

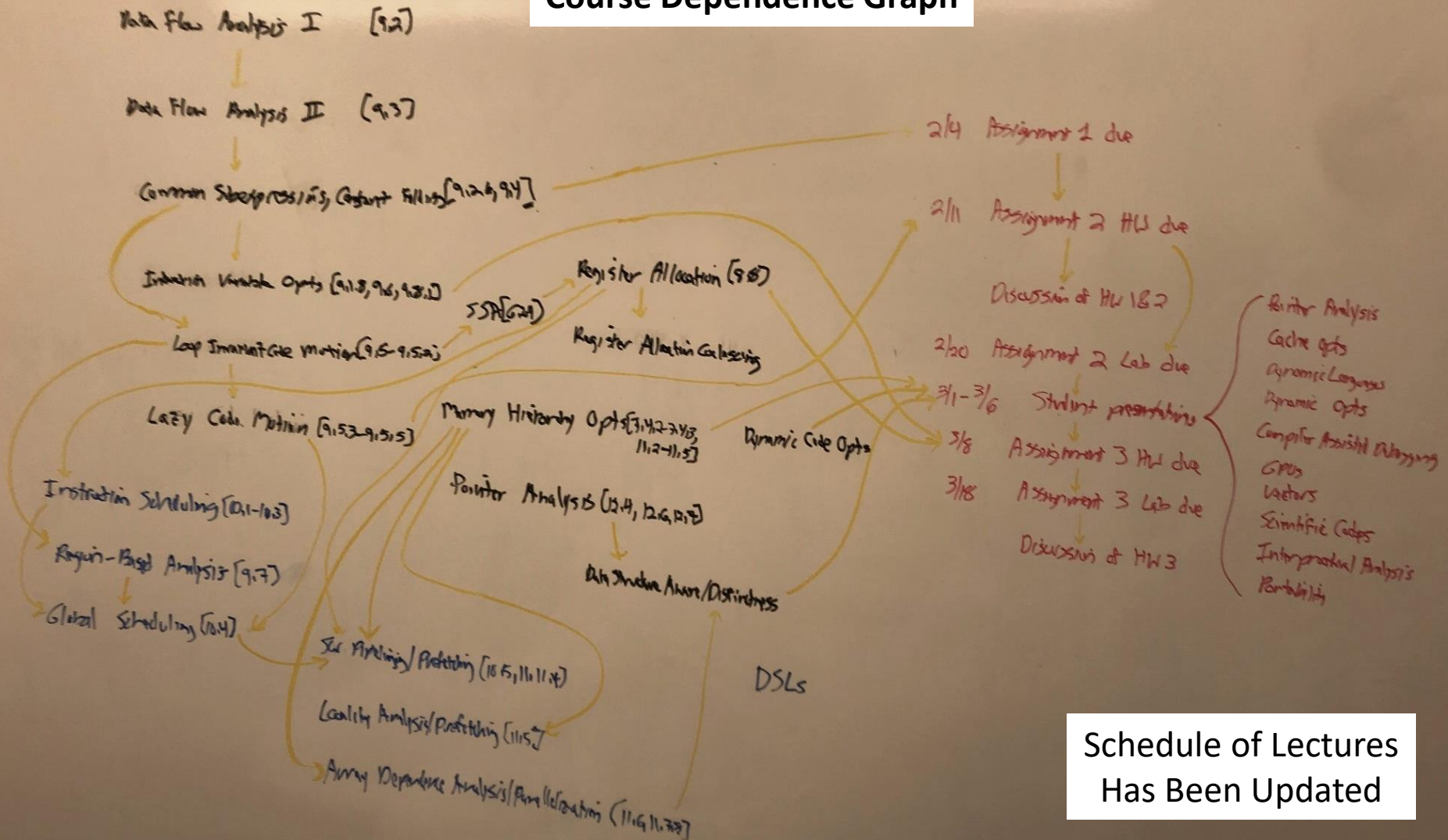
Lecture 5

Introduction to Data Flow Analysis

- I. Structure of data flow analysis
- II. Example 1: Reaching definition analysis
- III. Example 2: Liveness analysis
- IV. Framework

What is This?

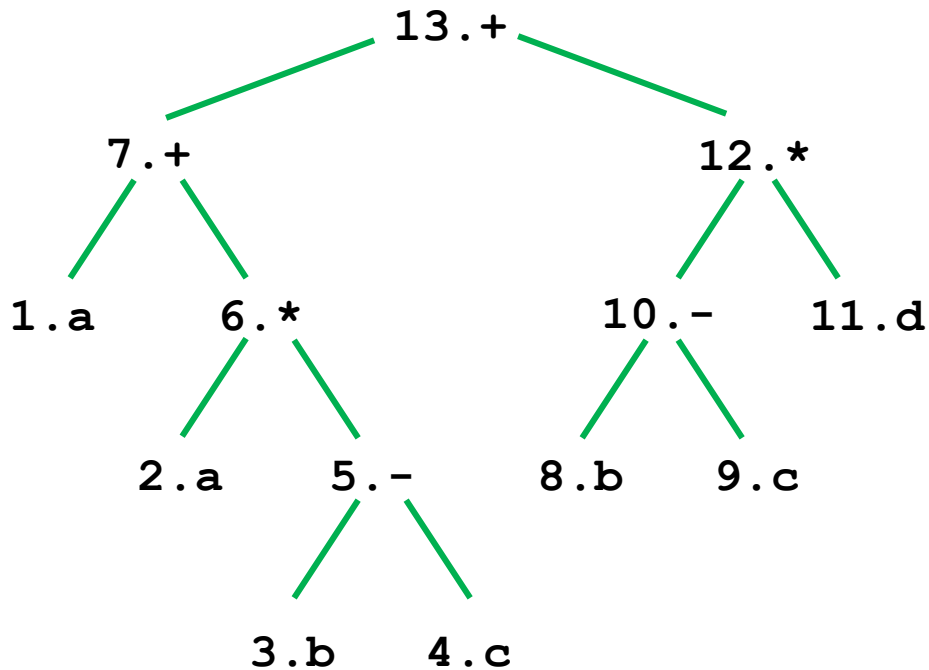
Course Dependence Graph



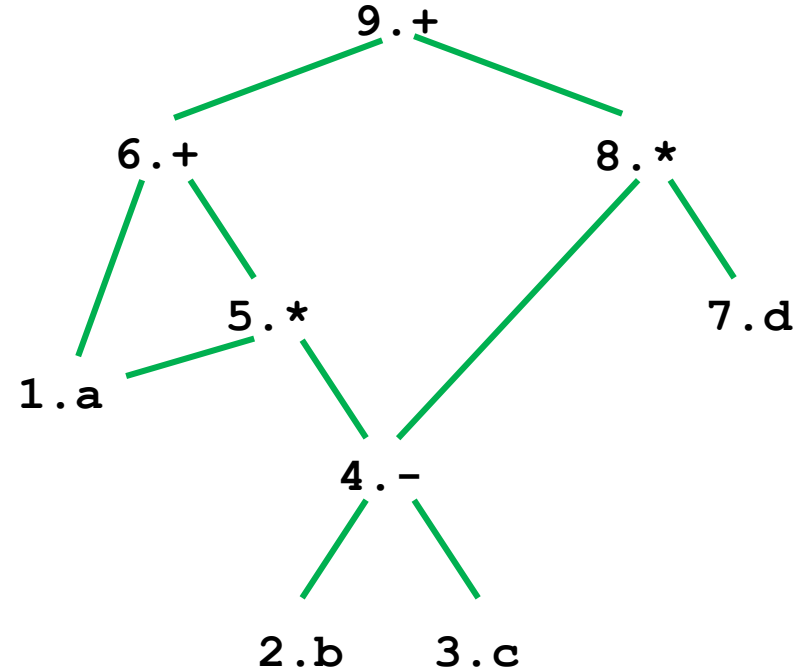
Review: Expression DAG

Example 1:

- grammar (for bottom-up parsing): $E \rightarrow E + T \mid E - T \mid T$, $T \rightarrow T * F \mid F$, $F \rightarrow (E) \mid id$
- expression: $a + a * (b - c) + (b - c) * d$



Parse tree



Expression DAG

Review: Value Numbering

Data structure:

VALUES = Table of

expression /* [OP, valnum1, valnum2] */

var /* name of variable currently holding expr */

var2value() /* variable's current value number */

a = b+c

t1 = b + c

a = t1

b = a-d

t2 = t1 - d

b = t2

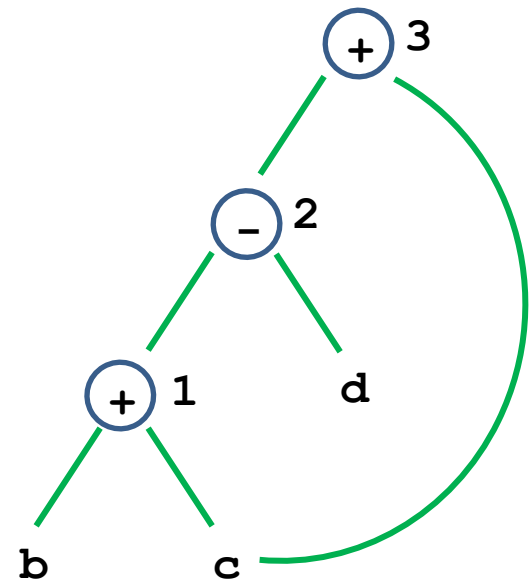
c = b+c

t3 = t2 + c

c = t3

d = a-d

d = t2



What is Data Flow Analysis?

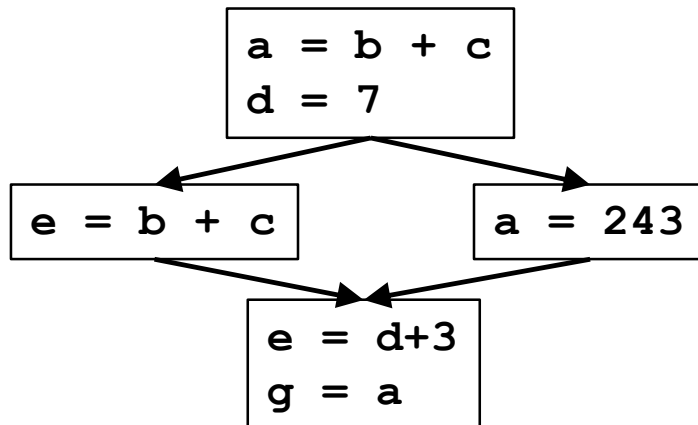
- **Local analysis (e.g. value numbering)**
 - analyze effect of each instruction
 - compose effects of instructions to derive information from beginning of basic block to each instruction

- **Data flow analysis**
 - analyze effect of each basic block
 - compose effects of basic blocks to derive information at basic block boundaries
 - from basic block boundaries, apply local technique to generate information on instructions

[ALSU 9.2]

What is Data Flow Analysis? (Cont.)

- **Data flow analysis:**
 - Flow-sensitive: sensitive to the control flow in a function
 - Intraprocedural analysis
- **Examples of optimizations:**
 - Constant propagation
 - Common subexpression elimination
 - Dead code elimination



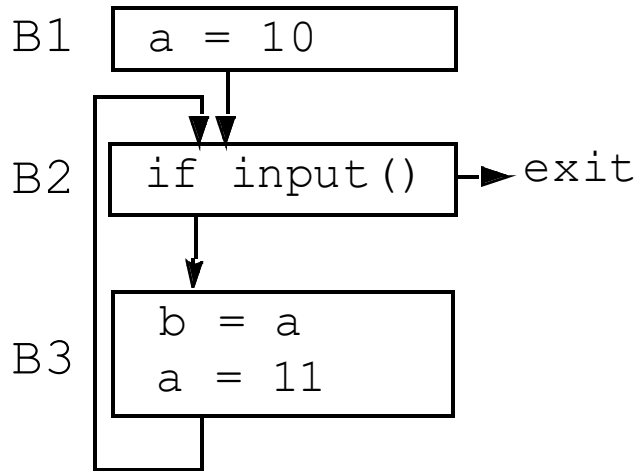
For each variable x , determine:

Value of x ?

Which “definition” defines x ?

Is the definition still meaningful (live)?

Static Program vs. Dynamic Execution



- **Statically:** Finite program
- **Dynamically:** Can have infinitely many possible execution paths
- **Data flow analysis abstraction:**
 - For each point in the program:
combines information of all the instances of the same program point.
- **Example of a data flow question:**
 - Which definition defines the value used in statement “`b = a`”?

Effects of a Basic Block

- Effect of a statement: $a = b+c$
 - **Use** source variables (b and c)
 - **Kills** an old definition (old definition of a)
 - Defines a new **definition** (a)
- Compose effects of statements -> Effect of a basic block
 - A **locally exposed use** in a b.b. is a use of a data item that is not preceded in the b.b. by a definition of the data item.
 - Any definition of a data item in the b.b. **kills** all other definitions of the same data item.
 - A **locally available definition** = last definition of data item in b.b.

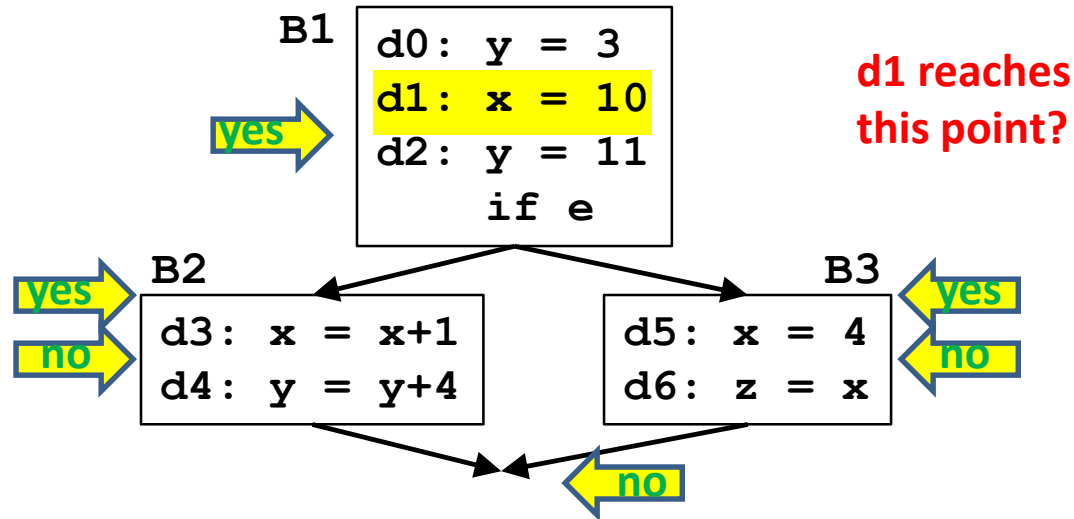
```
t1 = r1+r2
r2 = t1
t2 = r2+r1 ←
r1 = t2
t3 = r1*r1
r2 = t3
if r2>100 goto L1
```

locally exposed uses? r1

kills any definitions? Any other
defn of t2
in program

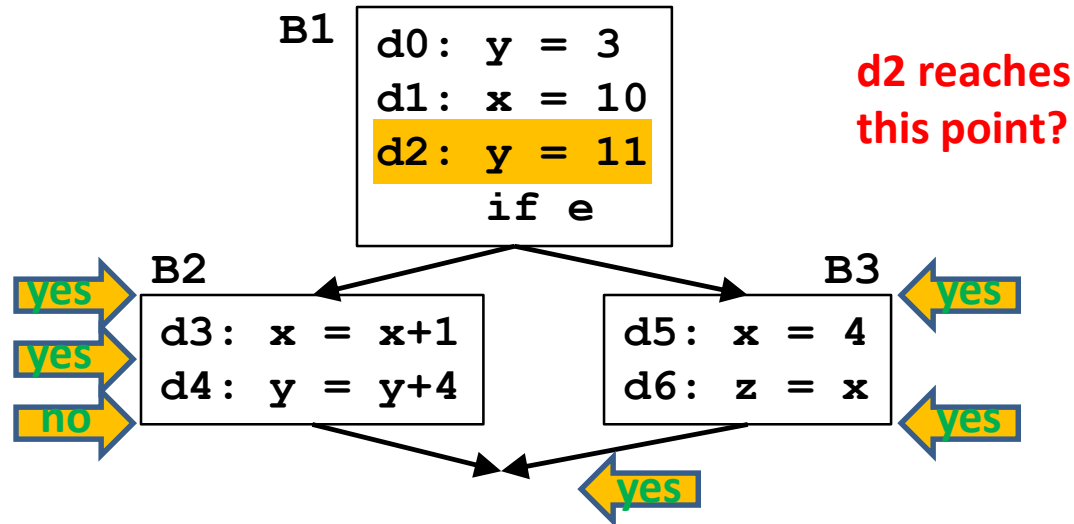
locally available definition? t2

II. Reaching Definitions



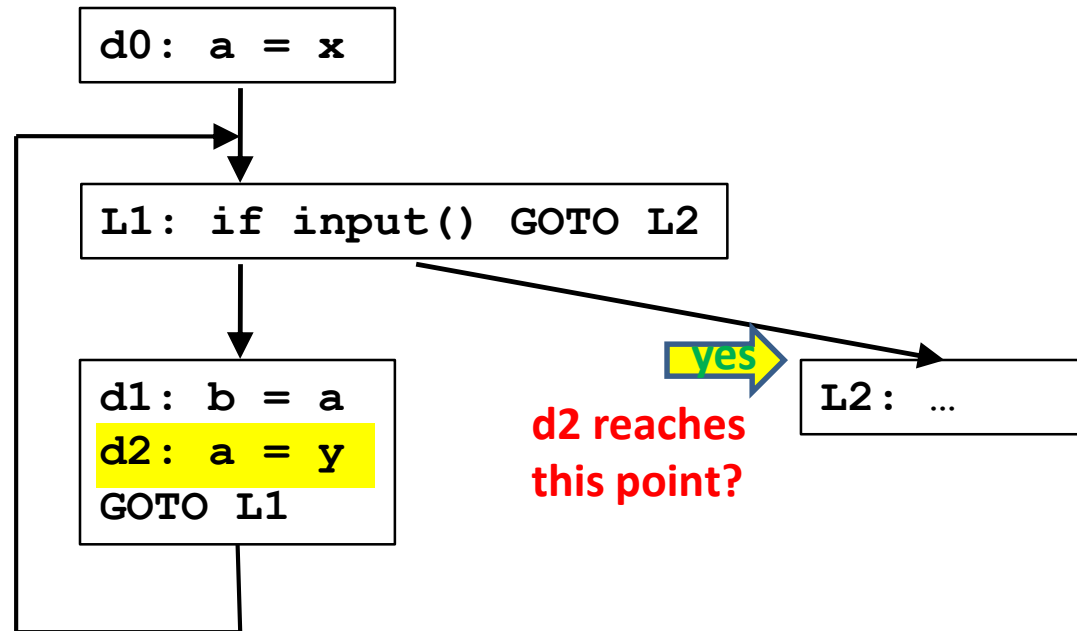
- Every assignment is a **definition**
- A **definition** d **reaches** a point p if **there exists** path from the point immediately following d to p such that d is **not killed** (overwritten) along that path.
- Problem statement
 - For each point in the program, determine if each definition in the program reaches the point
 - A bit vector per program point, vector-length = #defs

Reaching Definitions

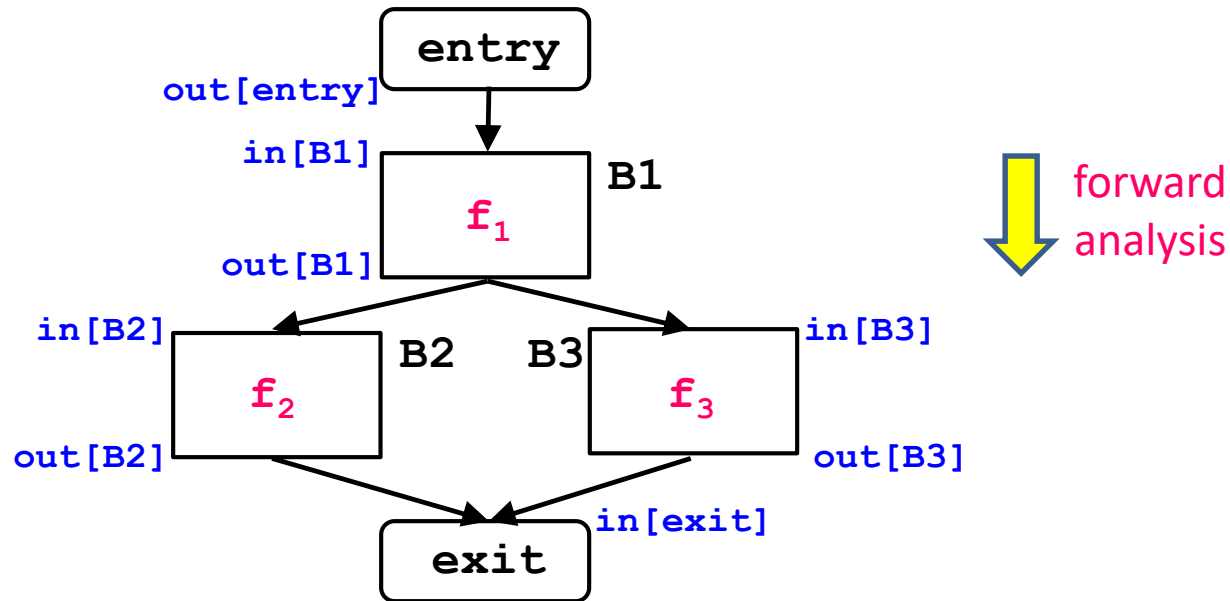


- Every assignment is a **definition**
- A **definition** d **reaches** a point p if **there exists** path from the point immediately following d to p such that d is **not killed** (overwritten) along that path.
- Problem statement
 - For each point in the program, determine if each definition in the program reaches the point
 - A bit vector per program point, vector-length = #defs

Reaching Definitions: Another Example

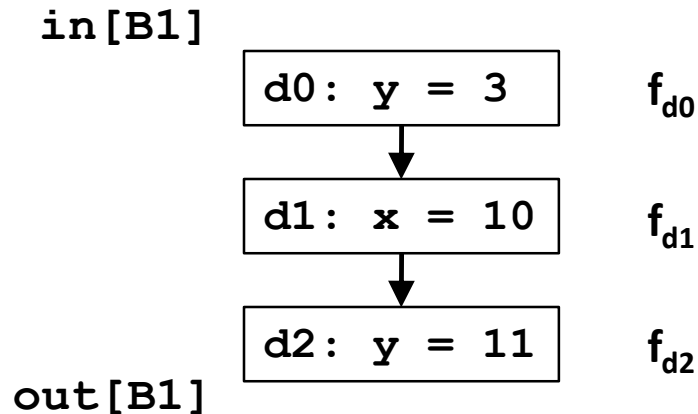


Data Flow Analysis Schema



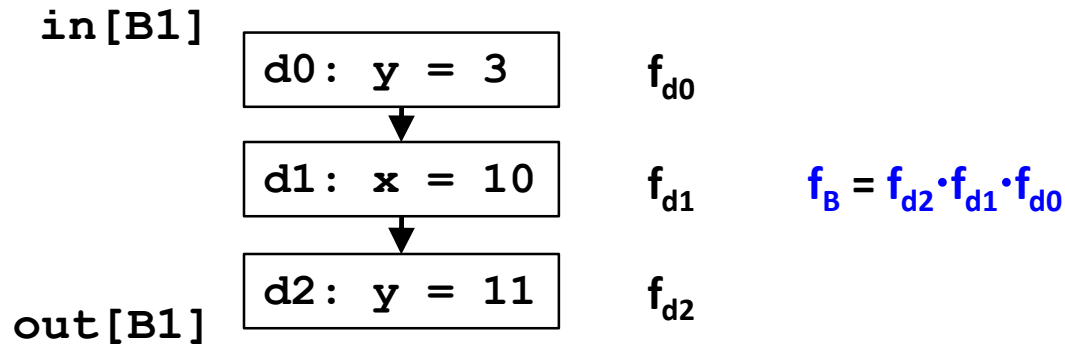
- Build a **flow graph** (nodes = basic blocks, edges = control flow)
- Set up a set of equations between $in[b]$ and $out[b]$ for all basic blocks b
 - Effect of **code in basic block**:
 - **Transfer function f_b** relates $in[b]$ and $out[b]$, for same b
 - Effect of **flow of control**:
 - relates $out[b]$, $in[b']$ if b and b' are **adjacent**
- Find a solution to the equations

Effects of a Statement



- f_s : A transfer function of a statement
 - abstracts the execution with respect to the problem of interest
- Consider Reaching Definitions. For a statement s (e.g., $d: x = y + z$):
 $out[s] = f_s(in[s]) = Gen[s] \cup (in[s] - Kill[s])$
 - **Gen[s]**: definitions generated: $Gen[s] = \{d\}$
 - **Propagated** definitions: $in[s] - Kill[s]$,
where **Kill[s]**=set of all other defs to x in the rest of program

Effects of a Basic Block



- Transfer function of a statement s :
 - $\text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] - \text{Kill}[s])$
- Transfer function of a **basic block B**:
 - Composition of transfer functions of statements in B
- $\text{out}[B] = f_B(\text{in}[B]) = f_{d2} f_{d1} f_{d0}(\text{in}[B])$

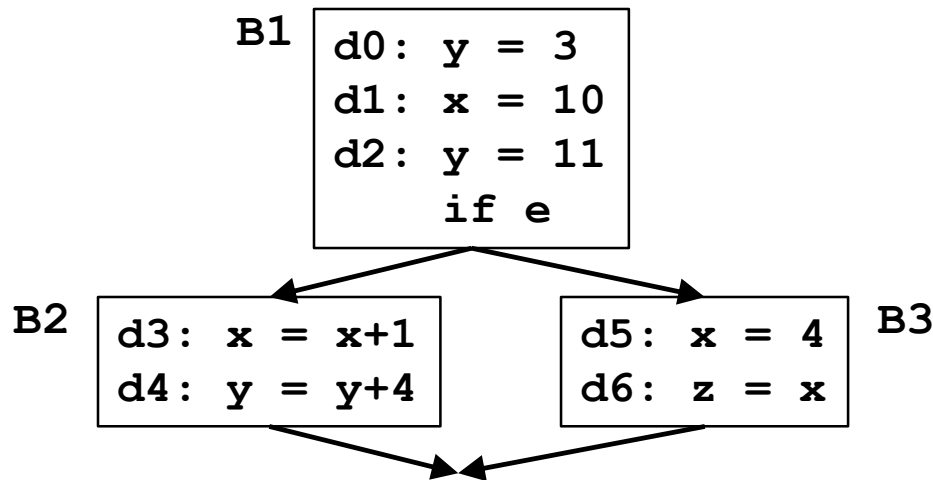
$$= \text{Gen}[d_2] \cup ((\text{Gen}[d_1] \cup ((\text{Gen}[d_0] \cup (\text{in}[B] - \text{Kill}[d_0])) - \text{Kill}[d_1])) - \text{Kill}[d_2])$$

$$= \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]) - \text{Kill}[d_2]) \cup$$

$$\quad (\text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1] \cup \text{Kill}[d_2]))$$

$$= \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B])$$
 - **Gen[B]**: locally available definitions (defined locally & reaches end of bb)
 - **Kill[B]**: set of definitions killed by B

Reaching Definitions Example

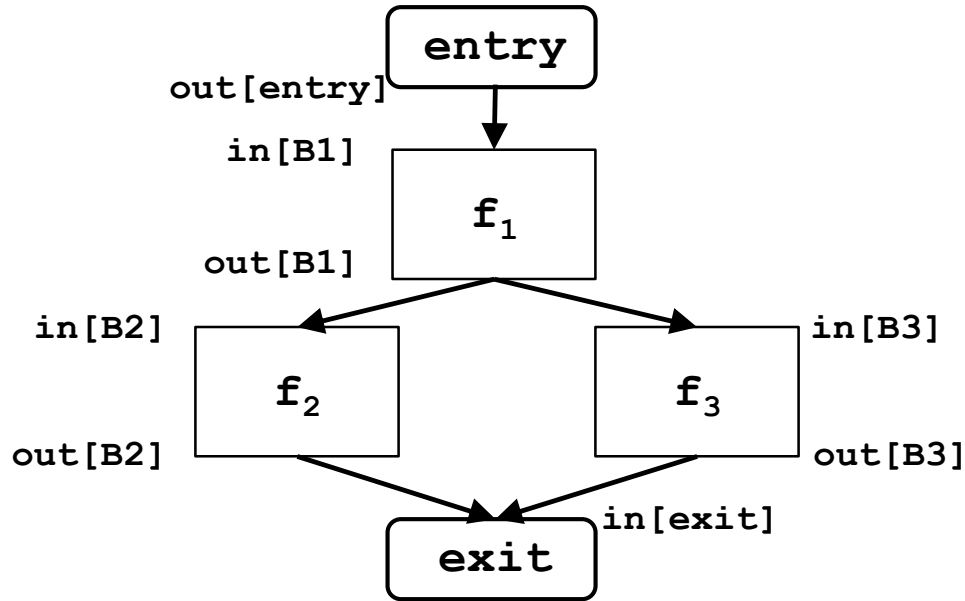


f	Gen	Kill
1	{1,2}	{0,2,3,4,5}
2	{3,4}	{0,1,2,5}
3	{5,6}	{1,3}

Subtlety: d0 & d2
kill each other, but
d2 is still generated

- **transfer function** f_b of a basic block b : $OUT[b] = f_b(IN[b])$
incoming reaching definitions \rightarrow outgoing reaching definitions
- A basic block b
 - **generates** definitions: $Gen[b]$,
– set of definitions in b that reach end of b
 - **kills** definitions: $in[b] - Kill[b]$,
where $Kill[b]$ =set of defs killed by defs in b
- **$out[b] = Gen[b] \cup (in[b] - Kill[b])$**

Effects of the Edges (acyclic)

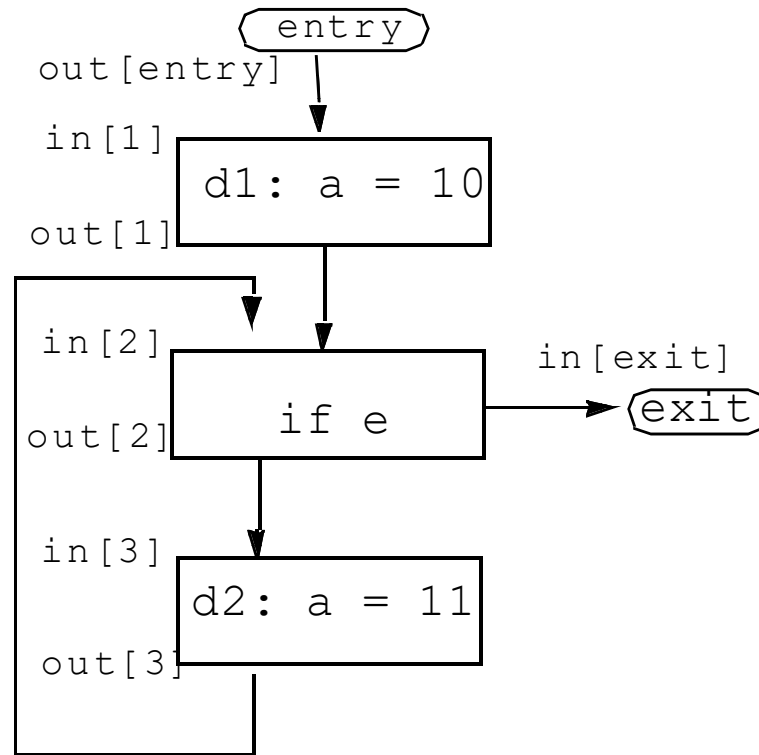


- $out[b] = f_b(in[b])$
- Join node: a node with multiple predecessors
- **meet** operator:

$in[b] = out[p_1] \cup out[p_2] \cup \dots \cup out[p_n]$, where
 p_1, \dots, p_n are all the predecessors of b

$in[exit] = out[B2] \cup out[B3]$

Cyclic Graphs



- Equations still hold
 - $out[b] = f_b(in[b])$
 - $in[b] = out[p_1] \cup out[p_2] \cup \dots \cup out[p_n]$, p_1, \dots, p_n pred.
- Find: fixed point solution

$$in[B2] = out[B1] \cup out[B3]$$

Reaching Definitions: Iterative Algorithm

input: control flow graph $CFG = (N, E, \text{Entry}, \text{Exit})$

// Boundary condition

$\text{out}[\text{Entry}] = \emptyset$

// Initialization for iterative algorithm

For each basic block B other than Entry

$\text{out}[B] = \emptyset$

// iterate

While (changes to any $\text{out}[]$ occur) {

For each basic block B other than Entry {

$\text{in}[B] = U(\text{out}[p])$, for all predecessors p of B

$\text{out}[B] = f_B(\text{in}[B])$ // $\text{out}[B] = \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B])$

}

Reaching Definitions: Worklist Algorithm

```
input: control flow graph CFG = (N, E, Entry, Exit)

// Initialize
  out[Entry] =  $\emptyset$            // could set out[Entry] to special def
                                // if reaching then undefined use

  For all nodes i
    out[i] =  $\emptyset$            // could optimize by out[i]=gen[i]
  ChangedNodes = N

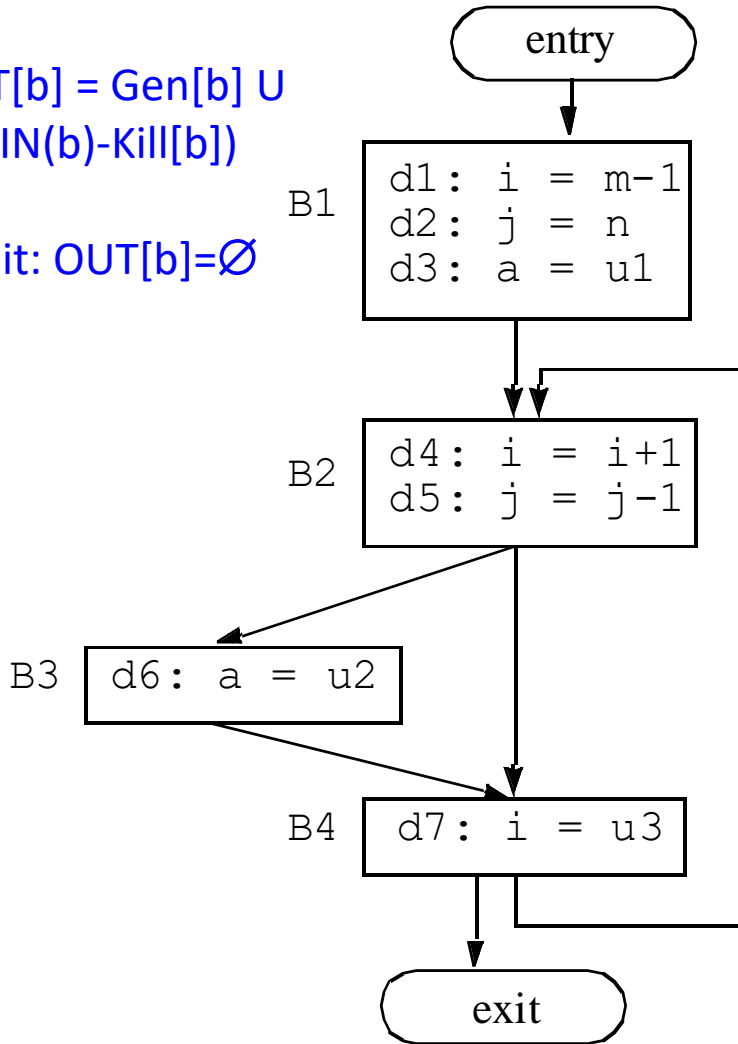
// iterate
  While ChangedNodes  $\neq \emptyset$  {
    Remove i from ChangedNodes
    in[i] = U(out[p]), for all predecessors p of i
    oldout = out[i]
    out[i] = fi(in[i])         // out[i]=gen[i] U(in[i]-kill[i])
    if (oldout  $\neq$  out[i]) {
      for all successors s of i
        add s to ChangedNodes
    }
  }
}
```

Reaching Definitions Example

$$IN[b] = U(OUT[pred(b)])$$

$$OUT[b] = Gen[b] \cup (IN[b] - Kill[b])$$

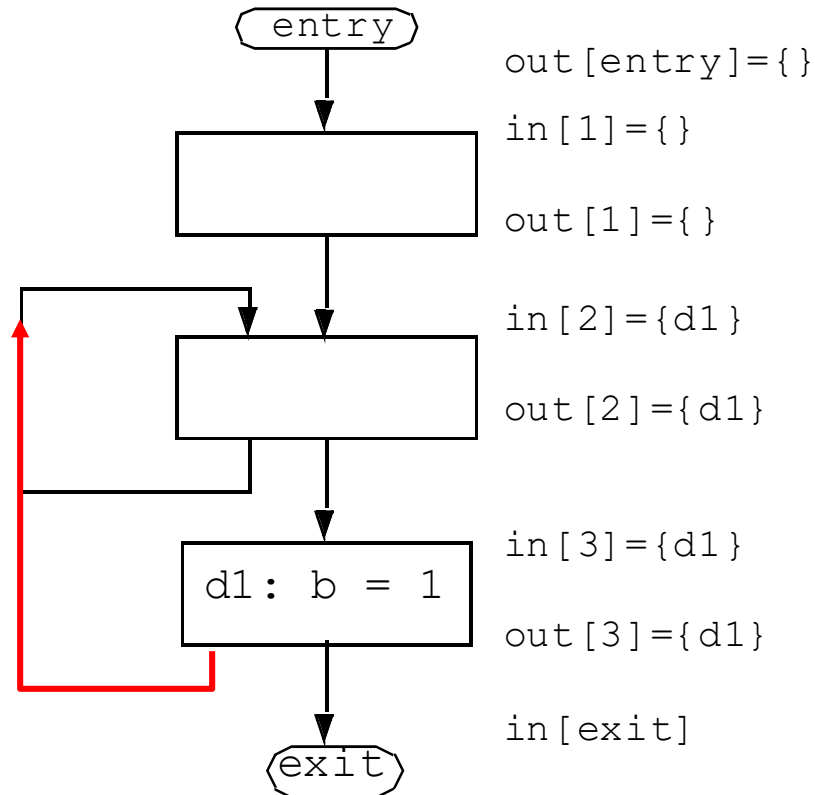
$$Init: OUT[b] = \emptyset$$



	First Pass	Second Pass
IN[B1]	000 00 0 0	000 00 0 0
OUT[B1]	111 00 0 0	111 00 0 0
IN[B2]	111 00 0 0	111 01 1 1
OUT[B2]	001 11 0 0	001 11 1 0
IN[B3]	001 11 0 0	001 11 1 0
OUT[B3]	000 11 1 0	000 11 1 0
IN[B4]	001 11 1 0	001 11 1 0
OUT[B4]	001 01 1 1	001 01 1 1
IN[exit]	001 01 1 1	001 01 1 1

Fixed point!

A valid solution to Reaching Definitions?



Fixed point:
another iteration
won't change
in/out values

- Is the solution a fixed point? **yes** Is it valid? **no**
- Will the worklist algorithm generate this answer? **no**
- What if add control flow edge shown in red? **yes**

III. Live Variable Analysis

- **Definition**

- A variable v is **live** at point p if
 - the value of v is used along some path in the flow graph starting at p .
- Otherwise, the variable is **dead**.

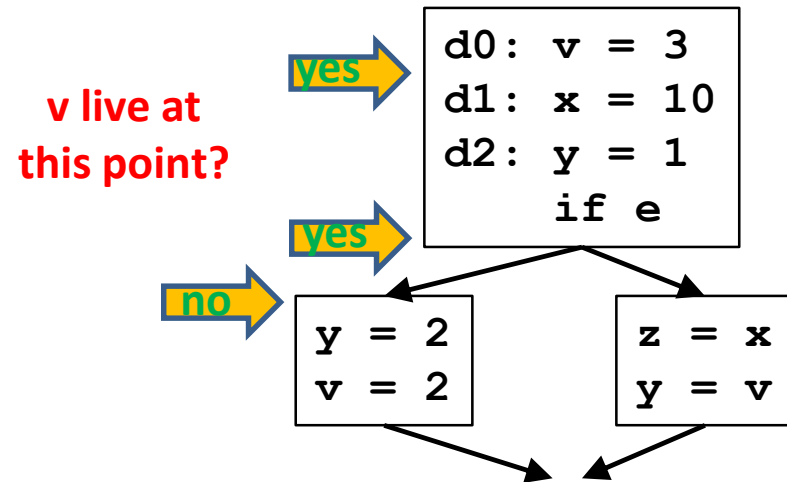
- **Motivation**

- e.g. register allocation

```
for i = 0 to n
  ... i ...
...
for i = 0 to n
  ... i ...
```

- **Problem statement**

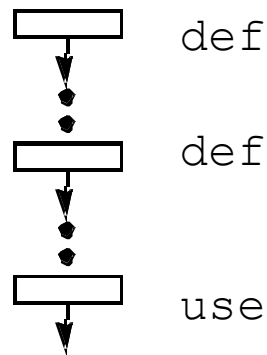
- For each basic block
 - determine if each variable is live in each basic block
- Size of bit vector: one bit for each variable



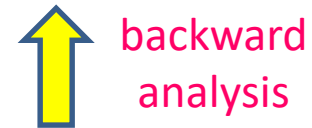
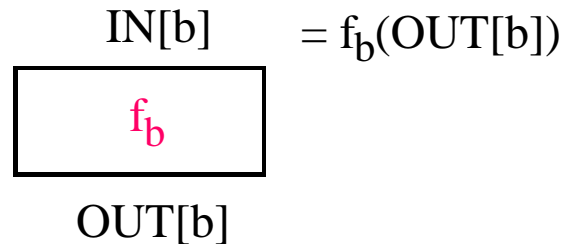
Live Variables: Effects of a Basic Block (Transfer Function)

- Insight: Trace uses **backwards** to the definitions

an execution path



control flow



- A basic block **b** can

- generate live variables: **Use[b]**
 - set of locally exposed uses in b
- propagate incoming live variables: **OUT[b] - Def[b]**,
 - where **Def[b]** = set of variables defined in b.b.

example

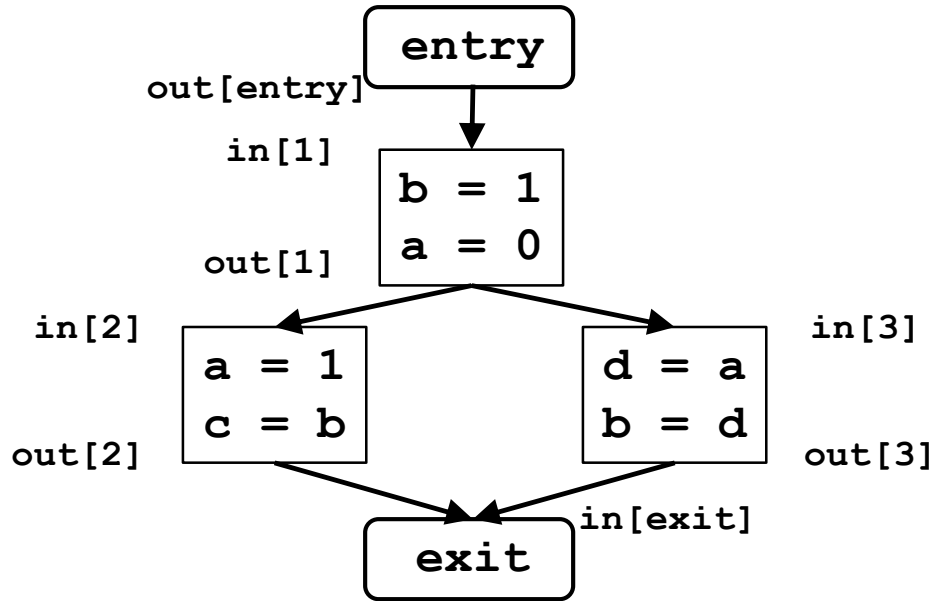
d4: d = 1
 d5: c = a
 d6: a = 4

- transfer function** for block b:

$$\text{in}[b] = \text{Use}[b] \cup (\text{out}[b] - \text{Def}[b])$$

$$\text{IN}[b] = \{a\} \cup (\text{OUT}[b] - \{a, c, d\})$$

Flow Graph



f	Use	Def
1	{}	{a,b}
2	{b}	{a,c}
3	{a}	{b,d}

- $in[b] = f_b(out[b])$
- **Join node**: a node with multiple successors
- **meet** operator:
 $out[b] = in[s_1] \cup in[s_2] \cup \dots \cup in[s_n]$, where
 s_1, \dots, s_n are all successors of b

Live Variables: Iterative Algorithm

input: control flow graph $CFG = (N, E, \text{Entry}, \text{Exit})$

// Boundary condition

$\text{in}[\text{Exit}] = \emptyset$

// Initialization for iterative algorithm

For each basic block B other than Exit

$\text{in}[B] = \emptyset$

// iterate

While (changes to any $\text{in}[]$ occur) {

For each basic block B other than Exit {

$\text{out}[B] = U(\text{in}[s])$, for all successors s of B

$\text{in}[B] = f_B(\text{out}[B])$ // $\text{in}[B] = \text{Use}[B] \cup (\text{out}[B] - \text{Def}[B])$

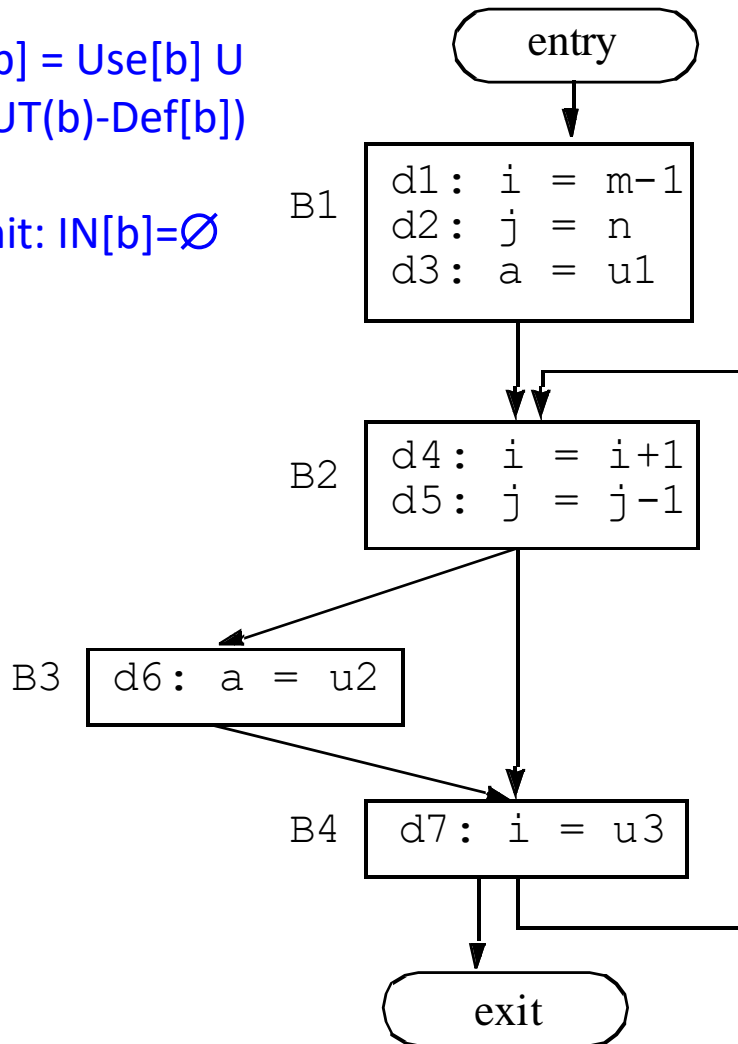
}

Live Variables Example

$OUT[b] = U(IN[succ(b)])$

$IN[b] = Use[b] \cup (OUT[b] - Def[b])$

Init: $IN[b] = \emptyset$



	First Pass	Second Pass
OUT[entry]	{m,n,u1,u2,u3}	{m,n,u1,u2,u3}
IN[B1]	{m,n,u1,u2,u3}	{m,n,u1,u2,u3}
OUT[B1]	{i,j,u2,u3}	{i,j,u2,u3}
IN[B2]	{i,j,u2,u3}	{i,j,u2,u3}
OUT[B2]	{u2,u3}	{j,u2,u3}
IN[B3]	{u2,u3}	{j,u2,u3}
OUT[B3]	{u3}	{j,u2,u3}
IN[B4]	{u3}	{j,u2,u3}
OUT[B4]	{}	{i,j,u2,u3}

Fixed point!

IV. Framework

	Reaching Definitions	Live Variables
Domain	Sets of definitions	Sets of variables
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge out[pred(b)]$	backward: $in[b] = f_b(out[b])$ $out[b] = \wedge in[succ(b)]$
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	$f_b(x) = Use_b \cup (x - Def_b)$
Meet Operation (\wedge)	\cup	\cup
Boundary Condition	$out[entry] = \emptyset$	$in[exit] = \emptyset$
Initial interior points	$out[b] = \emptyset$	$in[b] = \emptyset$

Other Data Flow Analysis problems fit into this general framework,
e.g., Available Expressions [ALSU 9.2.6]

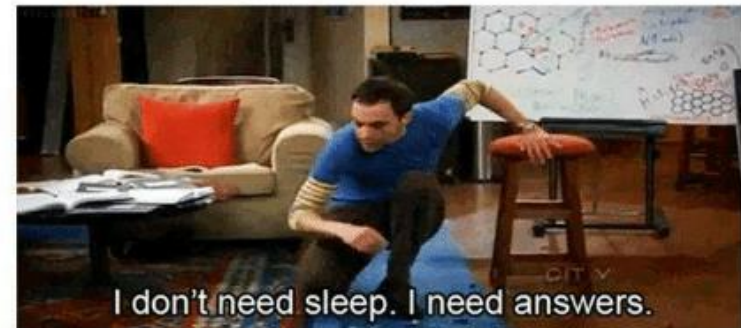
Key Questions

- **Correctness**
 - equations are satisfied, if the analysis algorithm terminates.
- **Precision: how good is the answer?**
 - is the answer ONLY a union of all possible executions?
- **Convergence: will the analysis terminate?**
 - or, could there always be some nodes that change?
- **Speed: how fast is the convergence?**
 - how many times will we visit each node?

lecture

Friday

When it's ~~3 AM~~ and the
~~episode~~ ends with a
cliffhanger 😊



Today's Class

- I. Structure of data flow analysis
- II. Example 1: Reaching definition analysis
- III. Example 2: Liveness analysis
- IV. Framework

Monday's Class

- Foundations of Data Flow Analysis
 - ALSU 9.3