

# CS443: Compiler Construction

Lecture 14: Dataflow Analysis

Stefan Muller

Based on material by Steve Zdancewic

# Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

# Generalized dataflow analysis: produce a set of “facts” in and out of each node

- Every statement (node):
  - Produces (**generates**) some set of facts
  - Eliminates (**kills**) some set of facts
- Constraints at each node computed from other nodes based on constraints (somewhat) specific to the analysis

# Dataflow analysis in 4 steps

1. Define facts, gen, kill
2. Define constraints
3. Convert constraints to equations
  - Sets should increase or decrease monotonically
4. Initialize facts for each node
  - Initial value should be consistent with whether sets are increasing or decreasing

# Liveness analysis as a dataflow analysis (Steps 1-2)

- Facts: Live variables
  - $\text{gen}[n] = \text{use}[n]$
  - $\text{kill}[n] = \text{def}[n]$
- Constraints:
  - $\text{in}[n] \supseteq \text{gen}[n]$
  - $\text{out}[n] \supseteq \text{in}[n']$  if  $n' \in \text{succ}[n]$
  - $\text{in}[n] \supseteq \text{out}[n] / \text{kill}[n]$

# Liveness analysis as a dataflow analysis (Steps 3-4)

- Equations:

- $\text{out}[n] := \bigcup_{n' \in \text{succ}[n]} \text{in}[n']$
- $\text{in}[n] := \text{gen}[n] \cup (\text{out}[n] / \text{kill}[n])$

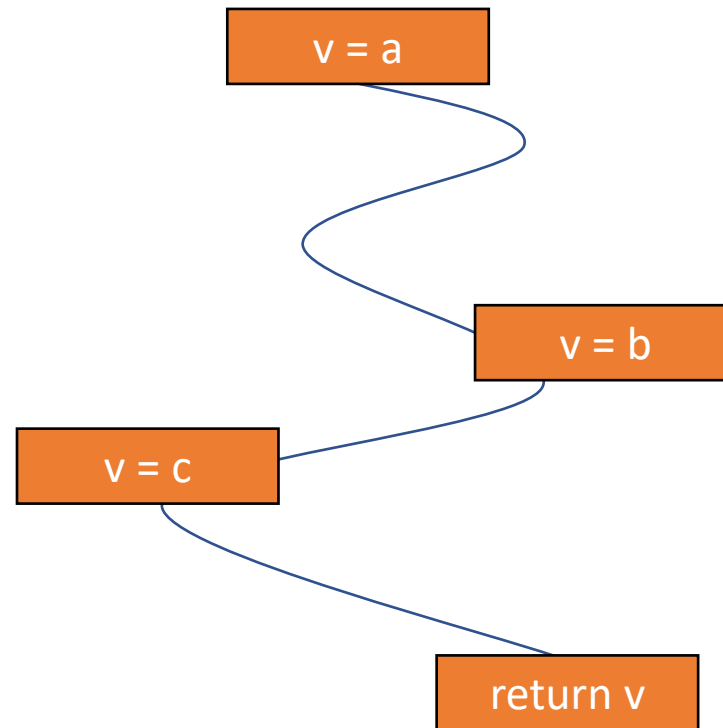
- Initial values:

- $\text{out}[n] := \emptyset$
- $\text{in}[n] := \emptyset$

# Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

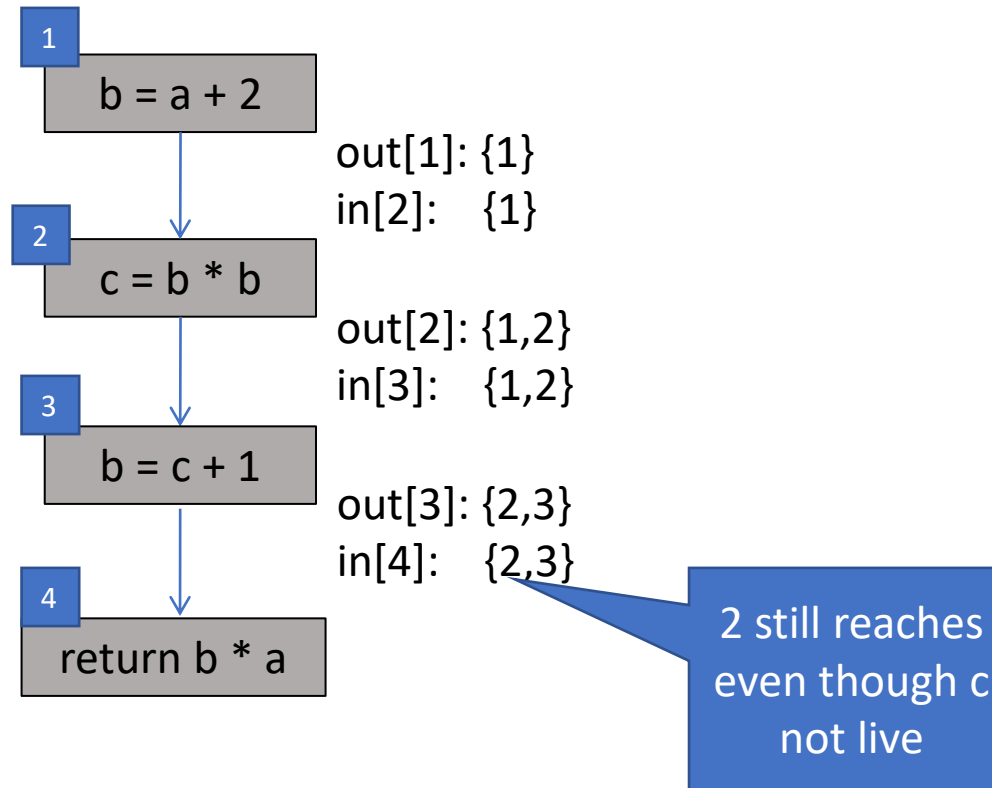
Recall from last time: a variable might be live for a long time, but w/ different definitions





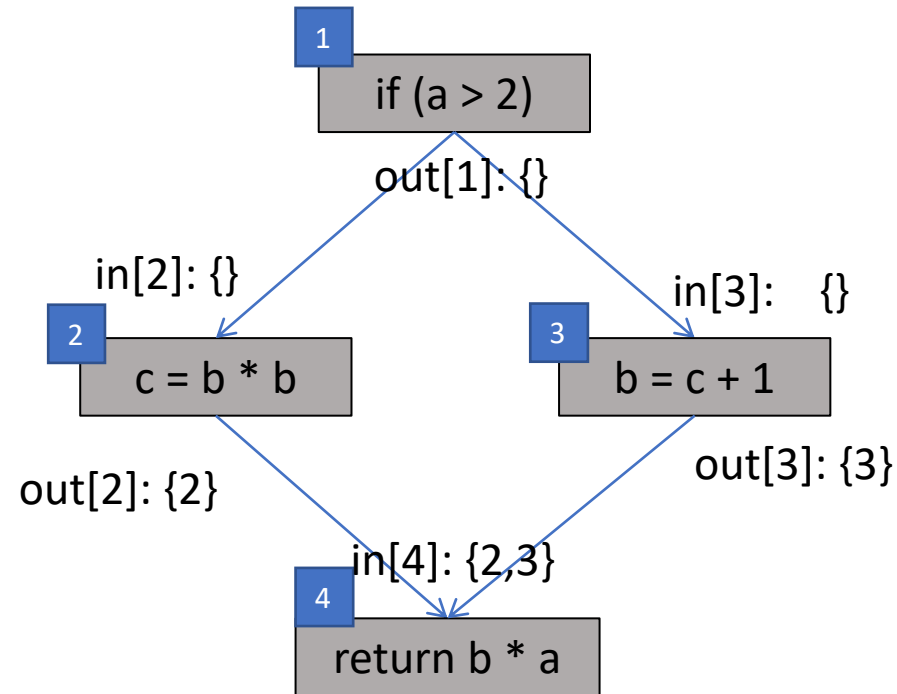
# Reaching definitions:

What *definitions* of a var might reach a node?



# Reaching definitions:

What *definitions* of a var might reach a node?



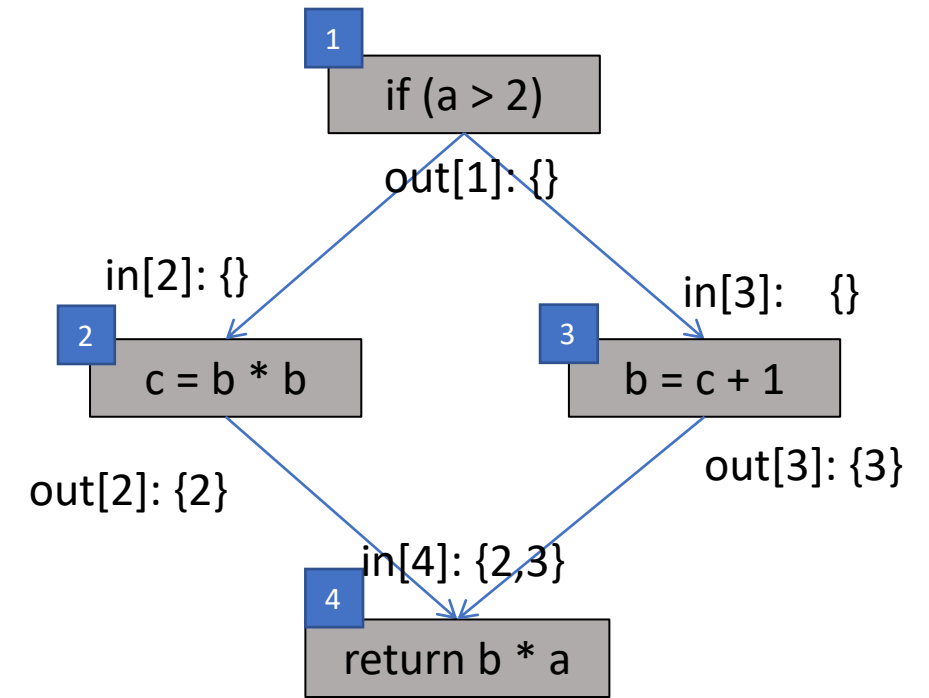
# Reaching definitions as a dataflow analysis (Step 1)

- Facts: set of nodes whose definition of a variable reaches  $n$
- Let  $\text{defs}[a]$  be the set of *nodes* that define the variable  $a$

$n$	$\text{gen}[n]$	$\text{kill}[n]$
$a = b \text{ op } c$	$\{n\}$	$\text{defs}[a] - \{n\}$
$a = \text{load } b$	$\{n\}$	$\text{defs}[a] - \{n\}$
$\text{store } b, a$	$\emptyset$	$\emptyset$
$a = f(b_1, \dots, b_n)$	$\{n\}$	$\text{defs}[a] - \{n\}$
$f(b_1, \dots, b_n)$	$\emptyset$	$\emptyset$
$\text{br } L$	$\emptyset$	$\emptyset$
$\text{br } a \text{ } L1 \text{ } L2$	$\emptyset$	$\emptyset$
$\text{return } a$	$\emptyset$	$\emptyset$

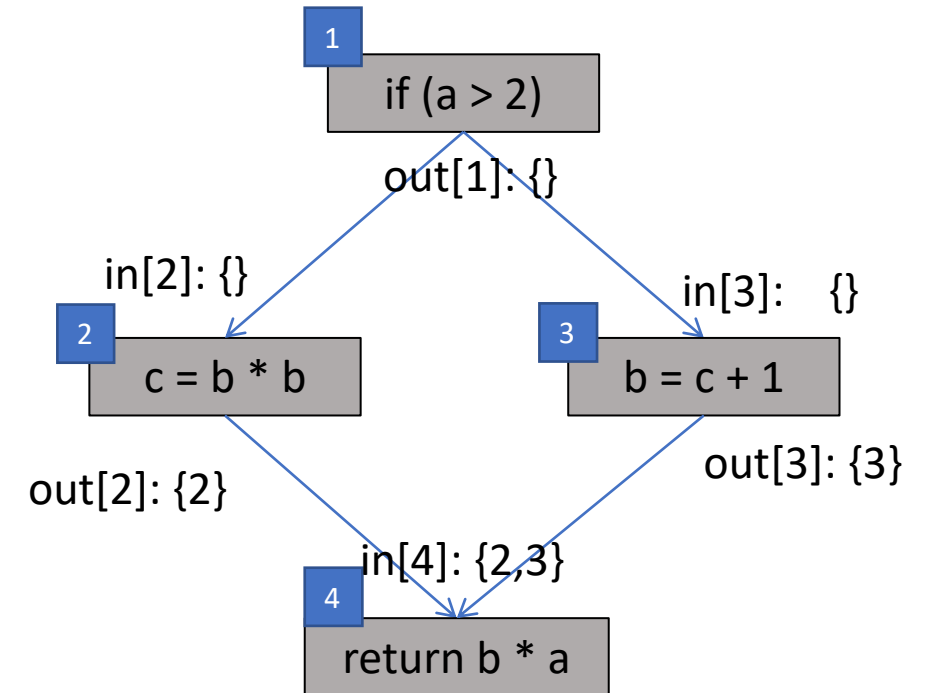
# Reaching definitions as a dataflow analysis (Step 2)

- $\text{out}[n] \supseteq \text{gen}[n]$
- $\text{in}[n] \supseteq \text{out}[n']$  if  $n'$  is in  $\text{pred}[n]$
- $\text{out}[n] \cup \text{kill}[n] \supseteq \text{in}[n]$ 
  - Equivalently:  $\text{out}[n] \supseteq \text{in}[n] / \text{kill}[n]$



# Reaching definitions as a dataflow analysis (Steps 3-4)

- $\text{in}[n] := \bigcup_{n' \in \text{pred}[n]} \text{out}[n']$
- $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$
- Algorithm: initialize  $\text{in}[n]$  and  $\text{out}[n]$  to  $\emptyset$



# Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

# When is this optimization safe?

- $a = x + 1$   
...  
 $b = x + 1$



- $a = x + 1$   
...  
 $b = a$



As long as  $a$   
isn't  
redefined  
here

- Available expressions: nodes whose definitions are “available”

# Available $\neq$ Live

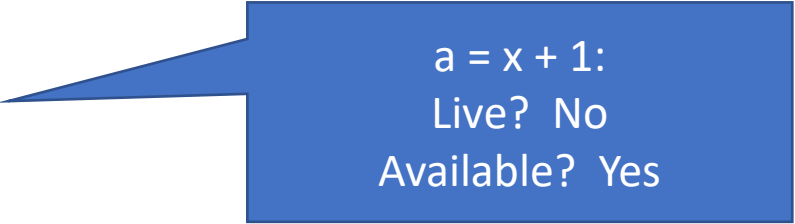
a = x + 1

c = a

b = x + 1

d = b \* 2

return d - c



a = x + 1:  
Live? No  
Available? Yes



# Available expressions as a dataflow analysis (Step 1)

n:	gen[n]	kill[n]
a = b op c	{n}	uses[a]
a = load b	{n}	uses[a]
store b, a	$\emptyset$	uses[*x] (for all x that may equal a)
br L	$\emptyset$	$\emptyset$
br a L1 L2	$\emptyset$	$\emptyset$
a = f(b <sub>1</sub> ,...,b <sub>n</sub> )	$\emptyset$	uses[a] ∪ uses[*x] (for all x)
f(b <sub>1</sub> ,...,b <sub>n</sub> )	$\emptyset$	uses[*x] (for all x)
return a	$\emptyset$	$\emptyset$

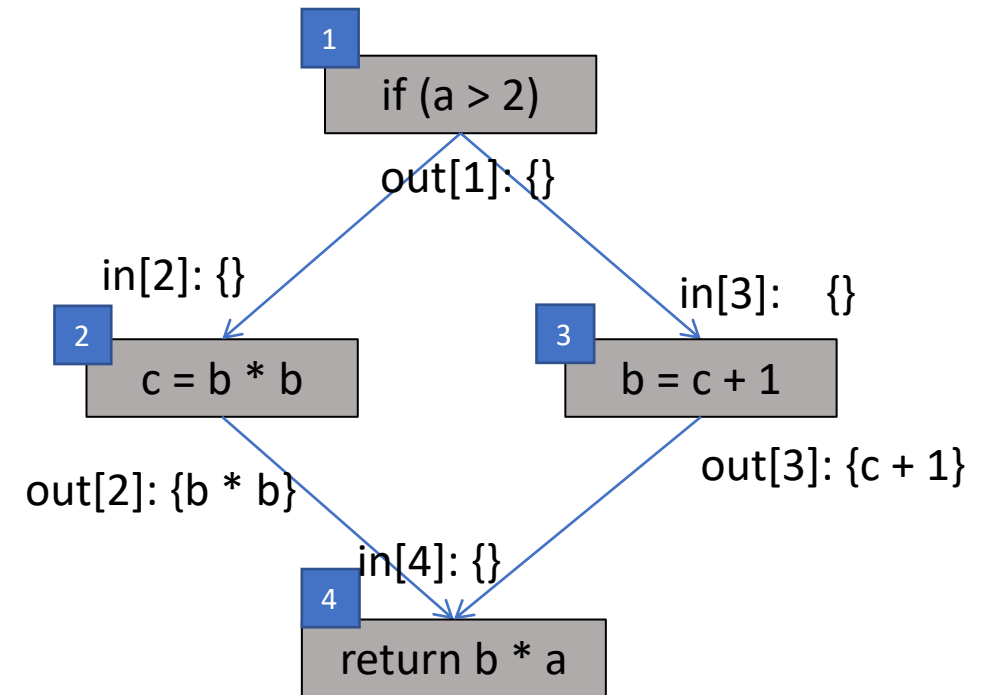
Memory at loc. x

Alias analysis!

(assuming impure functions)

# Available expressions as a dataflow analysis (Steps 2-3)

- $\text{out}[n] \supseteq \text{gen}[n]$
- $\text{in}[n] \subseteq \text{out}[n']$  if  $n'$  is in  $\text{pred}[n]$
- $\text{out}[n] \cup \text{kill}[n] \supseteq \text{in}[n]$ 
  - Equivalently:  $\text{out}[n] \supseteq \text{in}[n] / \text{kill}[n]$
- $\text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n']$
- $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$



# Available expressions as a dataflow analysis (Steps 3-4)

- $\text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n']$
- $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$
- Initialize  $\text{in}[n]$  and  $\text{out}[n]$  to {set of all nodes}
  - Iterate the update equations until a fixed point is reached
- The algorithm terminates because  $\text{in}[n]$  and  $\text{out}[n]$  *decrease monotonically*
  - At most to a minimum of the empty set
- The algorithm is precise because it finds the *largest* sets that satisfy the constraints.

# Contrasting RD/AE

## Reaching Defs

$\text{in}[n] := \bigcup_{n' \in \text{pred}[n]} \text{out}[n']$   
 $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$

Which definitions *may* reach  $n$ ?

Initialize to  $\emptyset$

“May” analysis

## Available Expressions

$\text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n']$   
 $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$

Which expressions *must* reach  $n$ ?

Initialize to all expressions

“Must” analysis

# Contrasting RD/Liveness

## Reaching Defs

$$\text{in}[n] := \bigcup_{n' \in \text{pred}[n]} \text{out}[n']$$
$$\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$$

Propagate information *forward*

Forward analysis

## Liveness

$$\text{out}[n] := \bigcup_{n' \in \text{succ}[n]} \text{in}[n']$$
$$\text{in}[n] := \text{gen}[n] \cup (\text{out}[n] / \text{kill}[n])$$

Propagate information *backward*

Backward analysis

	Backward	Forward
May	Liveness: What variables <i>may</i> be needed from $n$ ?	Reaching Definitions: What definitions <i>may</i> reach $n$ ?
Must	Very Busy Expressions: What expressions <i>will</i> be defined on every path from $n$ ?	Available Expressions: What expressions <i>must</i> reach $n$ ?