made it to statement 10. We don't have to update the successor of statement 7, because it's statement 10. But we do need to update the successor of statement 10:

$$\begin{aligned} IN(11) &= OUT(10) = \{[x,10],[y,6],[y,8],[a,5]\} \\ OUT(11) &= \{[x,10],[y,6],[y,8],[a,5]\} \end{aligned}$$

And now we need to update the successor of statement 11, statement 3:

$$\begin{aligned} IN(3) &= OUT(2) \cup OUT(11) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(3) &= \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \end{aligned}$$

Because OUT(3) changed from its previous value, we need to update 4 and 12:

$$\begin{aligned} IN(4) &= OUT(3) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(4) &= \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ IN(12) &= OUT(3) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(12) &= \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \end{aligned}$$

And because OUT(4) changed, we need to update 5 and 8:

$$\begin{aligned} IN(5) &= OUT(4) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(5) &= \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ IN(8) &= OUT(4) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(8) &= \{[x,1],[x,10],[y,8],[a,5]\} \end{aligned}$$

So now we need to change 6 and 9:

$$\begin{aligned} IN(6) &= OUT(5) = \{[x,1],[y,2],[x,10],[y,6],[y,8],[a,5]\} \\ OUT(6) &= \{[x,1],[x,10],[y,6],[a,5]\} \\ IN(9) &= OUT(8) = \{[x,1],[x,10],[y,8],[a,5]\} \\ OUT(9) &= \{[x,9],[y,8],[a,5]\} \end{aligned}$$

Both OUT(6) and OUT(9) changed, so we need to update 7 and 10:

$$\begin{aligned} IN(7) &= OUT(6) = \{[x,1],[x,10],[y,6],[a,5]\} \\ OUT(7) &= \{[x,1],[x,10],[y,6],[a,5]\} \\ IN(10) &= OUT(7) \cup OUT(9) = \{[x,10],[x,9],[y,8],[x,1],[y,6],[a,5]\} \\ OUT(10) &= \{[x,10],[y,6],[y,8],[a,5]\} \end{aligned}$$

Note that OUT(7) changed, but OUT(10) *didn't*. Because OUT(10) didn't change, we don't need to update its successors. At this point, no more nodes need to be changed, so we can aggregate the final results:

| Line # | IN | OUT |
|---|---|---|
| 1 | | [x, 1] |
| 2 | [x, 1] | [x, 1], [y, 2] |
| 3 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] |
| 4 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] |
| 5 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] |
| 6 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [x, 10], [y, 6], [a, 5] |
| 7 | [x, 1], [x, 10], [y, 6], [a, 5] | [x, 1], [x, 10], [y, 6], [a, 5] |
| 8 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [x, 10], [y, 8], [a, 5] |
| 9 | [x, 1], [x, 10], [y, 8], [a, 5] | [x, 9], [y, 8], [a, 5] |
| 10 | [x, 1], [x, 9], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 10], [y, 6], [y, 8], [a, 5] |
| 11 | [x, 10], [y, 6], [y, 8], [a, 5] | [x, 10], [y, 6], [y, 8], [a, 5] |
| 12 | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] | [x, 1], [y, 2], [x, 10], [y, 6], [y, 8], [a, 5] |

2. Show the results of running an *available expression* analysis on the code, by indicating which expressions are available at each instruction.

**Answer:** We begin by showing the GEN and KILL sets for the program:

| Line # | GEN | KILL |
|---|---|---|
| 1 | | (x + 1), (x + 2) |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | (x + 1) | |
| 6 | (x + 2) | |
| 7 | | |
| 8 | (x + 1) | |
| 9 | (x + 1) | (x + 1), (x + 2) |
| 10 | (x + 1) | (x + 1), (x + 2) |
| 11 | | |
| 12 | | |

Available expressions is a forward, *all-path* analysis. The dataflow equations look similar to those for reaching definitions, except that set union is replaced with set intersection (an expression is available only if it is available along all incoming paths). Also, think about what happens to a statement like `x = x + 1`. Although the expression (x + 1) is computed, it is immediately killed by overwriting x. We need to take this into account when computing available expressions:

$$IN(s) = \bigcap_{t \in pred(s)} OUT(t)$$

$$OUT(s) \quad = \quad (GEN(s) \cup IN(s)) - KILL(s)$$

We will not walk through the full computation of available expressions here – it proceeds similarly to the previous problem. The expressions available at any statement are the IN set for that statement:

| **Line #** | **IN** |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | (x + 1) |
| 7 | (x + 1), (x + 2) |
| 8 | |
| 9 | (x + 1) |
| 10 | |
| 11 | |
| 12 | |

3. In this problem, your goal is to develop a dataflow analysis to find `uninitialized` values. A common error in programs is using a variable that has not yet been defined. Write an analysis that can detect any use of a variable that has not been defined. Do this by tracking, for each variable in a program, whether it is currently defined or not. Hint: think about how this analysis relates to a reaching definition analysis.

   (a) This is a bitvector analysis, where each object of interest in the analysis takes on the value 0 or 1. a) What are the objects of interest in this analysis? b) What does it mean for an object to have value 0? c) What does it mean for the object to have value 1?

   **Answer:** The objects of interest are the variables in the program. We care if a variable has been initialized or not. If the variable has value 0, it has not been initialized. If it has value 1, it has been initialized.

   (b) Which direction should this analysis use?

   **Answer:** To determine if a variable has been initialized, we care if the variable has ever been assigned to *before*. Because we are looking back in time, we want to use a forward analysis.

   (c) Give the GEN and KILL sets for this analysis, in terms of **def(s)**, the variables defined in a statement.

**Answer:** If a variable is defined in a statement, then it is clearly initialized. Hence:

$$GEN(s) = \mathbf{def(s)}$$

Interestingly, there is no way to "undefine" a variable: once it has been defined, it will always be defined, so there is no way to kill a definition.

$$KILL(s) = \{\}$$

(d) Define IN and OUT for this analysis (don't forget which direction your analysis is running: if you're running forward, IN should be defined in terms of a statement's predecessors, and OUT should be defined in terms of IN. If you're running backwards, OUT is defined in terms of a statement's successors, and IN is defined in terms of OUT). Don't forget to think about how your analysis should behave at merge statements.

**Answer**: A variable only counts as defined if it defined along all paths that can reach this variable. That means at merge statements the set of variables that are defined are only the variables that are defined along both incoming paths. This means we want to use set intersection at merges:

$$IN(s) = \bigcap_{t \in pred(s)} OUT(t)$$
$$OUT(s) = IN(s) \cup GEN(s)$$

Note that we simplified the definition of OUT(s) because we know that KILL(s) is empty.

(e) How should this analysis be initialized? (In a forward analysis, what is IN of the first statement, in a backward analysis, what is OUT of the last statement?

**Answer:** Nothing is defined at the beginning of the program, so IN(start) is empty.