CS323: Compilers

Spring 2023

Week 12: Dataflow Analysis

Useful optimizations

- Common subexpression elimination (global)
  - Need to know which expressions are available at a point
- Dead code elimination
  - Need to know if the effects of a piece of code are never needed, or if code cannot be reached
- Constant folding
  - Need to know if variable has a constant value
- So how do we get this information?

Slide Courtesy: Milind Kulkarni

Dataflow analysis

- Framework for doing compiler analyses to drive optimization
- Works across basic blocks
- Examples
  - Constant propagation: determine which variables are constant
  - Liveness analysis: determine which variables are live
  - Available expressions: determine which expressions have valid computed values
  - Reaching definitions: determine which definitions could "reach" a use

Slide Courtesy: Milind Kulkarni

Dataflow Analysis - Common Traits

Common requirement among global optimizations:

- Know a particular **property X** at a *program point*

  (There is a program point one before a statement and one after a statement)

  - Say that property X definitely holds.

    OR

  - Don't know if property X holds or not (okay to be conservative)

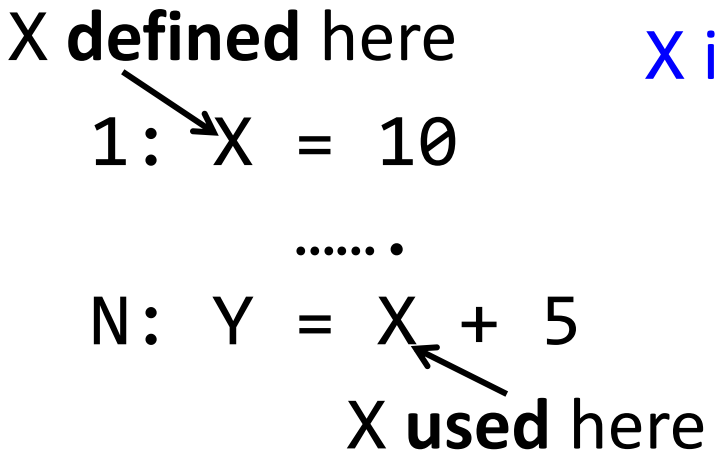
  This requires the knowledge of entire program

Dataflow analysis

- Framework for doing compiler analyses to drive optimization
- Works across basic blocks
- Examples
  - Constant propagation: determine which variables are constant
  - Liveness analysis: determine which variables are live
  - Available expressions: determine which expressions have valid computed values
  - Reaching definitions: determine which definitions could "reach" a use

Slide Courtesy: Milind Kulkarni

Liveness – Recap..



X is live at 1

..used in future

- A variable X is live at statement S if:
  - There is a statement S' that uses X
  - There is a path from S to S'
  - There are no intervening definitions of X

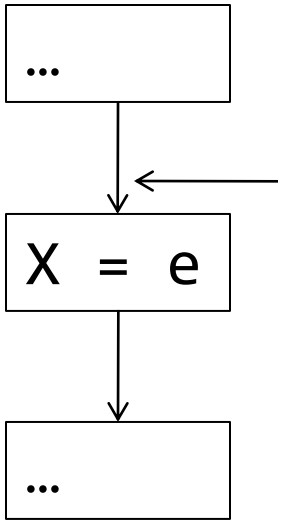
Liveness – Recap..

```
1: X = 10
2: X = Y + 2
   ….
N: Y = X + 5
```

X is dead at 1

- A variable X is dead at statement S if it is not live at S
  - What about …; `X = X + 1`?

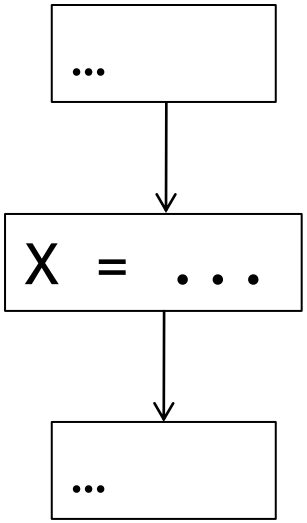
Liveness in a CFG



Given that e does not use X, X is *definitely dead* here (i.e. before the statement).

- Define a set `LiveIn(b)`, where b is a basic block, as: the set of all variables live at the entrance of a basic block

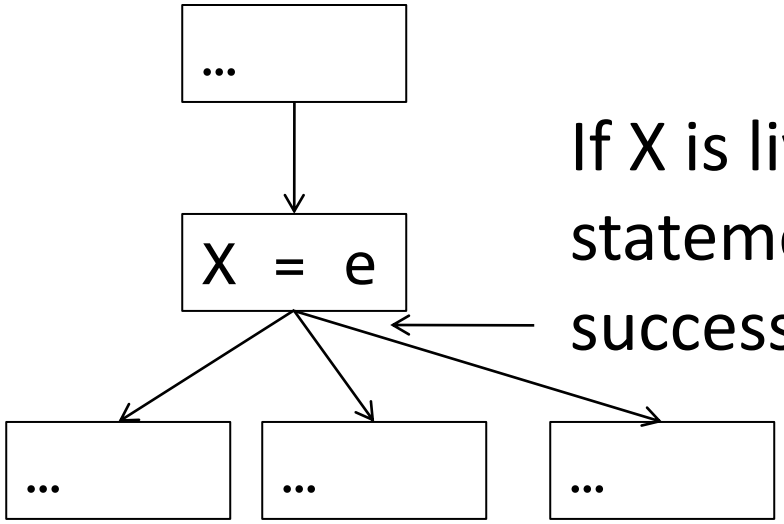
Liveness in a CFG



X is defined here

- Define a set `Def(b)`, where b is a basic block, as: the set of all variables that are defined in b

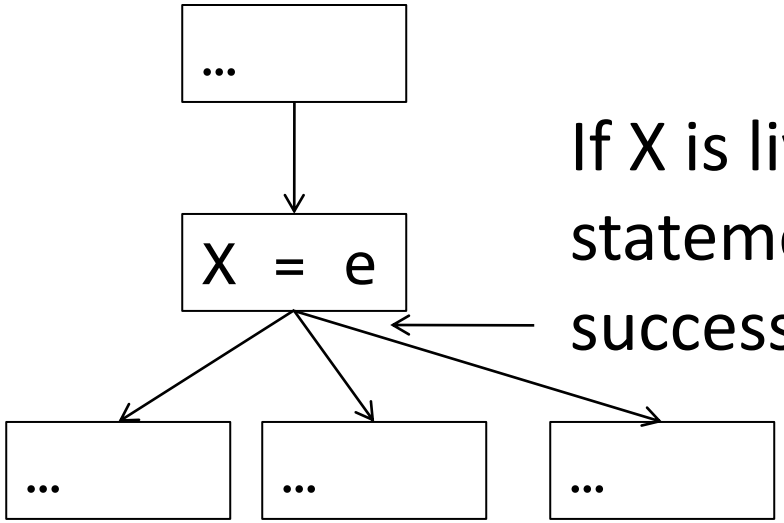
Liveness in a CFG



If X is live here (i.e. after the statement), X is used in *some* successor

- Define a set `LiveOut(b)`, where b is a basic block, as: the set of all variables live at the exit of a basic block

Liveness in a CFG

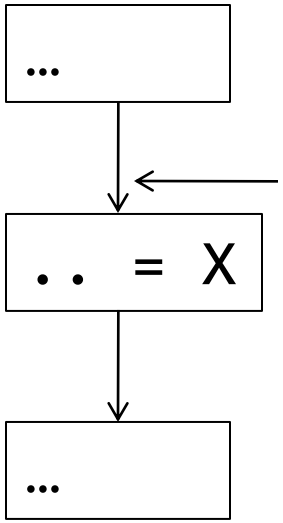


If X is live here (i.e. after the statement), X is used in *some* successor

- If $S(b)$ is the set of all successors of b, then

$$\text{LiveOut}(b) = \bigcup_{i \in S(b)} \text{LiveIn}(i)$$

Liveness in a CFG

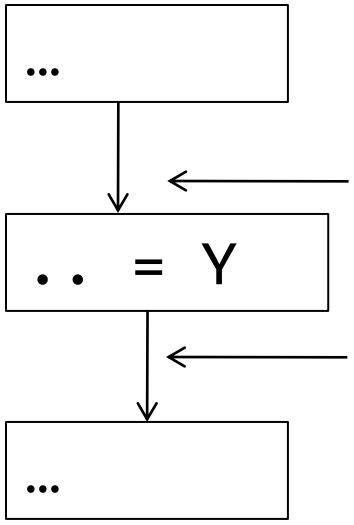


X *must be* live here (i.e. before the statement)

• Define a set `LiveUse(b)`, where b is a basic block, as the set of all variables that are used before they are defined within block b. `LiveIn(b)` ⊇ `LiveUse(b)`

Liveness in a CFG - Observation

- If a node neither uses nor defines X, the liveness property remains the same before and after executing the node



X not live here / X is live here

If X is not live here / X is live here

Liveness in a CFG

- If a variable is live on exit from b, it is either defined in b or live on entrance to b

$$\text{LiveIn}(b) \supseteq \text{LiveOut}(b) - \text{Def}(b)$$

- Under what scenarios can a variable be live at the entrance of a basic block?

Liveness in a CFG

- If a variable is live on exit from b, it is either defined in b or live on entrance to b

$$\text{LiveIn}(b) \supseteq \text{LiveOut}(b) - \text{Def}(b)$$

- Under what scenarios can a variable be live at the entrance of a basic block?
  - Either the variable is used in the basic block

Liveness in a CFG

- If a variable is live on exit from b, it is either defined in b or live on entrance to b

$$\text{LiveIn}(b) \supseteq \text{LiveOut}(b) - \text{Def}(b)$$

- Under what scenarios can a variable be live at the entrance of a basic block?
  - Either the variable is used in the basic block
  - OR the variable is live at exit and not defined within the block

Liveness in a CFG

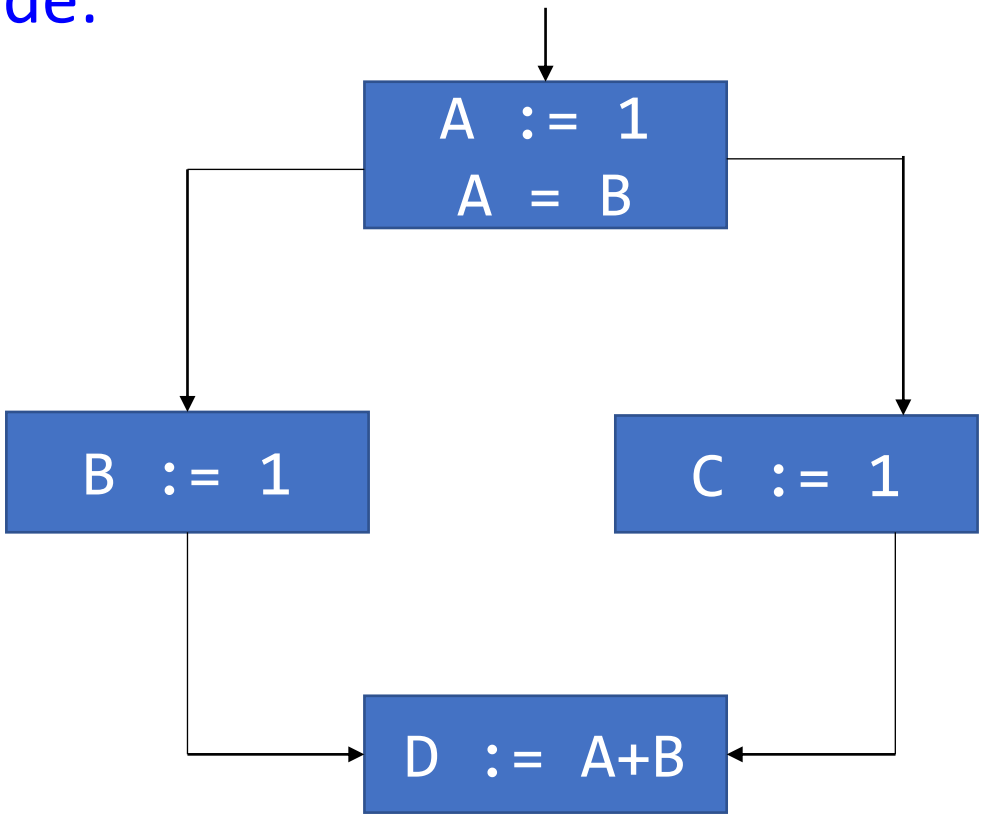
- Under what scenarios can a variable be live at the entrance of a basic block?
  - Either the variable is used in the basic block
  - OR the variable is live at exit and not defined within the block

```
LiveIn(b) = LiveUse(b) U  (LiveOut(b) – Def(b))
```

Liveness in a CFG - Example

- Draw CFG for the code:

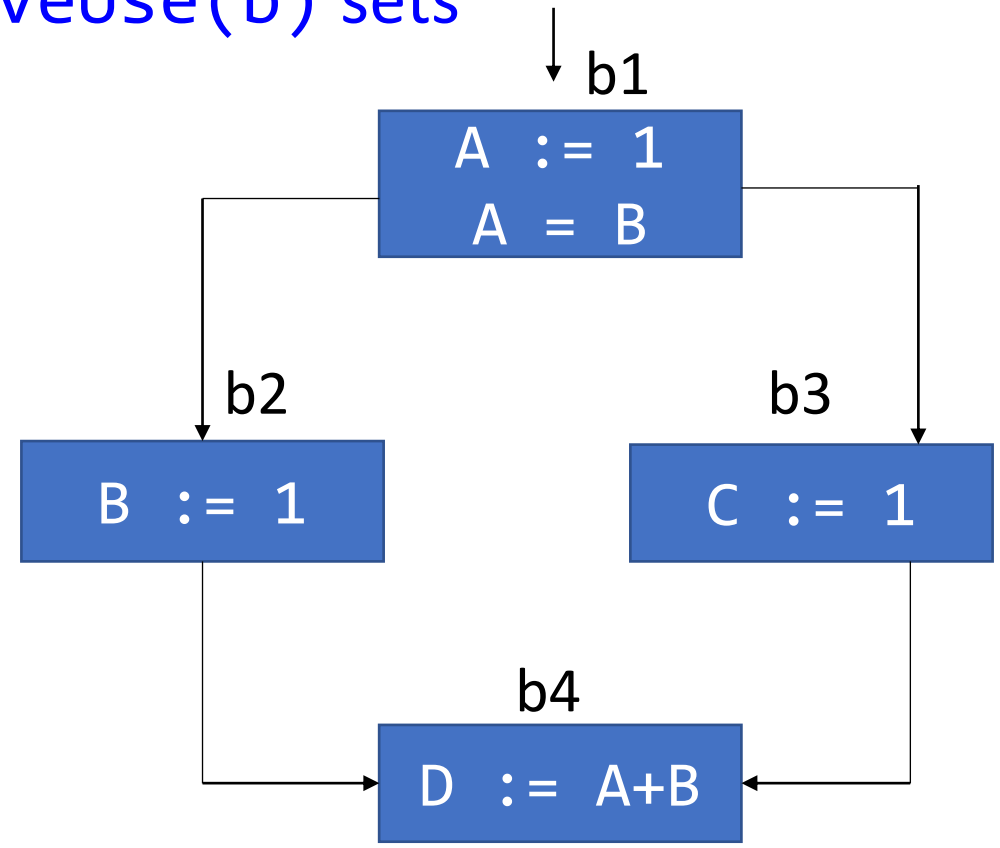
```
A:=1
if A=B then
   B:=1
else
   C:=1
endif
D:=A+B
```



Liveness in a CFG - Example

- Compute `Def(b)` and `LiveUse(b)` sets

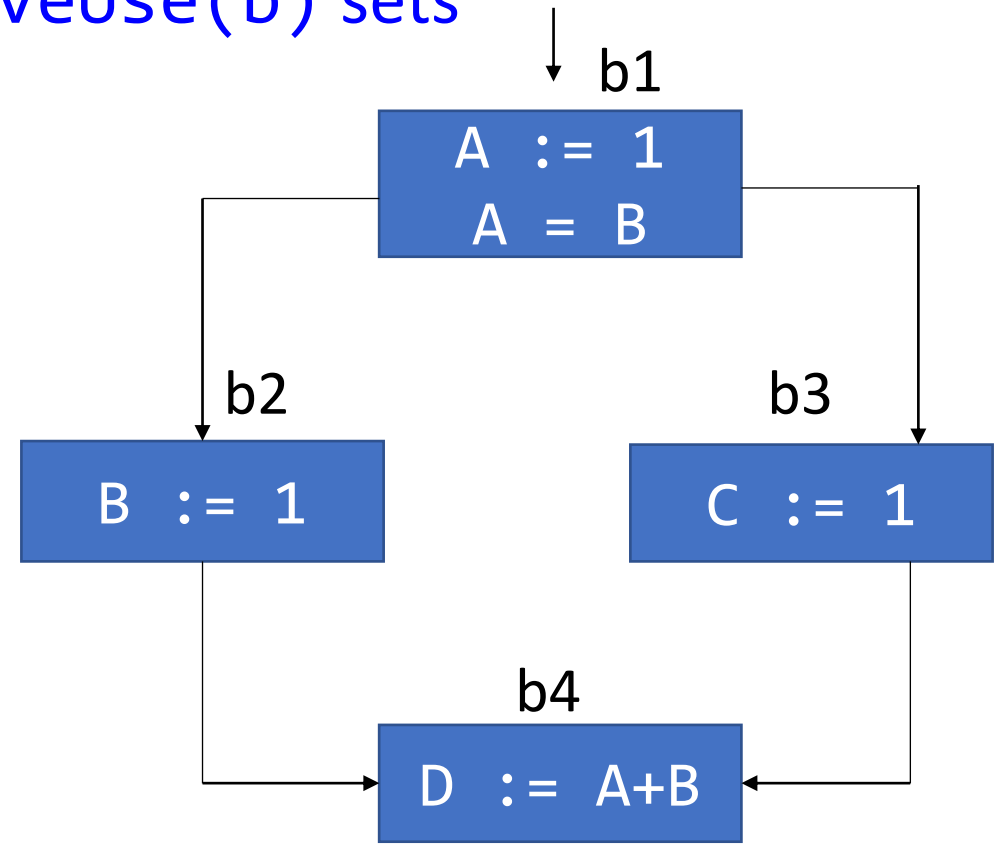
Block	Def	LiveUse
b1		
b2		
b3		
b4		



Liveness in a CFG - Example

- Compute `Def(b)` and `LiveUse(b)` sets

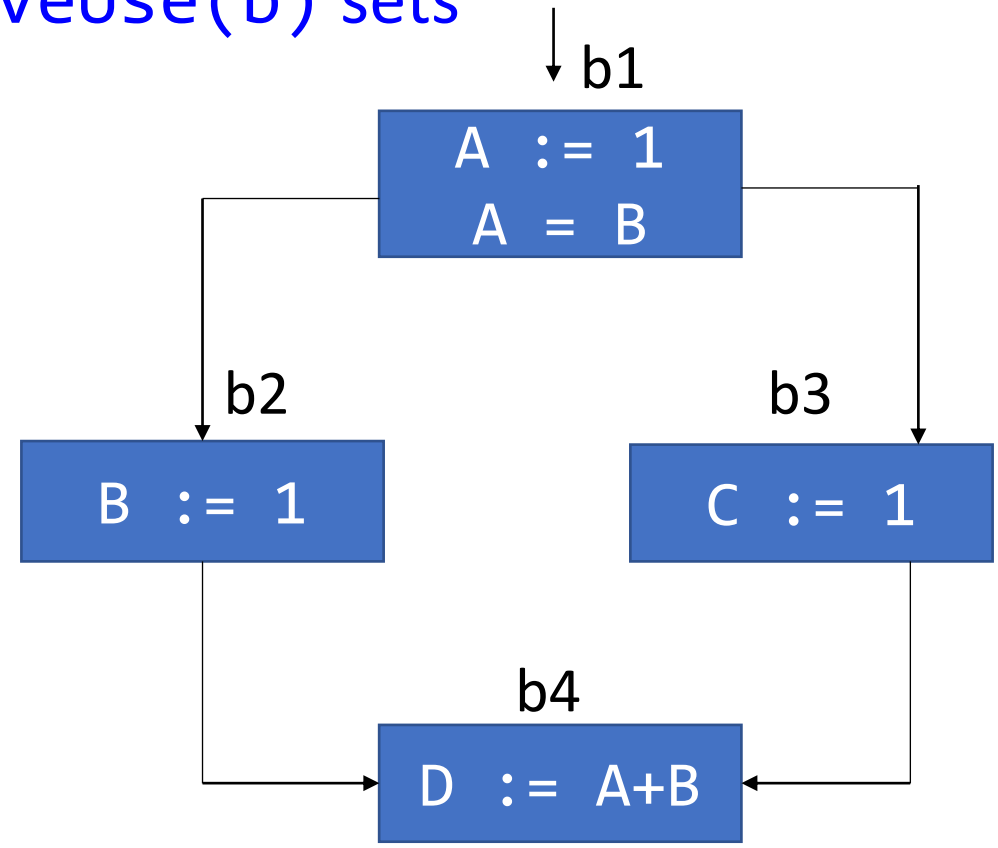
Block	Def	LiveUse
b1	{A}	{B}
b2		
b3		
b4		



Liveness in a CFG - Example

- Compute `Def(b)` and `LiveUse(b)` sets

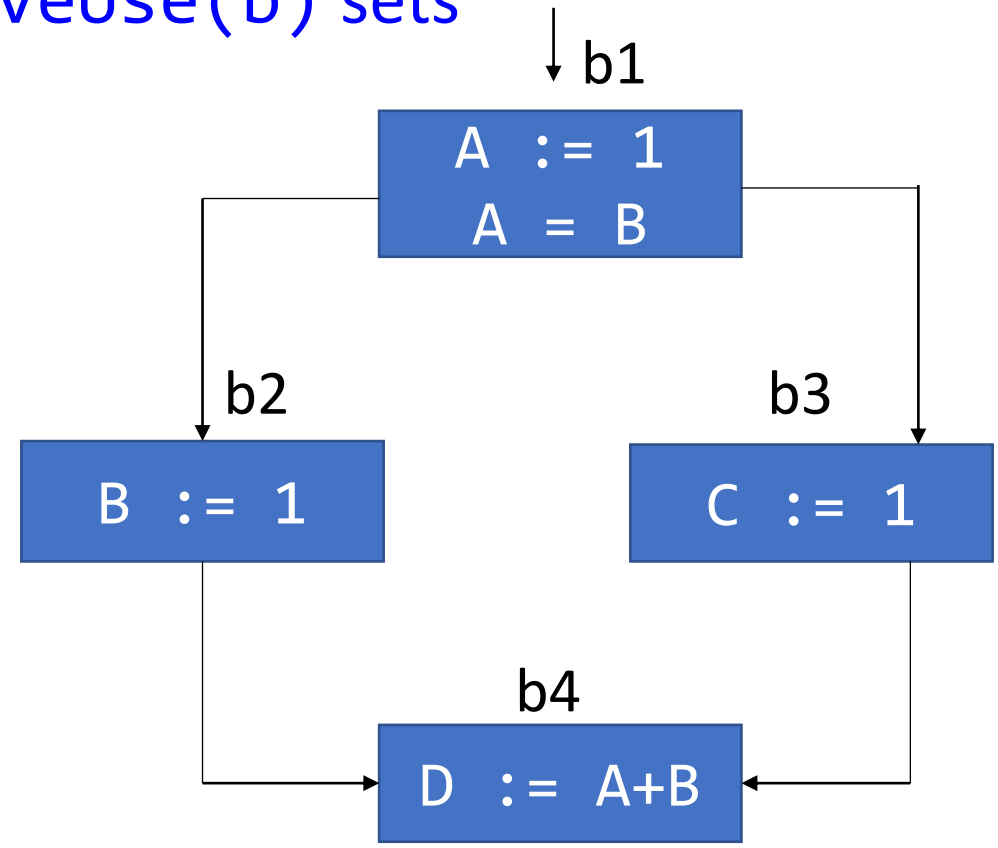
Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3		
b4		



Liveness in a CFG - Example

- Compute `Def(b)` and `LiveUse(b)` sets

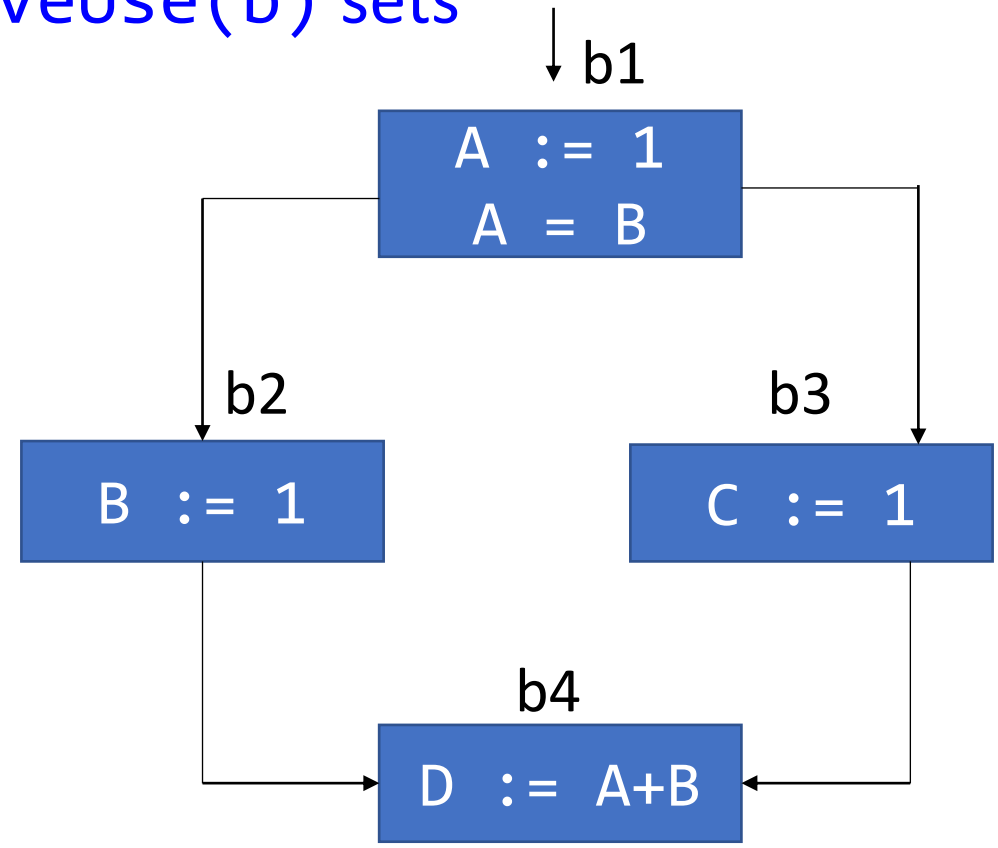
Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4		



Liveness in a CFG - Example

- Compute `Def(b)` and `LiveUse(b)` sets

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

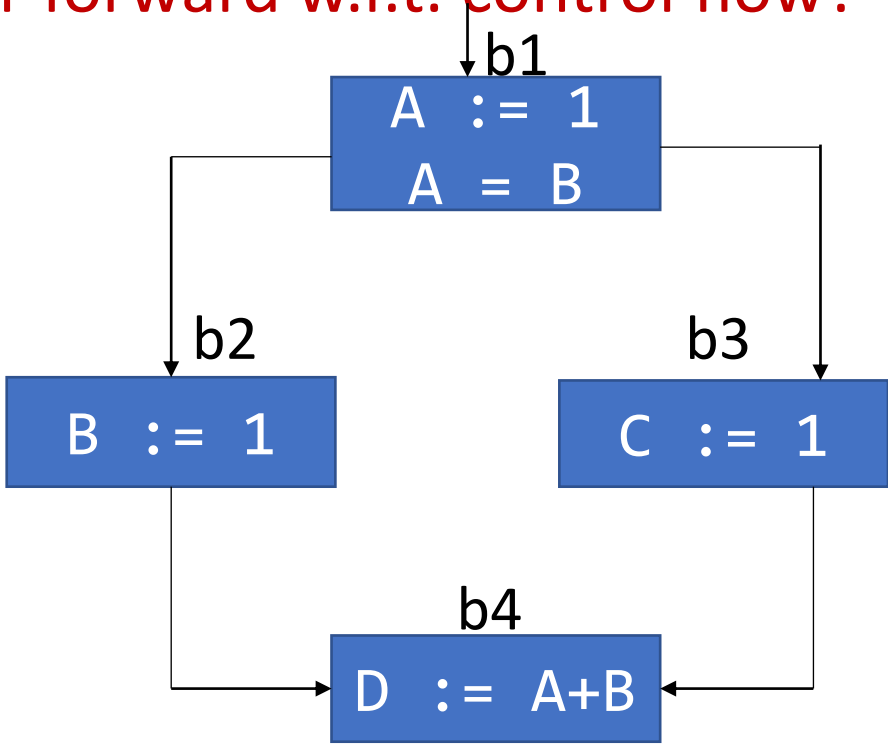


Liveness in a CFG - Example

- start from use of a variable to its definition.

Is this analysis going backward or forward w.r.t. control flow?

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

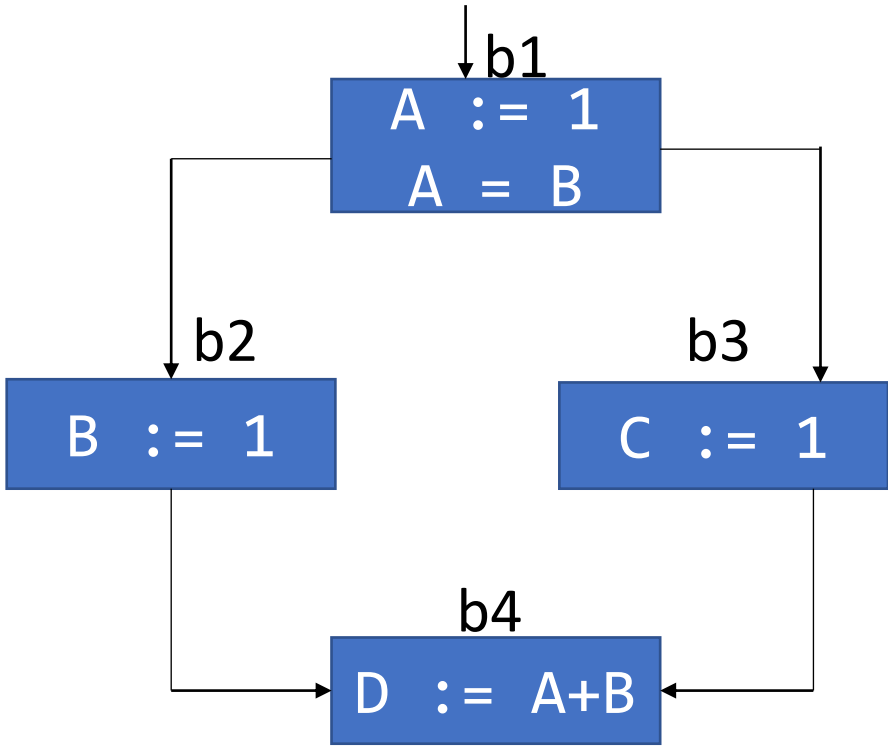


Liveness in a CFG - Example

- start from use of a variable to its definition.

Backward-flow problem

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

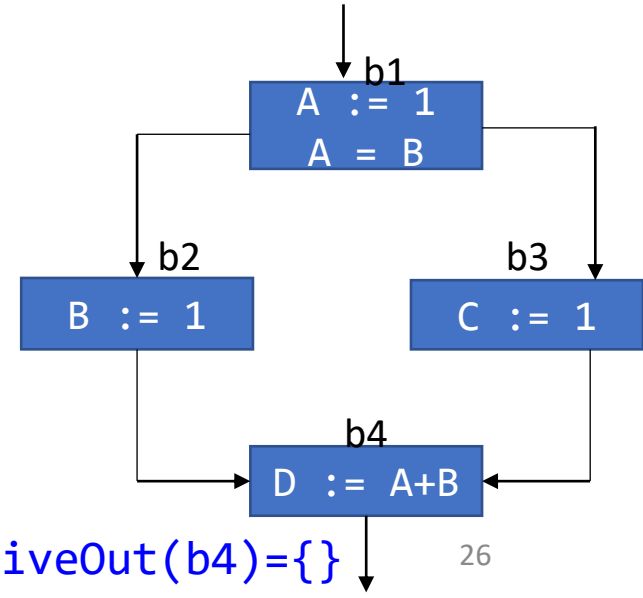


Liveness in a CFG - Example

- Start from use of a variable to its definition.
- Compute LiveOut and LiveIn sets:

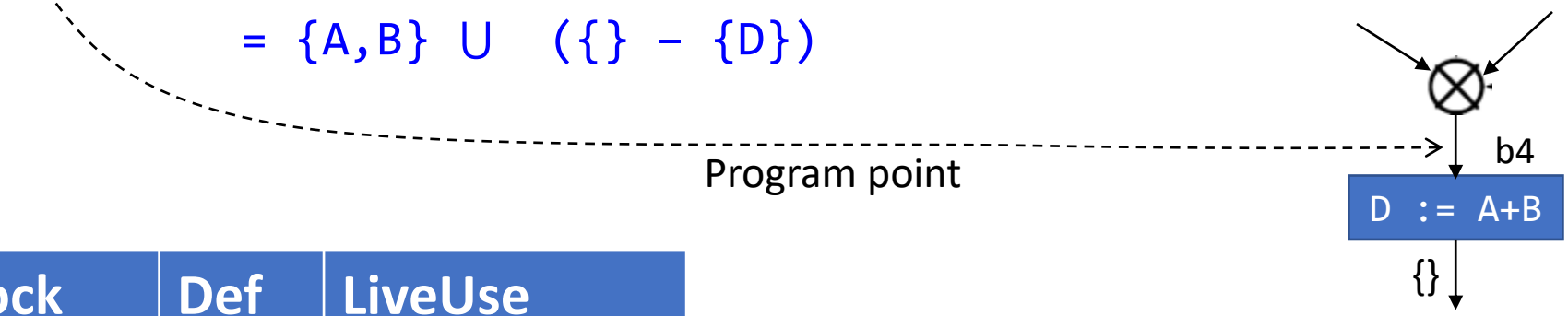
```
LiveIn(b) = LiveUse(b) U  (LiveOut(b) – Def(b))
```

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}



Liveness in a CFG - Example

```
LiveIn(b4) = LiveUse(b4) U  (LiveOut(b4) – Def(b4))
           = {A,B} U  ({} – {D})
```



Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

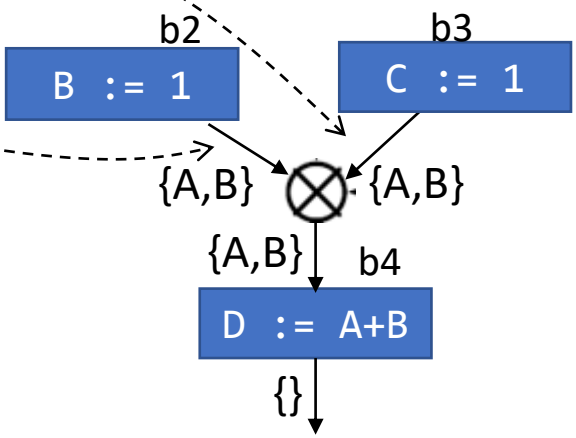
Liveness in a CFG - Example

LiveOut(b) = $\cup_{i \in S(b)}$ LiveIn(i)

LiveOut(b3) = LiveIn(b4) = {A,B}

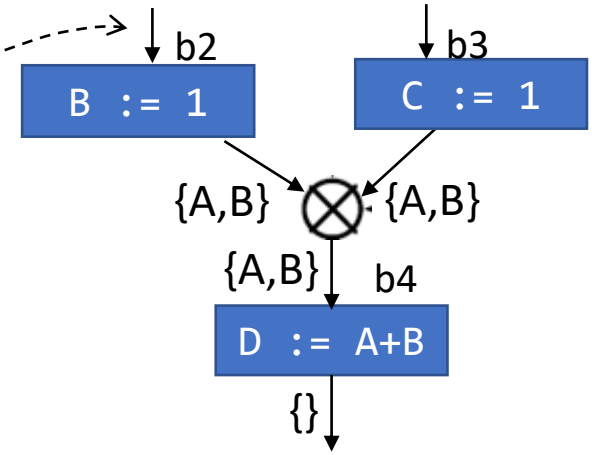
LiveOut(b2) = LiveIn(b4) = {A,B}

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}



Liveness in a CFG - Example

```
LiveIn(b3) = LiveUse(b3) ∪  (LiveOut(b3) – Def(b3))
           = {} ∪  ({A,B} – {C}) = {A,B}
LiveIn(b2) = LiveUse(b2) ∪  (LiveOut(b2) – Def(b2))
           = {} ∪  ({A,B} – {B}) = {A}
```



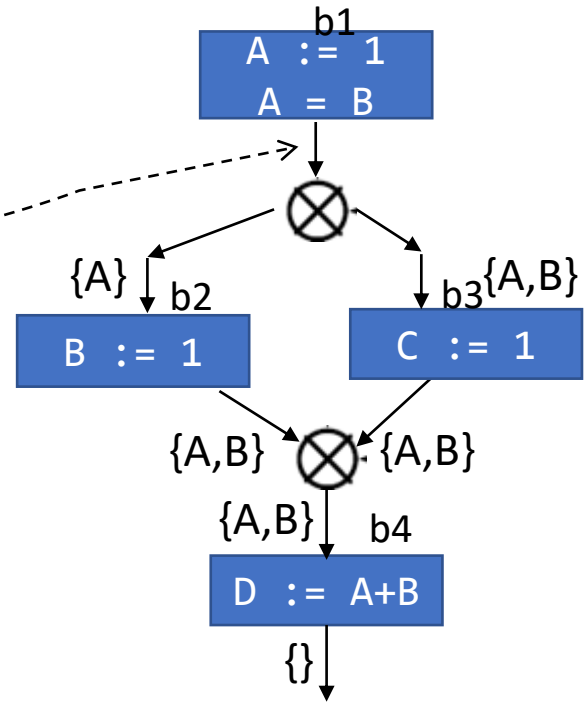
Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

Liveness in a CFG - Example

$\texttt{LiveOut(b)} = \bigcup_{i \in S(b)} \text{LiveIn(i)}$

$\texttt{LiveOut(b1)} = \text{LiveIn(b2)} \cup \text{LiveIn(b3)}$

$= \{A\} \cup \{A,B\} = \{A,B\}$

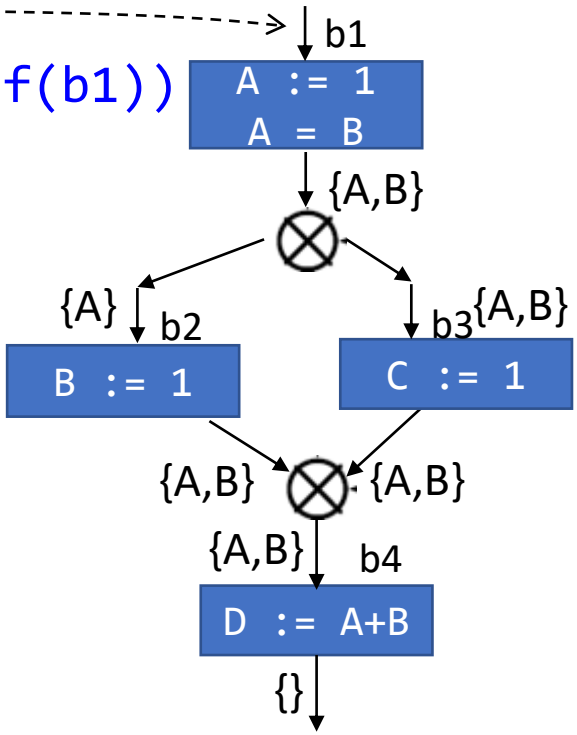


Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

Liveness in a CFG - Example

LiveIn(b1) = LiveUse(b1) ∪ (LiveOut(b1) – Def(b1))

= {B} ∪ ({A,B} – {A}) = {B}



Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

Liveness in a CFG - Example

- Summary: Compute `LiveIn(b)` and `LiveOut(b)`

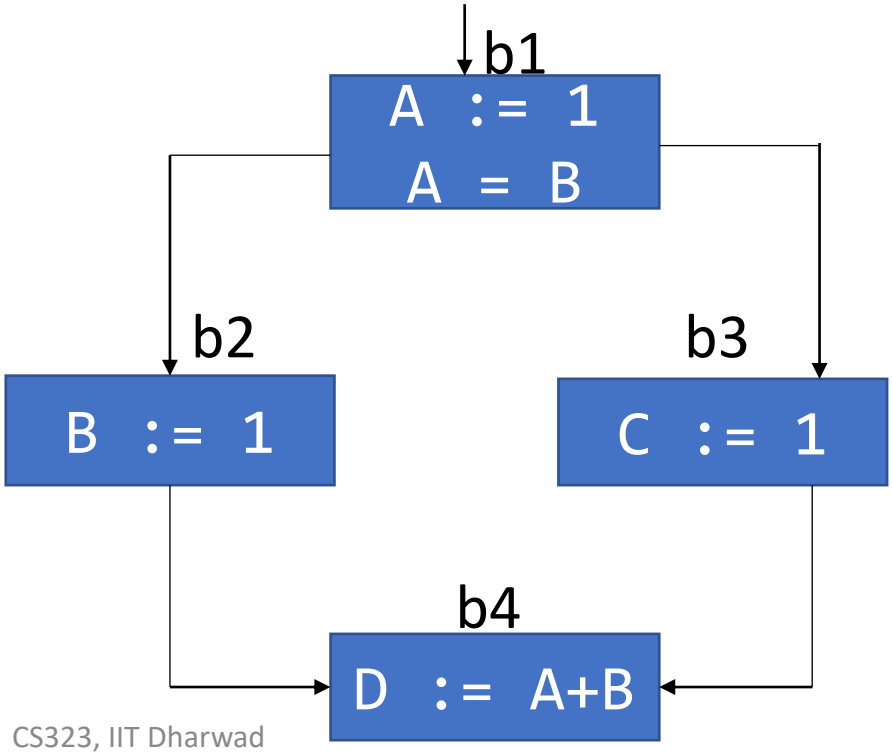
```
LiveIn(b) = LiveUse(b) U  (LiveOut(b) – Def(b))
```

Block	Def	LiveUse
b1	{A}	{B}
b2	{B}	{}
b3	{C}	{}
b4	{D}	{A,B}

Block	LiveIn	LiveOut
b1	{B}	{A,B}
b2	{A}	{A,B}
b3	{A,B}	{A,B}
b4	{A,B}	{}

Liveness in a CFG – Use Case

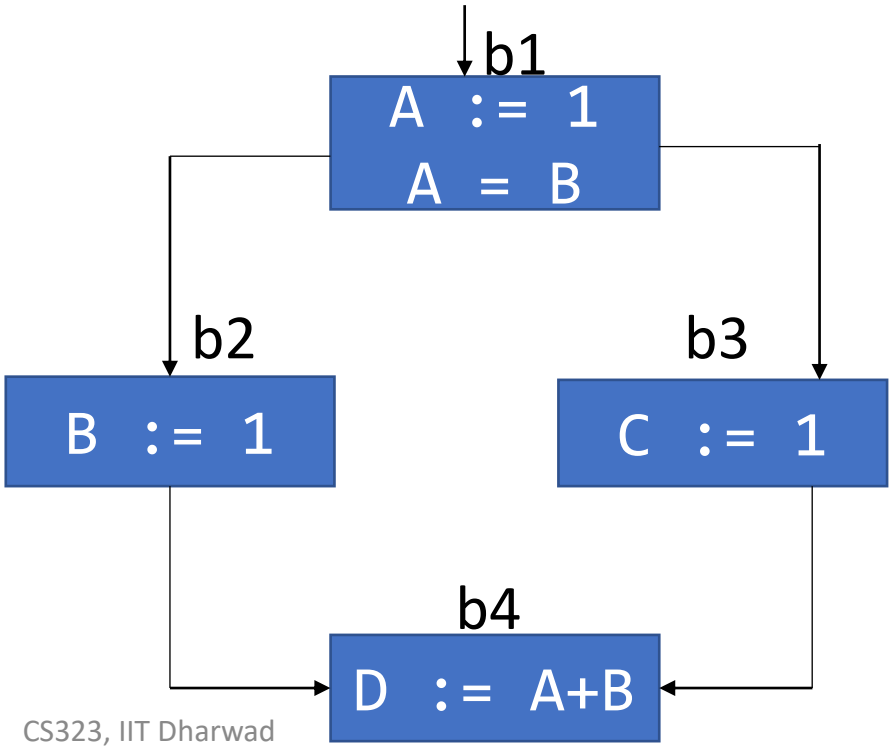
- Assume that the CFG below represents *your entire program* (b1 is the entry to program and b4 is the exit)
  - What can you infer from the table?



Block	LiveIn	LiveOut
b1	{B}	{A,B}
b2	{A}	{A,B}
b3	{A,B}	{A,B}
b4	{A,B}	{}

Liveness in a CFG – Use Case

- Assume that the CFG below represents *your entire program*
  - Variable B is live at the entrance of b1, the entry basic block of CFG. This implies that B is used before it is defined. An error!



Block	LiveIn	LiveOut
b1	{B}	{A,B}
b2	{A}	{A,B}
b3	{A,B}	{A,B}
b4	{A,B}	{}

Liveness in a CFG – Use Case

- Liveness information tells us what variable is dead. Can remove statements that assign to dead variables.

X is dead here implies that we can remove this statement.

```
X = 1
Y = X + 2
Z = Y + A
```

```
X = 1
Y = 1 + 2
Z = Y + A
```

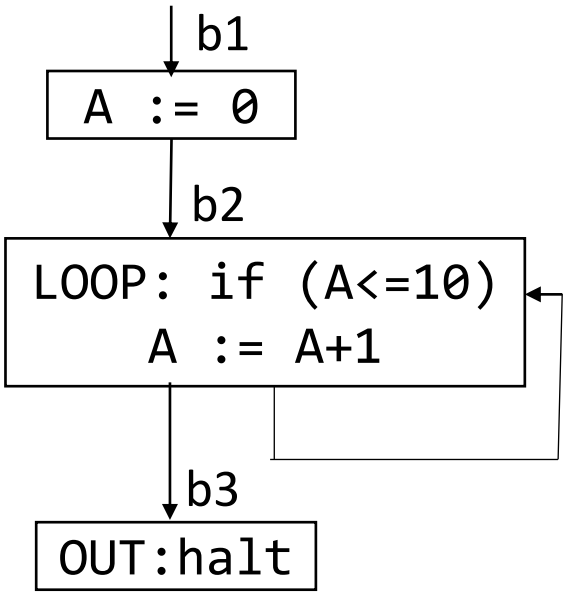
```
X = 1
Y = 1 + 2
Z = Y + A
```

Constant Propagation

Dead Code Elimination

Liveness in a CFG – Example (Loop)

- How do we compute liveness information when a loop is present?



Block	Def	LiveUse
b1	{A}	{}
b2	{A}	{A}
b3	{}	{}

Block	LiveIn	LiveOut
b1	{}	{A}
b2	{A}	{A}
B3	{}	{}

Liveness in a CFG - Observations

- Liveness is computed as information is *transferred* between adjacent statements
- At a program point, a variable can be live or not live (property: true or false)
  - To begin with we did not have any information=property is false

At a program point can the liveness information change?

- Yes, Liveness information changes from false to true and not otherwise.