Dataflow Analysis

CS 6340

What Is Dataflow Analysis?

- Static analysis reasoning about flow of data in program
- Different kinds of data: constants, variables, expressions
- Used by bug-finding tools and compilers

The WHILE Language

```
x = 5;
y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1
}
```

(statement) S ::= x = a | S1 ; S2 |
if (b) { S1 } else { S2 } |
while (b) { S1 }

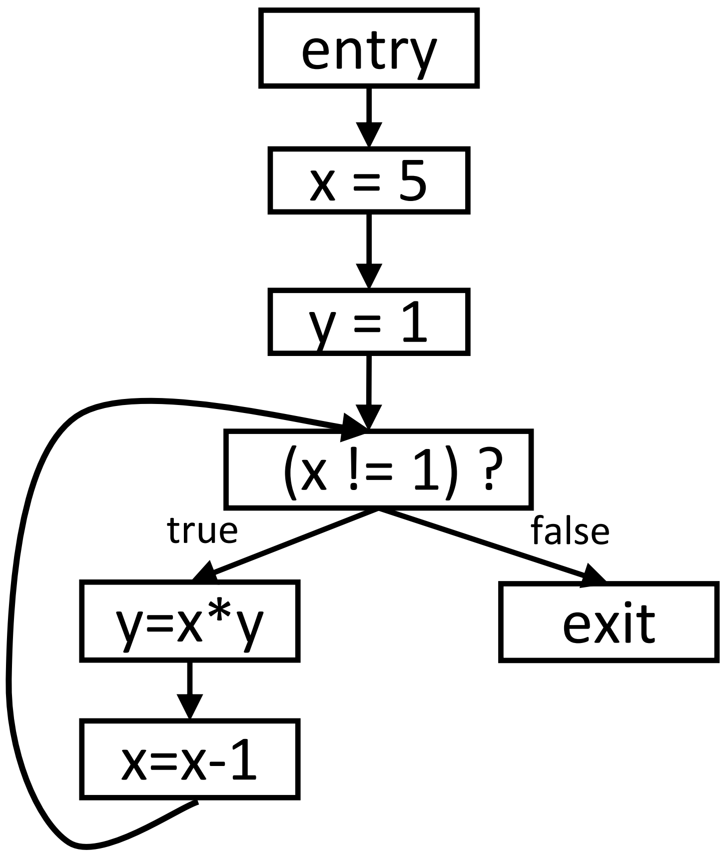
(arithmetic expression) a ::= x | n | a1 * a2 | a1 - a2

(boolean expression) b ::= true | !b | b1 && b2 |
a1 != a2

(integer variable) x

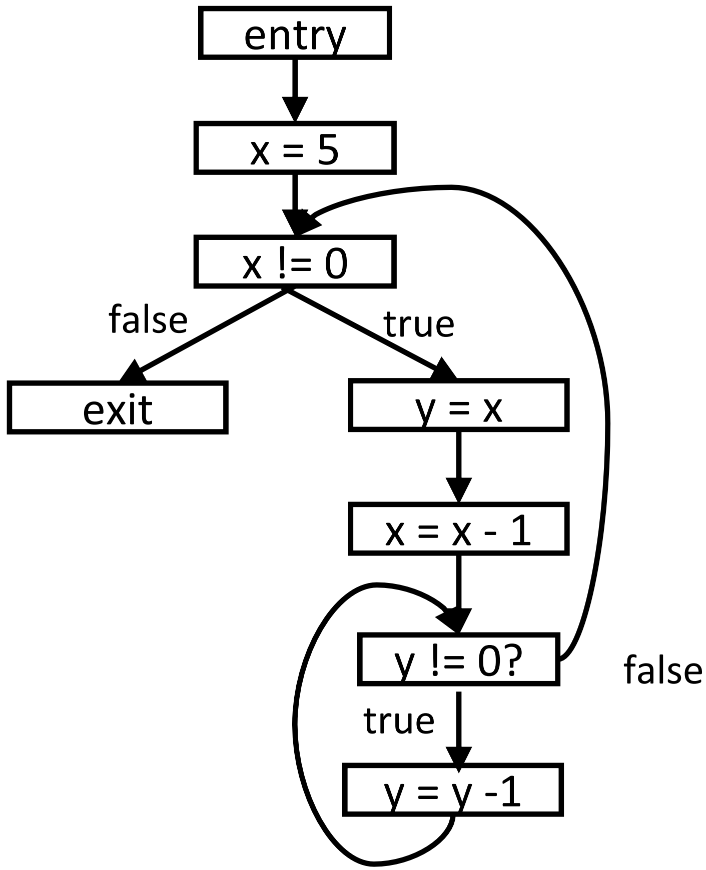
(integer constant) n

Control-Flow Graphs

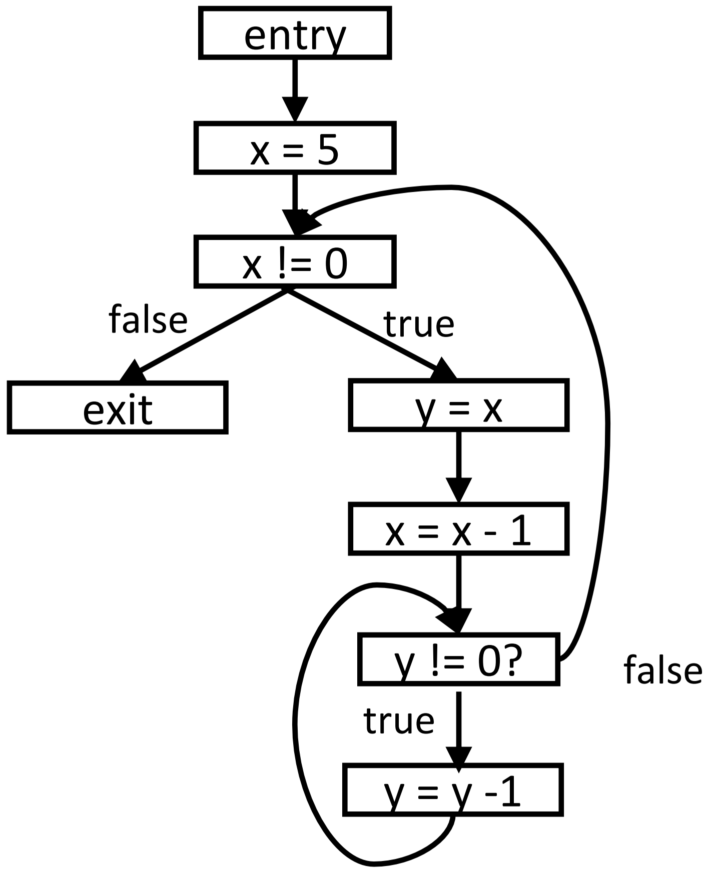



```
x = 5;
y = 1;
while (x != 1) {
  y = x * y;
  x = x - 1
}
```

QUIZ: Control-Flow Graphs



QUIZ: Control-Flow Graphs



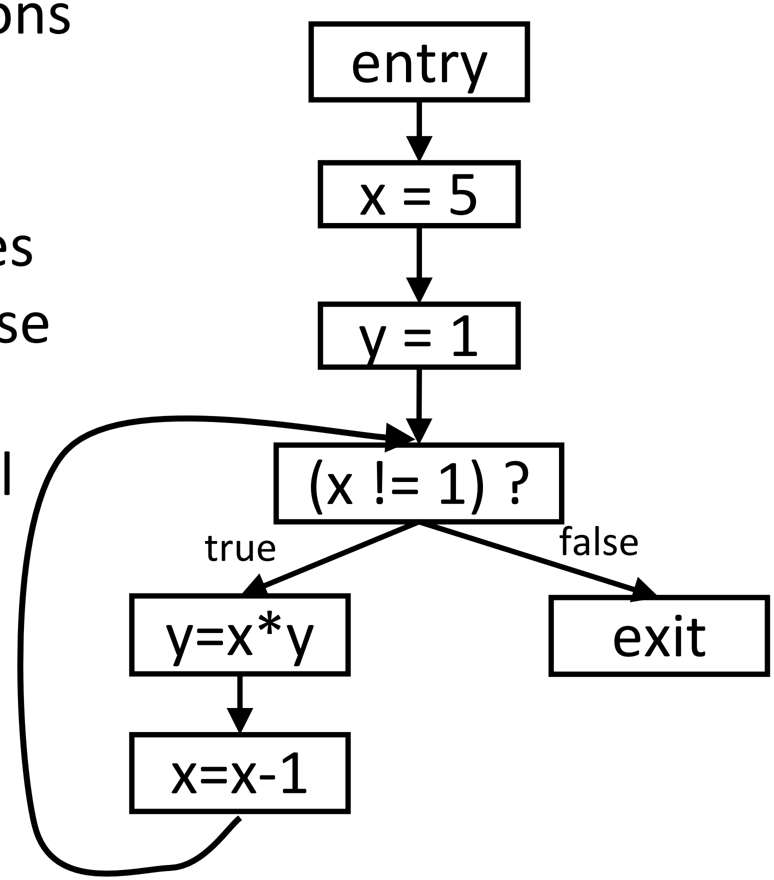

```
x = 5;
while (x != 0) {
  y = x;
  x = x - 1;
  while (y != 0) {
    y = y - 1
  }
}
```

Soundness, Completeness, Termination

- Impossible for analysis to achieve all three together
- Dataflow analysis sacrifices completeness
- Sound: Will report all facts that could occur in actual runs
- Incomplete: May report additional facts that can’t occur in actual runs

Abstracting Control-Flow Conditions

- Abstracts away control-flow conditions with non-deterministic choice (*)
- Non-deterministic choice => assumes condition can evaluate to true or false
- Considers all paths possible in actual runs (sound), and maybe paths that are never possible (incomplete).



Applications of Dataflow Analysis

Reaching Definitions Analysis

- Find usage of uninitialized variables

Available Expressions Analysis

- Avoid recomputing expressions

Very Busy Expressions Analysis

- Reduce code size

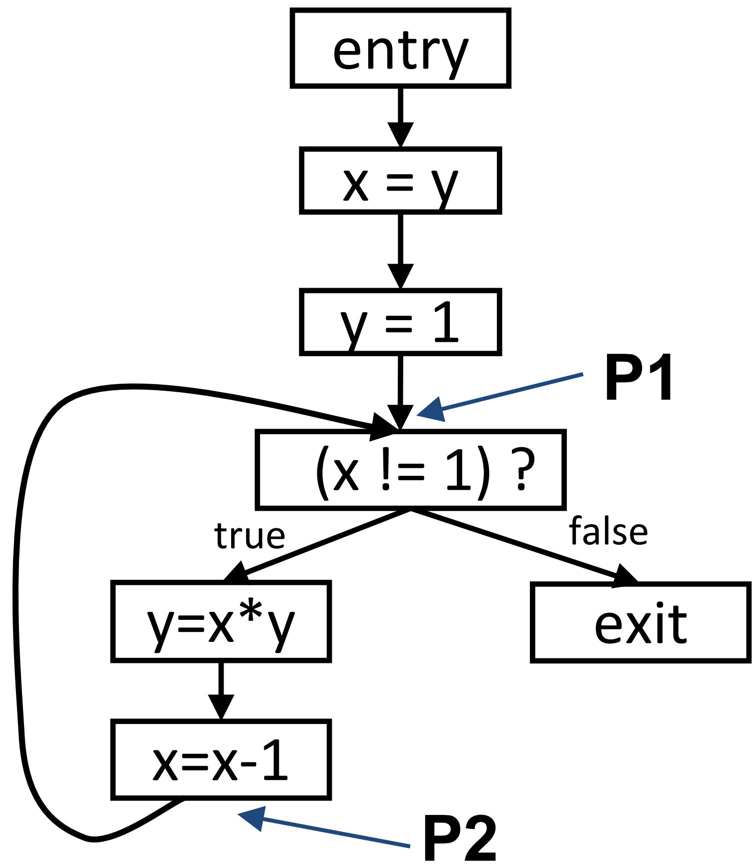
Live Variables Analysis

- Allocate registers efficiently

Reaching Definitions Analysis

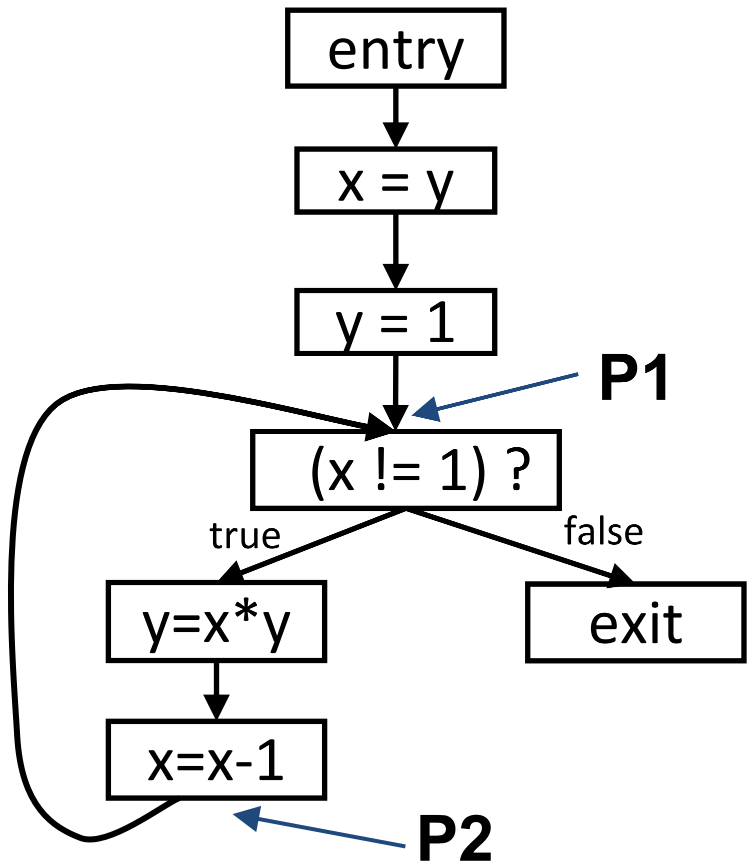
Goal: Determine, for each program point, which assignments have been made and not overwritten, when execution reaches that point along some path

- "Assignment" == "Definition"



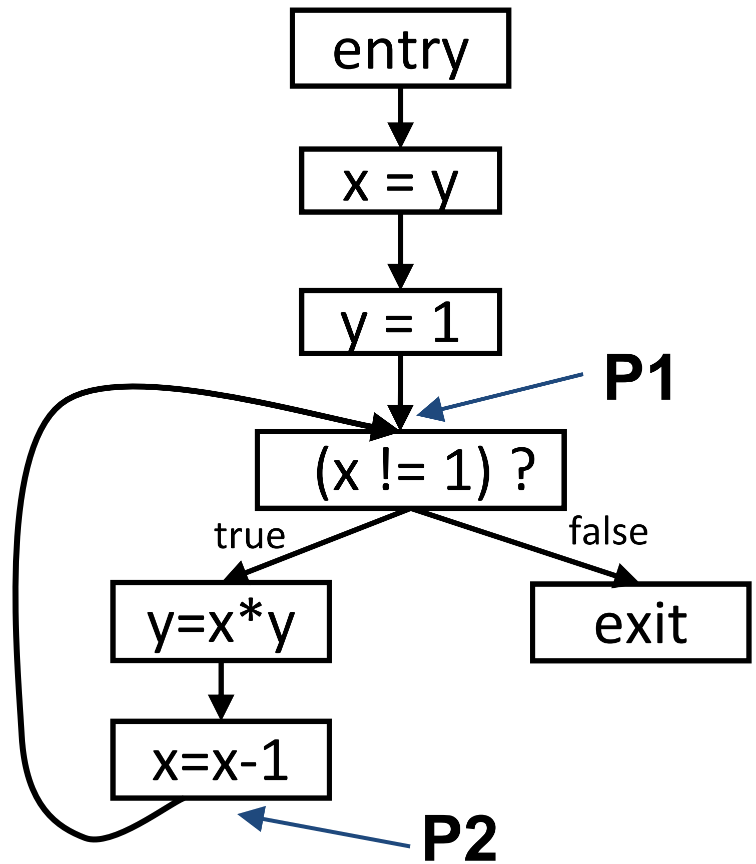
QUIZ: Reaching Definitions Analysis

1. The assignment y = 1 reaches P1 ☐
2. The assignment y = 1 reaches P2 ☐
3. The assignment y = x * y reaches P1 ☐



QUIZ: Reaching Definitions Analysis

1. The assignment y = 1 reaches P1 ■
2. The assignment y = 1 reaches P2 □
3. The assignment y = x * y reaches P1 ■

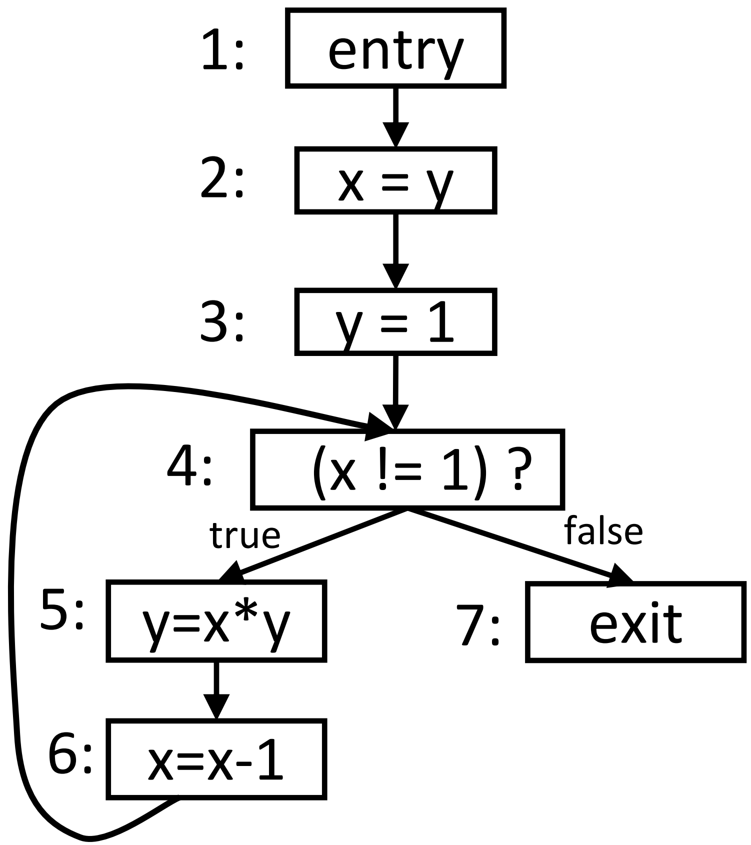


Result of Dataflow Analysis (Informally)

- Set of facts at each program point
- For reaching definitions analysis, fact is a pair of the form:

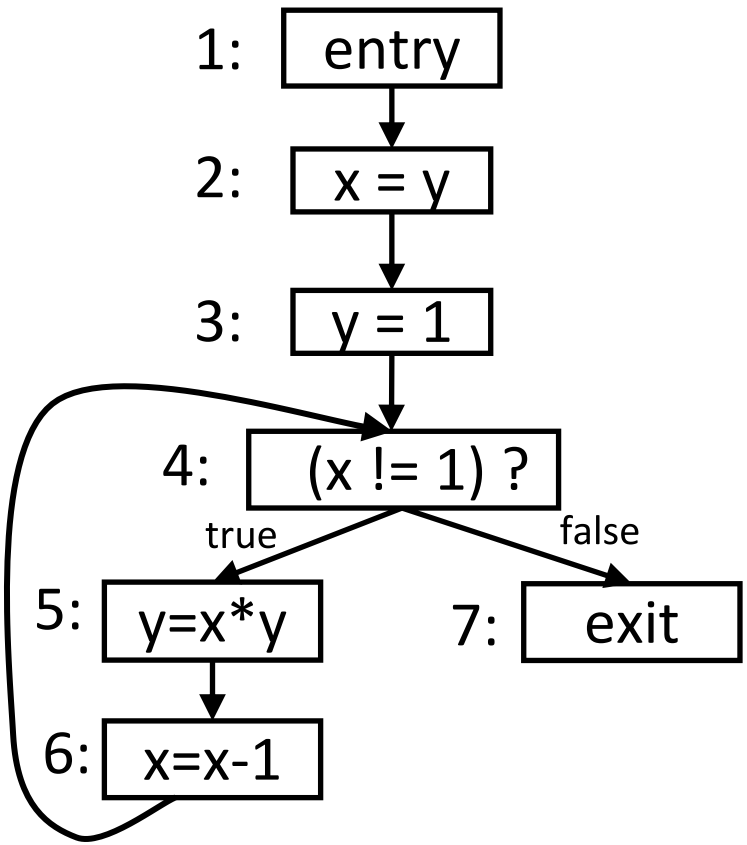
  <defined variable name, defining node label>

- Examples: <x,2> , <y,5>



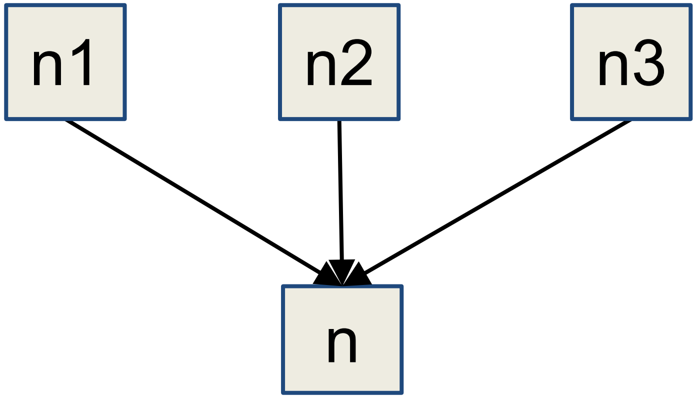
Result of Dataflow Analysis (Formally)

- Give distinct label n to each node
- IN[n] = set of facts at entry of node n
- OUT[n] = set of facts at exit of node n
- Dataflow analysis computes IN[n] and OUT[n] for each node
- Repeat two operations until IN[n] and OUT[n] stop changing
  - Called "saturated" or "fixed point"



Reaching Definitions Analysis: Operation #1

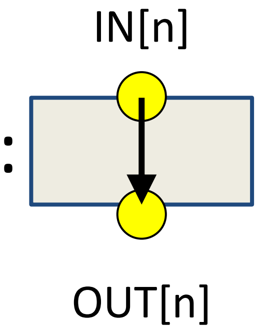
$$IN[n] = \bigcup_{n' \in \text{predecessors}(n)} OUT[n']$$



IN[n] = OUT[n1] U OUT[n2] U OUT[n3]

Reaching Definitions Analysis: Operation #2

OUT[n] = (IN[n] - KILL[n]) ∪ GEN[n]

n: 

n: | b ? |

GEN[n] = Ø   KILL[n] = Ø

n: | x = a |

GEN[n] = { <x, n> }

KILL[n] = { <x, m> : m != n }

Overall Algorithm: Chaotic Iteration

for (each node n):

  IN[n] = OUT[n] = Ø

OUT[entry] = { <v, ?> : v is a program variable }

repeat:

  for (each node n):

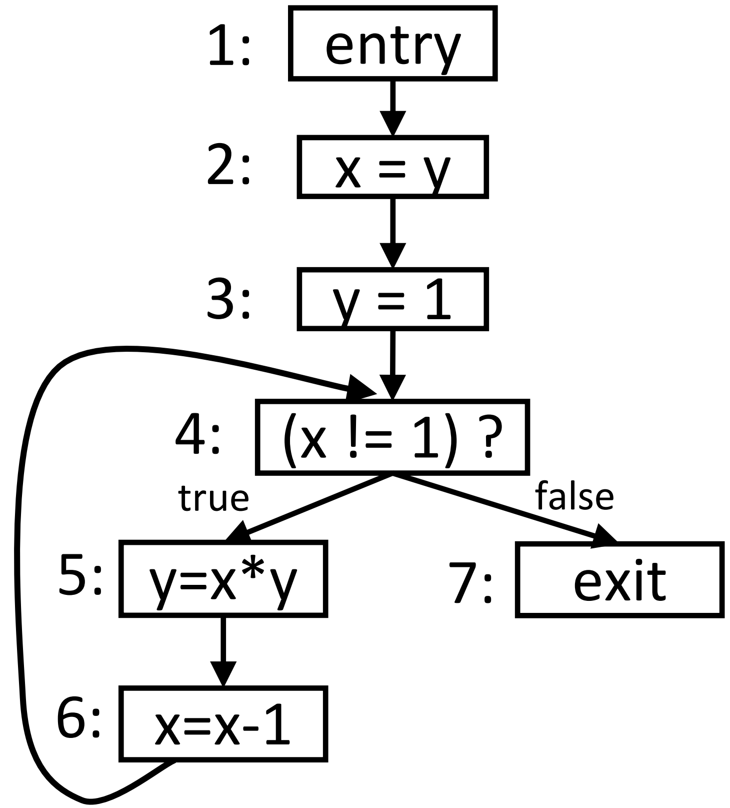
$$\mathrm{IN}[n] = \bigcup_{n' \in \mathrm{predecessors}(n)} \mathrm{OUT}[n']$$

$$\mathrm{OUT}[n] = (\mathrm{IN}[n] - \mathrm{KILL}[n]) \cup \mathrm{GEN}[n]$$

until IN[n] and OUT[n] stop changing for all n

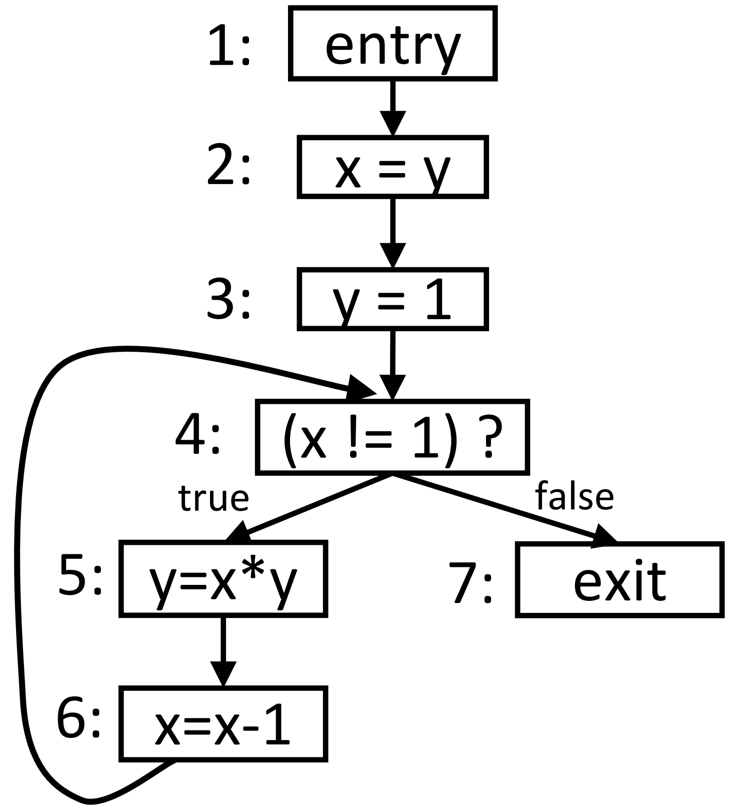
Reaching Definitions Analysis Example

n	IN[n]	OUT[n]
1	--	{<x,?>,<y,?>}
2	∅	∅
3	∅	∅
4	∅	∅
5	∅	∅
6	∅	∅
7	∅	--



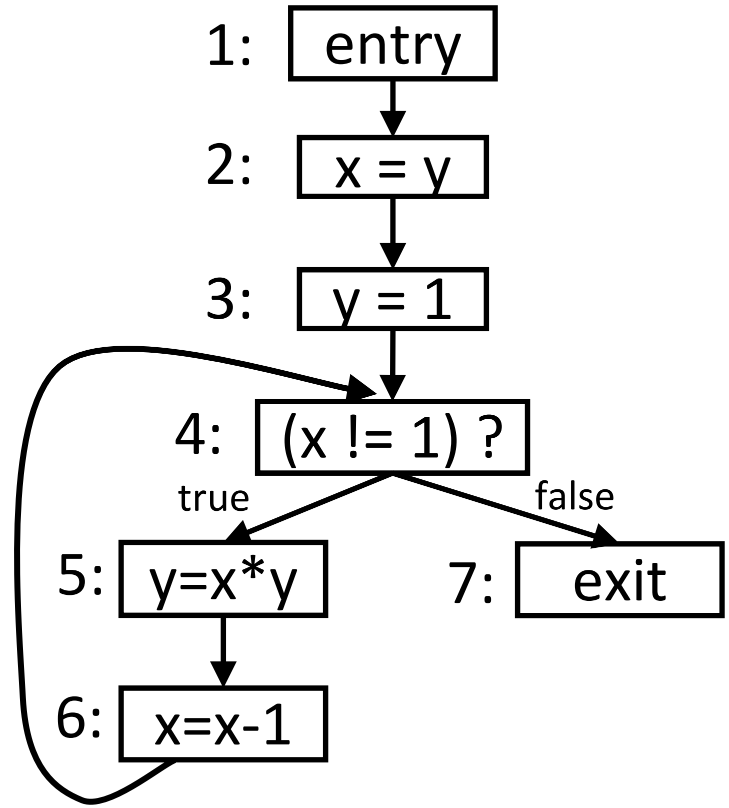
Reaching Definitions Analysis Example

n	IN[n]	OUT[n]
1	--	{<x,?>,<y,?>}
2	{<x,?>,<y,?>}	{<x,2>,<y,?>}
3	{<x,2>,<y,?>}	{<x,2>,<y,3>}
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	$\emptyset$	$\emptyset$
7	$\emptyset$	--



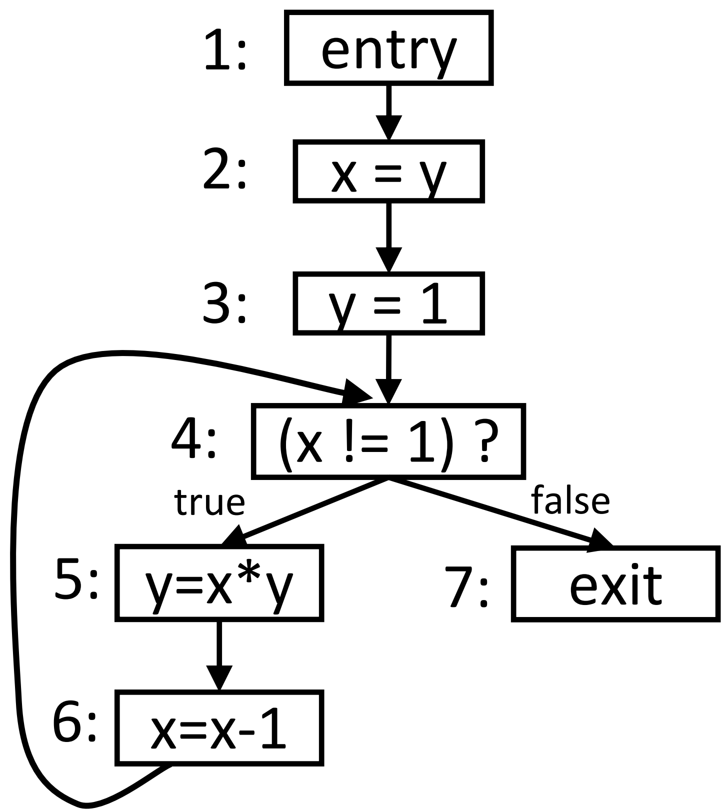
QUIZ: Reaching Definitions Analysis

n	IN[n]	OUT[n]
1	--	{<x,?>,<y,?>}
2	{<x,?>,<y,?>}	{<x,2>,<y,?>}
3	{<x,2>,<y,?>}	{<x,2>,<y,3>}
4		
5		
6		
7		--



QUIZ: Reaching Definitions Analysis

n	IN[n]	OUT[n]
1	--	{<x,?>,<y,?>}
2	{<x,?>,<y,?>}	{<x,2>,<y,?>}
3	{<x,2>,<y,?>}	{<x,2>,<y,3>}
4	{<x,2>,<y,3>,<y,5>,<x,6>}	{<x,2>,<y,3>,<y,5>,<x,6>}
5	{<x,2>,<y,3>,<y,5>,<x,6>}	{<x,2>,<y,5>,<x,6>}
6	{<x,2>,<y,5>,<x,6>}	{<y,5>,<x,6>}
7	{<x,2>,<y,3>,<y,5>,<x,6>}	--



Does It Always Terminate?

Chaotic Iteration algorithm always terminates

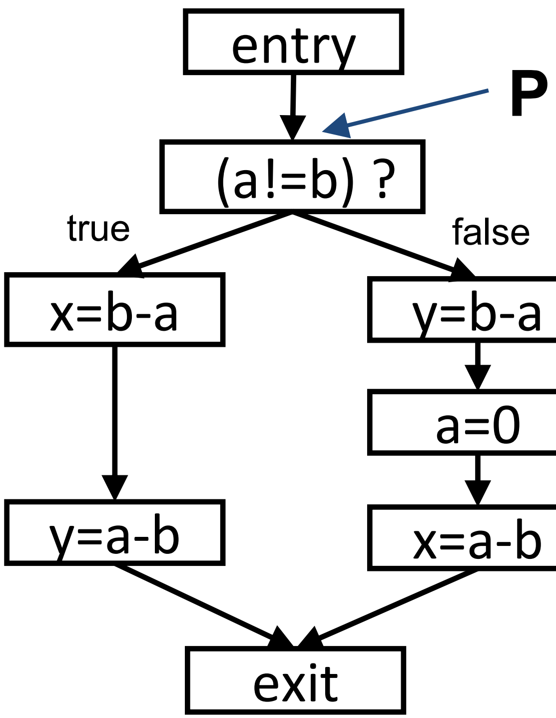
- The two operations of reaching definitions analysis are monotonic
  => IN and OUT sets never shrink, only grow
- Largest they can be is set of all definitions in program, which is finite
  => IN and OUT cannot grow forever

=> IN and OUT will stop changing after some iteration

Very Busy Expressions Analysis

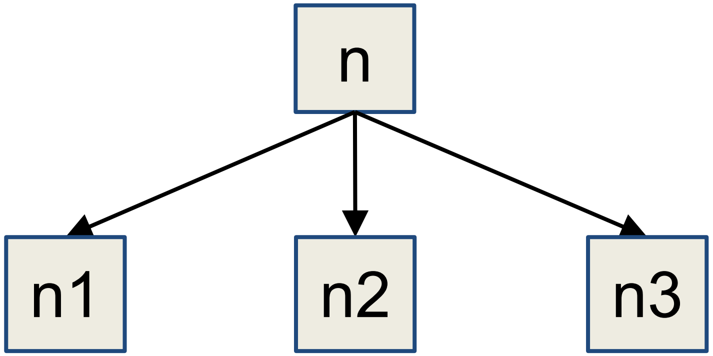
Goal: Determine very busy expressions at the exit from the point.

An expression is **very busy** if, no matter what path is taken, the expression is used before any of the variables occurring in it are redefined



Very Busy Expressions Analysis: Operation #1

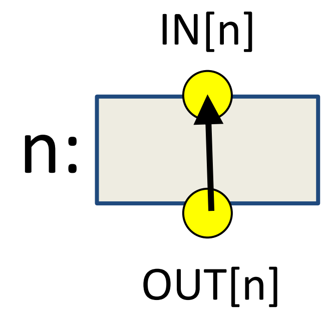
$$OUT[n] = \bigcap_{n' \in \text{successors}(n)} IN[n']$$



OUT[n] = IN[n1] ∩ IN[n2] ∩ IN[n3]

Very Busy Expressions Analysis: Operation #2

IN[n] = (OUT[n] - KILL[n]) ∪ GEN[n]



n: b ?

GEN[n] = Ø KILL[n] = Ø

n: x = a

GEN[n] = { a }

KILL[n] = { expr e : e contains x }

Overall Algorithm: Chaotic Iteration

for (each node n)
    IN[n] = OUT[n] = set of all exprs in program
IN[exit] = $\emptyset$
repeat:
    for (each node n)

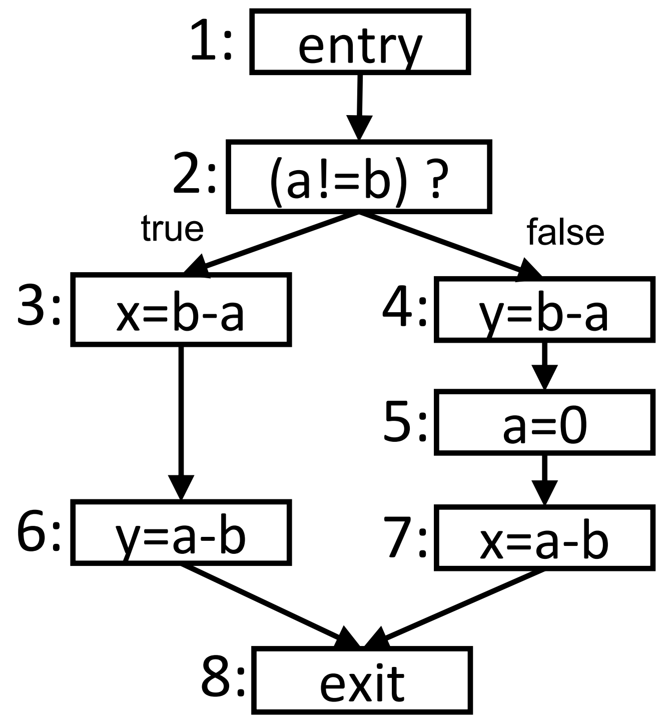
$$\mathrm{OUT}[n] = \bigcap_{n' \in \mathrm{successors}(n)} \mathrm{IN}[n']$$

$$\mathrm{IN}[n] = (\mathrm{OUT}[n] - \mathrm{KILL}[n]) \cup \mathrm{GEN}[n]$$

until IN[n] and OUT[n] stop changing for all n

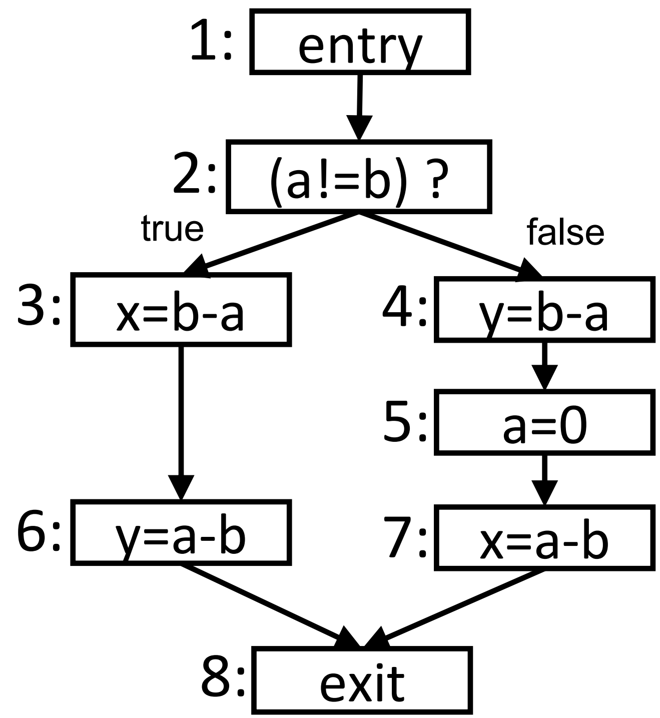
Very Busy Expressions Analysis Example

n	IN[n]	OUT[n]
1	--	{ b-a, a-b }
2	{ b-a, a-b }	{ b-a, a-b }
3	{ b-a, a-b }	{ b-a, a-b }
4	{ b-a, a-b }	{ b-a, a-b }
5	{ b-a, a-b }	{ b-a, a-b }
6	{ b-a, a-b }	{ b-a, a-b }
7	{ b-a, a-b }	{ b-a, a-b }
8	Ø	--



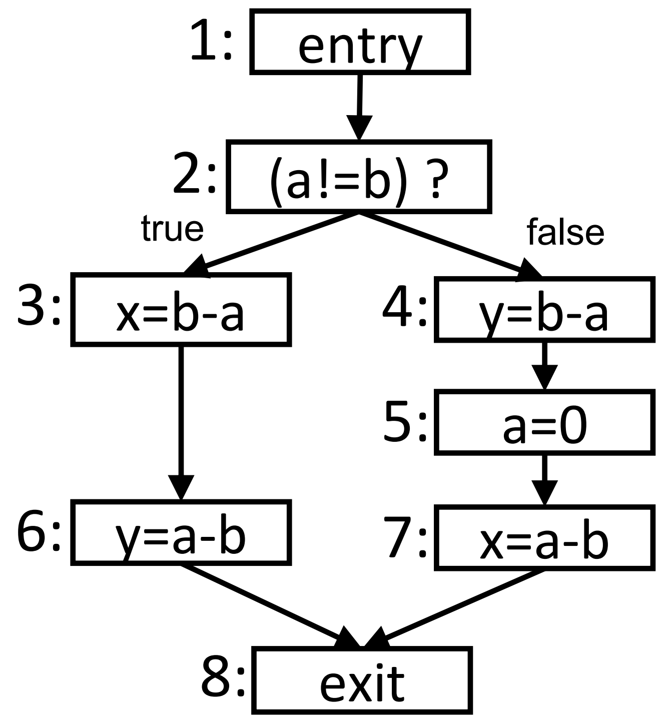
Very Busy Expressions Analysis Example

n	IN[n]	OUT[n]
1	--	{ b-a, a-b }
2	{ b-a, a-b }	{ b-a, a-b }
3	{ b-a, a-b }	{ b-a, a-b }
4	{ b-a, a-b }	{ b-a, a-b }
5	{ b-a, a-b }	{ b-a, a-b }
6	{ a-b }	Ø
7	{ a-b }	Ø
8	Ø	--



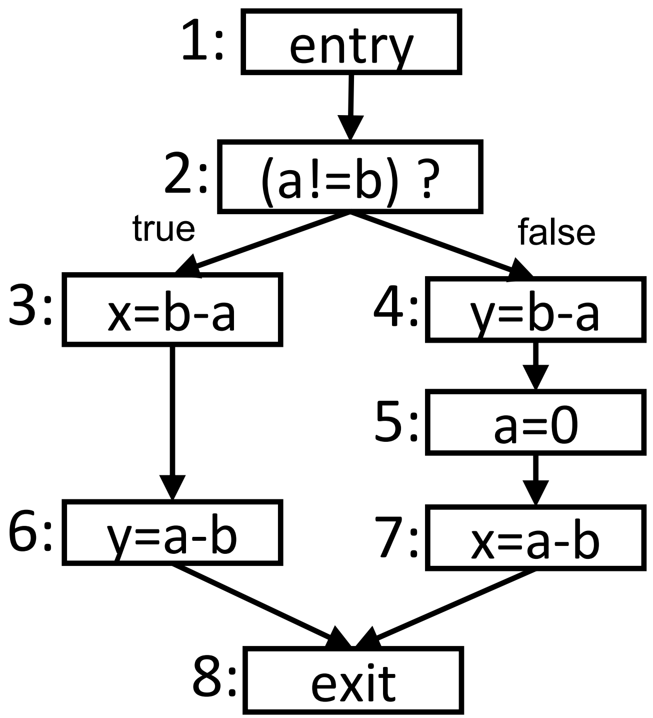
QUIZ: Very Busy Expressions Analysis

n	IN[n]	OUT[n]
1	--	
2		
3		
4		
5	Ø	{ a-b }
6	{ a-b }	Ø
7	{ a-b }	Ø
8	Ø	--



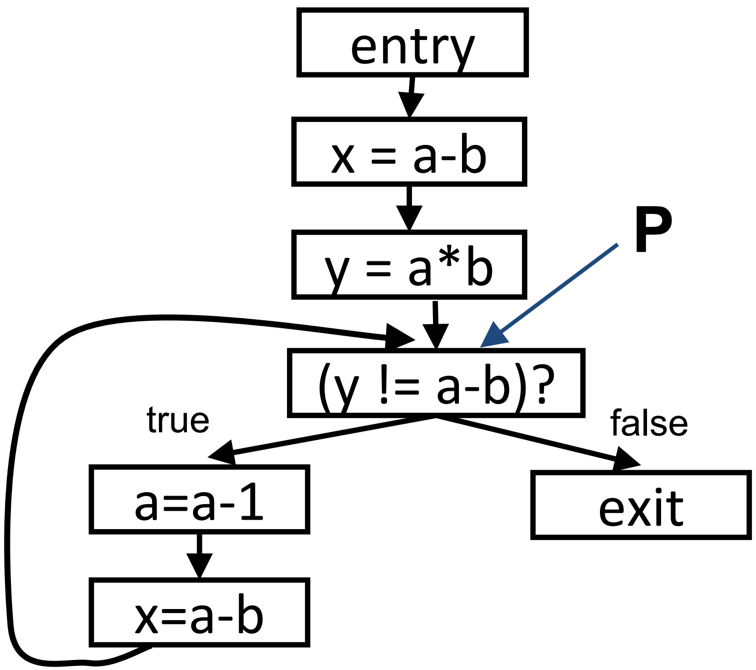
QUIZ: Very Busy Expressions Analysis

n	IN[n]	OUT[n]
1	--	{ b-a }
2	{ b-a }	{ b-a }
3	{ b-a, a-b }	{ a-b }
4	{ b-a }	Ø
5	Ø	{ a-b }
6	{ a-b }	Ø
7	{ a-b }	Ø
8	Ø	--



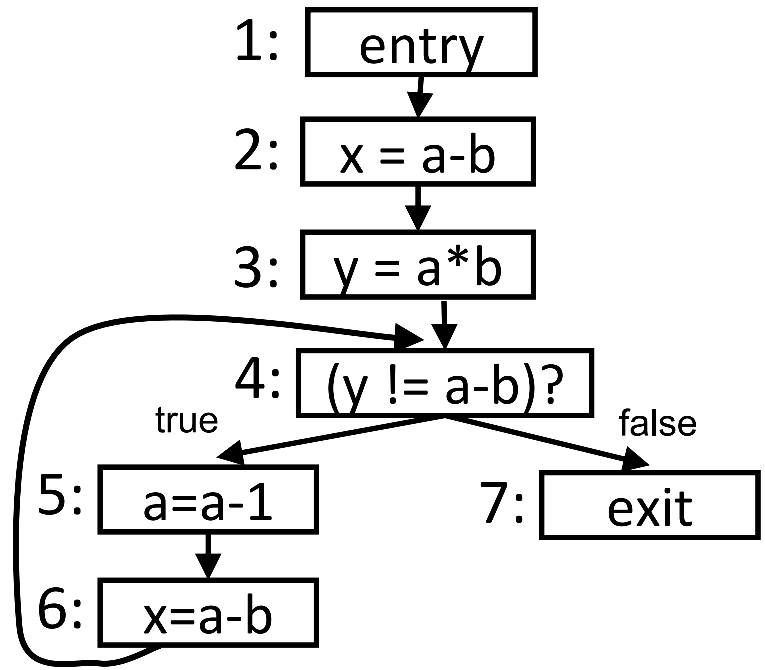
Available Expressions Analysis

Goal: Determine, for each program point, which expressions must already have been computed, and not later modified, on all paths to the program point.



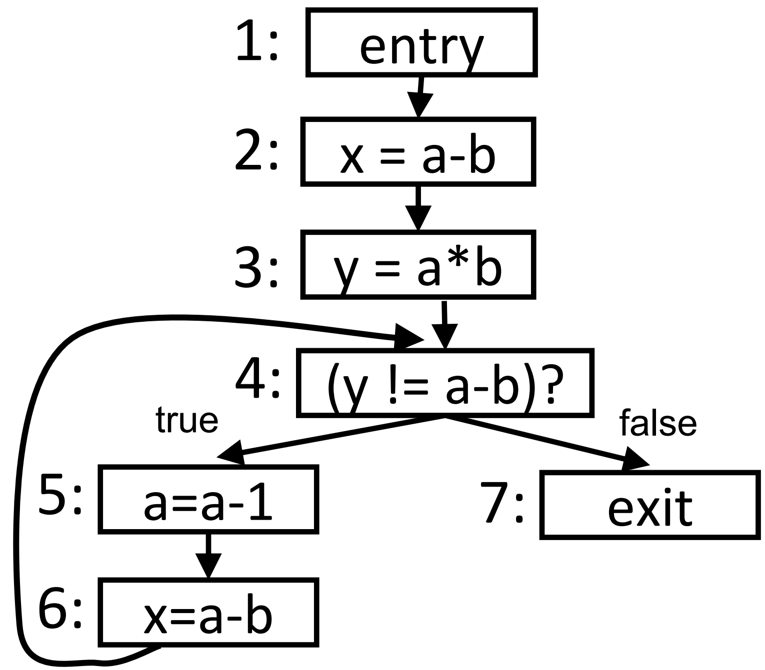
Available Expressions Analysis

n	IN[n]	OUT[n]
1	--	Ø
2	{a-b, a*b, a-1}	{a-b, a*b, a-1}
3	{a-b, a*b, a-1}	{a-b, a*b, a-1}
4	{a-b, a*b, a-1}	{a-b, a*b, a-1}
5	{a-b, a*b, a-1}	{a-b, a*b, a-1}
6	{a-b, a*b, a-1}	{a-b, a*b, a-1}
7	{a-b, a*b, a-1}	--



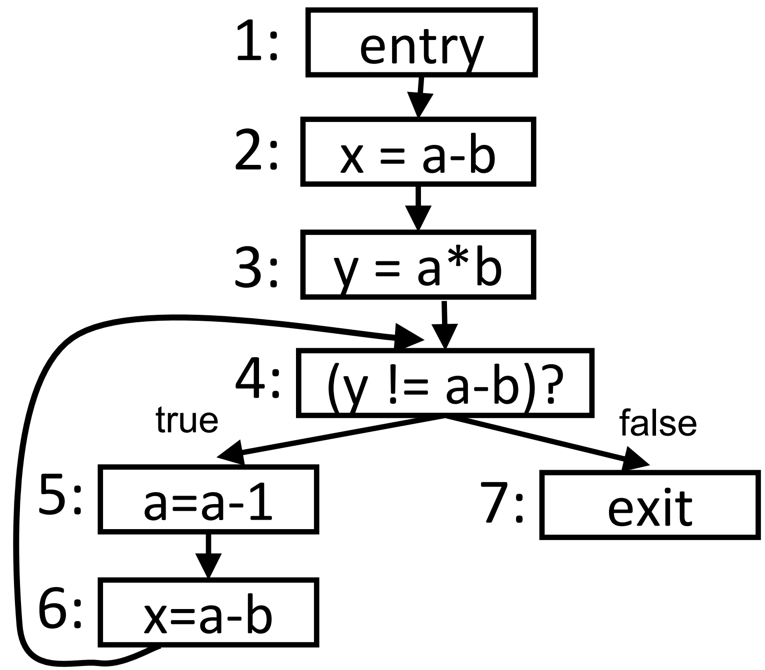
Available Expressions Analysis

n	IN[n]	OUT[n]
1	--	Ø
2	Ø	{a-b}
3	{a-b}	{a-b, a*b}
4	{a-b, a*b}	{a-b, a*b}
5	{a-b, a*b}	Ø
6	Ø	{a-b}
7	{a-b, a*b}	--



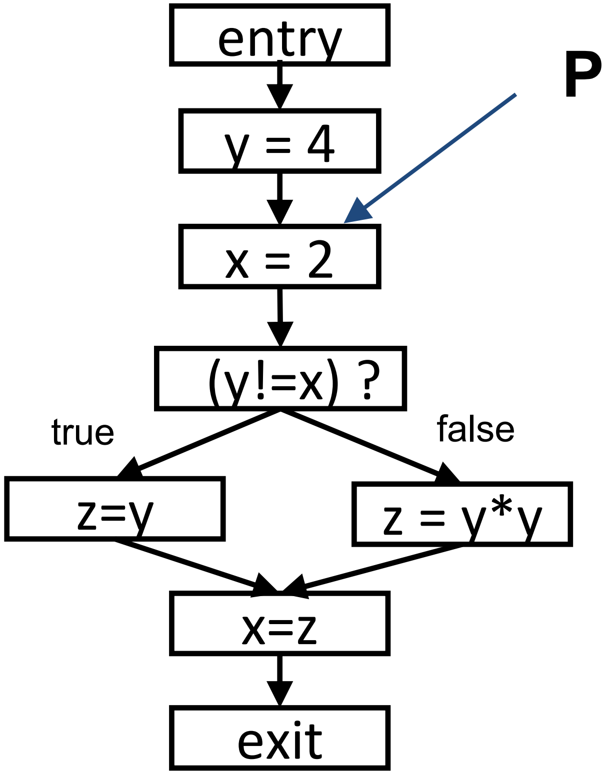
Available Expressions Analysis

n	IN[n]	OUT[n]
1	--	Ø
2	Ø	{a-b}
3	{a-b}	{a-b, a*b}
4	{a-b}	{a-b}
5	{a-b}	Ø
6	Ø	{a-b}
7	{a-b}	--



Live Variables Analysis

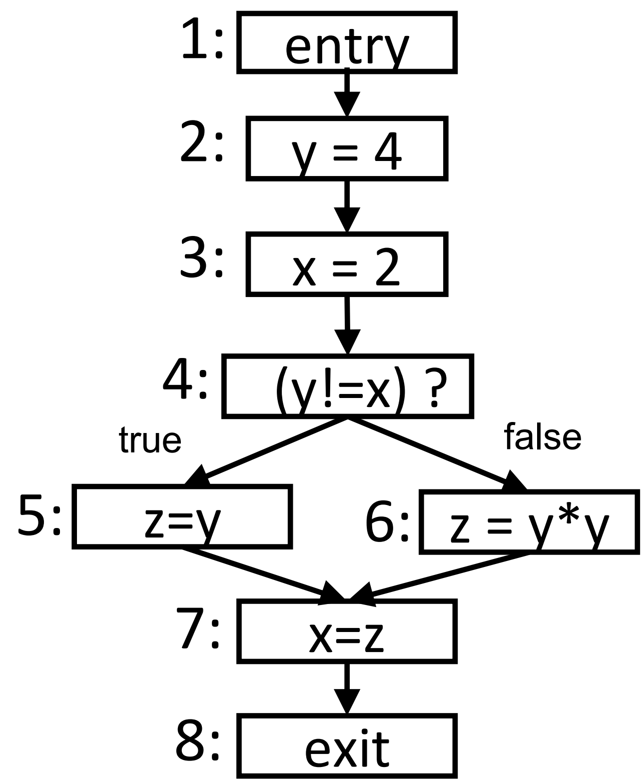
Goal: Determine for each program point which variables could be **live** at the point's exit



A variable is **live** if there is a path to a use of the variable that doesn't redefine the variable

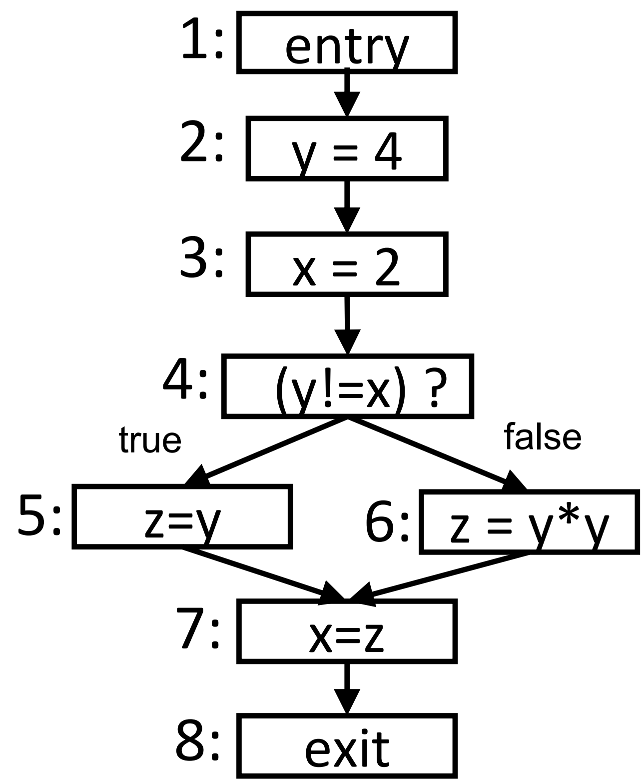
Live Variables Analysis

n	IN[n]	OUT[n]
1	--	Ø
2	Ø	Ø
3	Ø	Ø
4	Ø	Ø
5	Ø	Ø
6	Ø	Ø
7	Ø	Ø
8	Ø	--



Live Variables Analysis

n	IN[n]	OUT[n]
1	--	∅
2	∅	{ y }
3	{ y }	{ x, y }
4	{ x, y }	{ y }
5	{ y }	{ z }
6	{ y }	{ z }
7	{ z }	∅
8	∅	--



Overall Pattern of Dataflow Analysis

$$\boxed{\quad\quad}[n] = (\boxed{\quad\quad}[n] - KILL[n]) \cup GEN[n]$$

$$\boxed{\quad\quad}[n] = \boxed{\quad\quad}_{n' \in \boxed{\quad\quad}(n)} \boxed{\quad\quad}[n']$$

$\boxed{\quad\quad}$ $\boxed{\quad\quad}$ = IN or OUT

$\boxed{\quad\quad}$ = ∪ (may) or ∩ (must)

$\boxed{\quad\quad}$ = predecessors or successors

Reaching Definitions Analysis

$\boxed{\text{OUT}}[n] = (\boxed{\text{IN}}[n] - \text{KILL}[n]) \cup \text{GEN}[n]$

$$\boxed{\text{IN}}[n] = \underset{n' \in \boxed{\text{preds}}(n)}{\boxed{\cup}} \boxed{\text{OUT}}[n']$$

$\boxed{}$ $\boxed{}$ = IN or OUT

$\boxed{}$ = U (may) or ∩ (must)

$\boxed{}$ = predecessors or successors

Very Busy Expression Analysis

$$\boxed{\text{IN}}[n] = (\boxed{\text{OUT}}[n] - \text{KILL}[n]) \cup \text{GEN}[n]$$

$$\boxed{\text{OUT}}[n] = \underset{n' \in \boxed{\text{succs}}(n)}{\boxed{\cap}} \boxed{\text{IN}}[n']$$

$\boxed{}$ $\boxed{}$ = IN or OUT

$\boxed{}$ = U (may) or ∩ (must)

$\boxed{}$ = predecessors or successors

QUIZ: Available Expressions Analysis

$$\boxed{\quad}[n] = (\boxed{\quad}[n] - KILL[n]) \cup GEN[n]$$

$$\boxed{\quad}[n] = \boxed{\quad}_{n' \in \boxed{\quad}(n)} \boxed{\quad}[n']$$

$\boxed{\quad}$ $\boxed{\quad}$ = IN or OUT

$\boxed{\quad}$ = ∪ (may) or ∩ (must)

$\boxed{\quad}$ = predecessors or successors

QUIZ: Available Expressions Analysis

$$\boxed{\text{OUT}}[n] = (\boxed{\text{IN}}[n] - \text{KILL}[n]) \cup \text{GEN}[n]$$

$$\boxed{\text{IN}}[n] = \bigcap_{n' \in \boxed{\text{preds}}(n)} \boxed{\text{OUT}}[n']$$

= IN or OUT

= U (may) or ∩ (must)

= predecessors or successors

QUIZ: Live Variables Analysis

$$\boxed{\quad}[n] = (\boxed{\quad}[n] - \mathrm{KILL}[n]) \cup \mathrm{GEN}[n]$$

$$\boxed{\quad}[n] = \underset{n' \in \boxed{\quad}(n)}{\boxed{\quad}} \boxed{\quad}[n']$$

$\boxed{\quad}$ $\boxed{\quad}$ = IN or OUT

$\boxed{\quad}$ = ∪ (may) or ∩ (must)

$\boxed{\quad}$ = predecessors or successors

QUIZ: Live Variables Analysis

$\boxed{\text{IN}}\,[n] = (\boxed{\text{OUT}}\,[n] - \text{KILL}[n]) \cup \text{GEN}[n]$

$$\boxed{\text{OUT}}\,[n] = \underset{n' \in \boxed{\text{succs}}(n)}{\boxed{\cup}} \boxed{\text{IN}}\,[n']$$

$\boxed{\phantom{\text{IN}}}$ $\boxed{\phantom{\text{IN}}}$ = IN or OUT

$\boxed{\phantom{\text{IN}}}$ = U (may) or ∩ (must)

$\boxed{\phantom{\text{IN}}}$ = predecessors or successors

QUIZ: Classifying Dataflow Analyses

Match each analysis with its characteristics.

	May	Must
Forward		
Backward		

Very Busy Expressions

Reaching Definitions

Live Variables

Available Expressions

QUIZ: Classifying Dataflow Analyses

Match each analysis with its characteristics.

	May	Must
Forward	Reaching Definitions	Available Expressions
Backward	Live Variables	Very Busy Expressions

What Have We Learned?

- What is dataflow analysis
- Reasoning about flow of data using control-flow graphs
- Specifying dataflow analyses using local rules
- Chaotic iteration algorithm to compute global properties
- Four classical dataflow analyses
- Classification: forward vs. backward, may vs. must