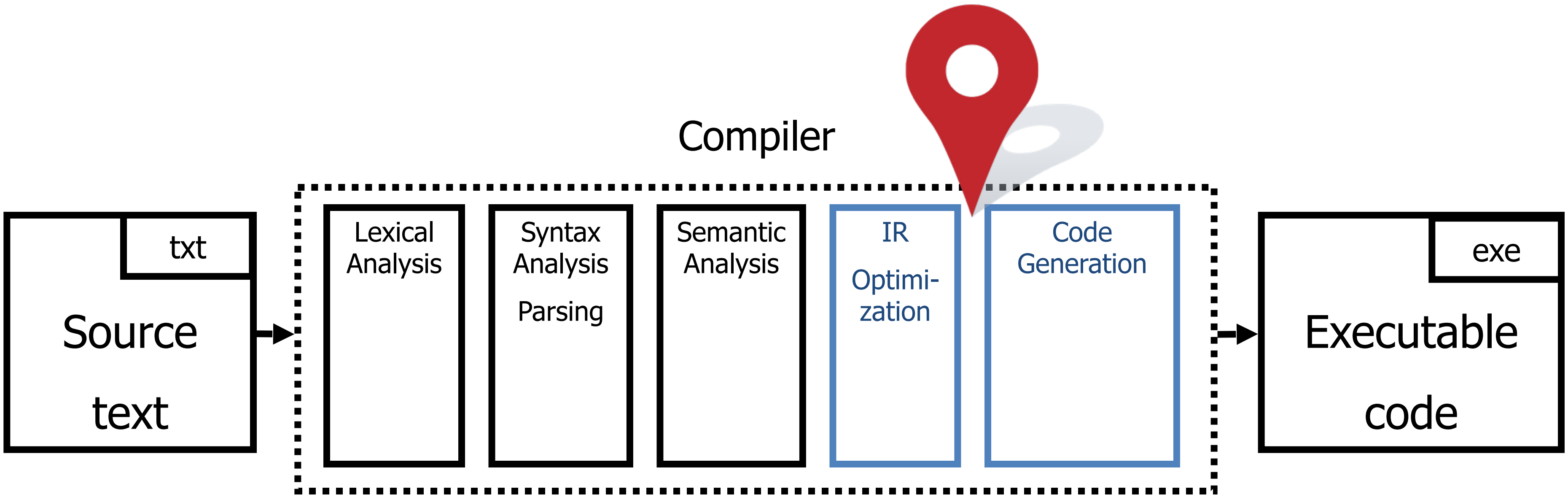
THEORY OF COMPILATION

LECTURE 07

STATIC ANALYSIS

You are here



Compiler
txt
Source
text
Lexical Analysis
Syntax Analysis
Parsing
Semantic Analysis
IR
Optimi-zation
Code Generation
exe
Executable
code

Up Until Now

AST

(*...parsing & shit*)

IR

(1) t = A – B
(2) u = A – C
(3) v = t + u
(4) A = D
(5) D = v + u

Asm

```
LD R1, @A
LD R2, @B
SUB R2,R1,R2
LD R3, @C
SUB R1,R1,R3
ADD R3,R2,R1
LD R2, @D
ADD R1,R3,R1
ST @A, R2
ST @D, R1
```

Syntax directed translation
Backpatching

Register allocation
Instruction translation

Static Analysis

"The **algorithmic discovery** of **properties** of a program by inspection of its **source text**"

-- Manna, Pnueli

Reason statically — at compile time — about the possible runtime behaviors of a program

- Does not have to *literally* be the source text, just means w/o running it
- In a compiler, we mostly use IR

Static Analysis

- What for..?

Register allocation

(liveness analysis used in the previous lecture)

Optimizations

```
area = width * height
p = 0
z = p * area + 1
```

e.g. in this code, `z` can be replaced by 1, and `area` can be discarded.

(or can it?)

Advanced semantic checks

```
Record a1;
if (..) { a1 = new }
a1.write();
```

"a1 may be uninitialized"

Static Analysis

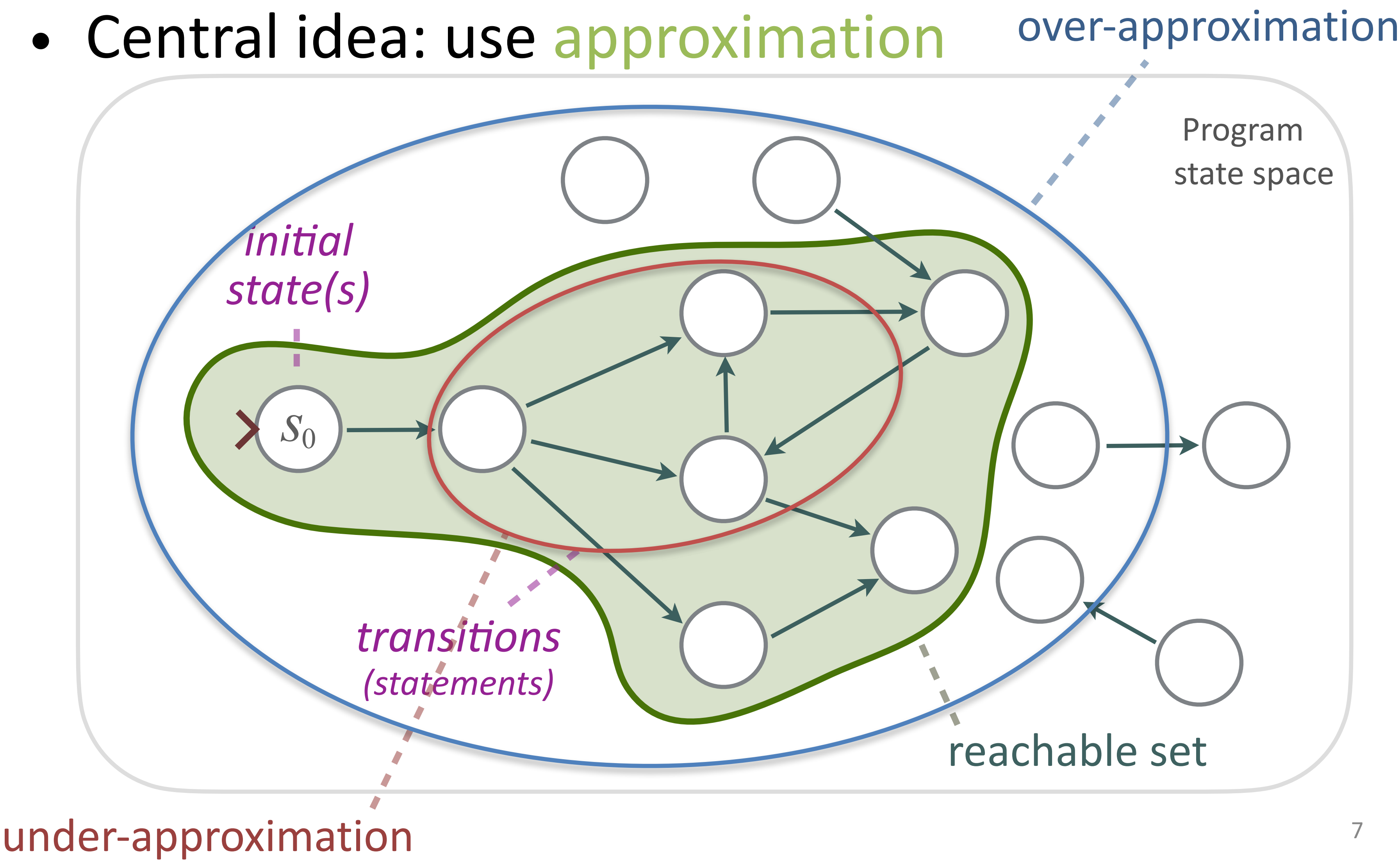
```
x = ?
if (x > 0) {
    y = 42;
} else {
    y = 73;
    foo();
}
assert (y == 42);
```

Is the assertion true in *all* possible executions?

- Bad news: problem is generally undecidable

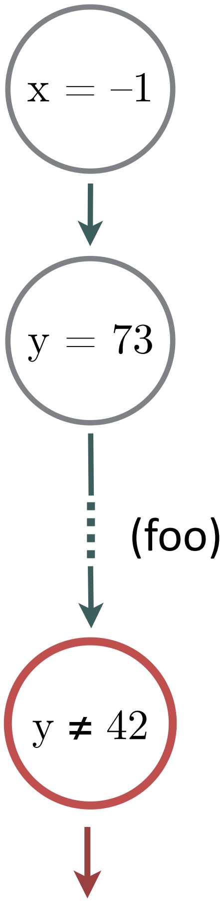
Static Analysis

- Central idea: use approximation



Over-Approximation

```
x = ?
if (x > 0) {
    y = 42;
} else {
    y = 73;
    foo();
}
assert (y == 42);
```



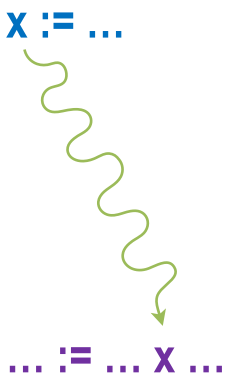
- Conservative static analysis: **assertion may be violated**

Example: Def-Before-Use

- Concept of definition and use:

$\boxed{x = y + z}$

  - is a **definition** of x
  - is a **use** of y and z



- A program satisfies *def-before-use* if
  - on any execution path, and for any variable x —
  - there is some definition of x before all uses of x

Example: Def-Before-Use

fun see(x)

{ x }

1 y := x

{ x, y }

2 **do** {

{ x, y }

3 **if** (2 * y = x) {

{ x, y }

4 z = y

{ x, y, z }

}

{ x, y, ~~z~~ }

5 y := y – 1

{ x, y, ~~z~~ }

} **while** (y ≥ 0)

{ x, y, ~~z~~ }

6 ret := z + 1

{ x, y, ~~z~~, ret }

set of variables that *must* have been defined until now

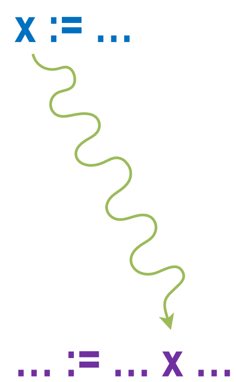
Control flow

Example: Reaching Definitions

- Concept of definition and use:

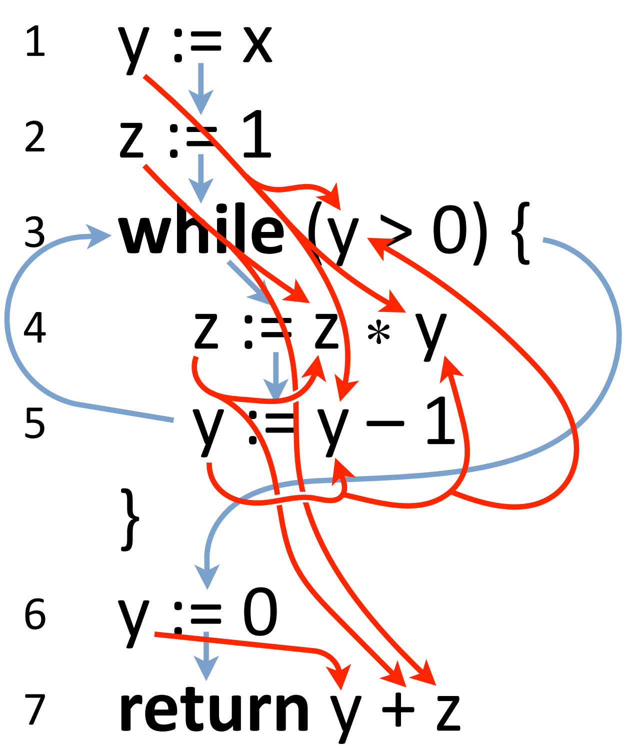
$\boxed{x = y + z}$

  - is a **definition** of x
  - is a **use** of y and z



- A definition *reaches* a use if
  - value written by definition…
  - …may be read by use

Example: Reaching Definitions



- A definition *reaches* a use if
  - value written by definition…
  - …may be read by use

(adapted from Nielson, Nielson & Hankin)

Example: Reaching Definitions

set of definitions that *may* reach this location

```
                              { }
1   y := x
                              { y@1 }
2   z := 1
                              { y@1, z@2 }
3   while (y > 0) {
                              { y@1, z@2 }
4       z := z * y
                              { y@1, z@4 }
5       y := y − 1
                              { y@5, z@4 }
    }
                              { y@1, z@2 }
6   y := 0
                              { y@6, z@2 }
7   return y + z
```

(adapted from Nielson, Nielson & Hankin)

Example: Reaching Definitions

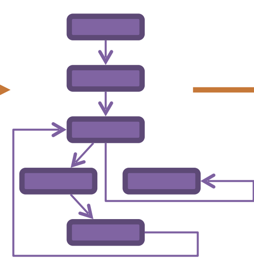
```
                         { }
1   y := x
                         { y@1 }
2   z := 1
                         { y@1, z@2, y@5, z@4 }
3   while (y > 0) {
                         { y@1, z@2, y@5, z@4 }
4       z := z * y
                         { y@1, z@4, y@5 }
5       y := y – 1
                         { y@5, z@4 }
    }
                         { y@1, z@2, y@5, z@4 }
6   y := 0
                         { y@6, z@2, z@4 }
7   return y + z
```

(adapted from Nielson, Nielson & Hankin)

Dataflow Analysis: Overview

```
y := x
z := 1
while (y > 0) {
  z := z * y
  y := y − 1
}
y := 0
return y + z
```

Build Control-flow Graph



Derive Semantic Equations

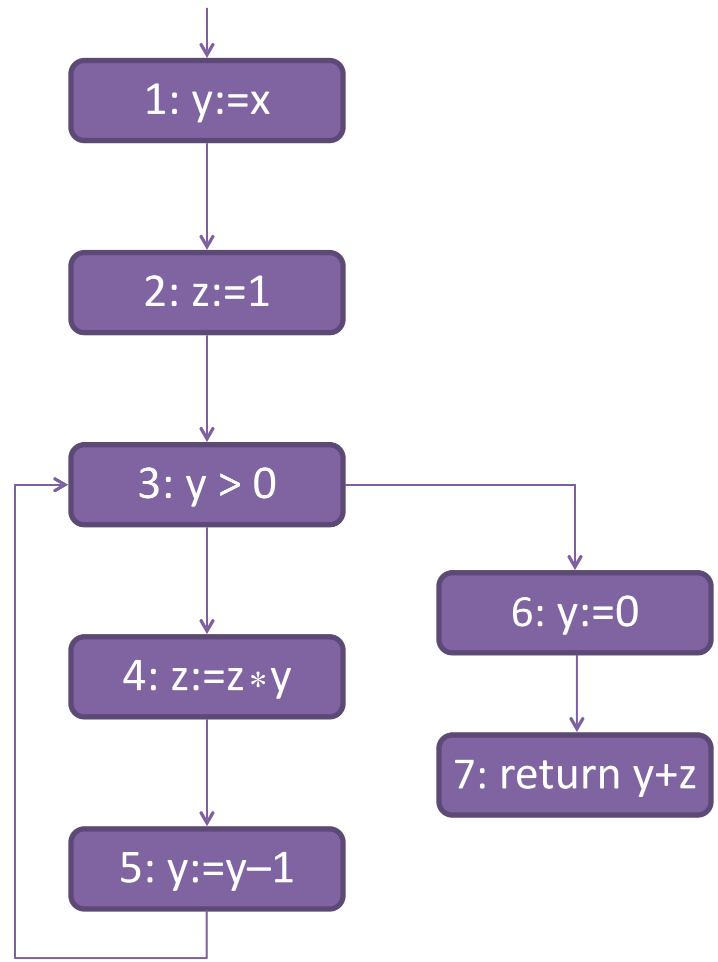
in(1) = { }
in(2) = out(1)
in(3) = out(2) U out(5)
in(4) = out(3)
in(5) = out(4)
in(6) = out(3)
out(1) = in(1) \ { y@* } U { y@1 }

Compute Solution (Fixed Point)

✓ ✗

Control-Flow Graph

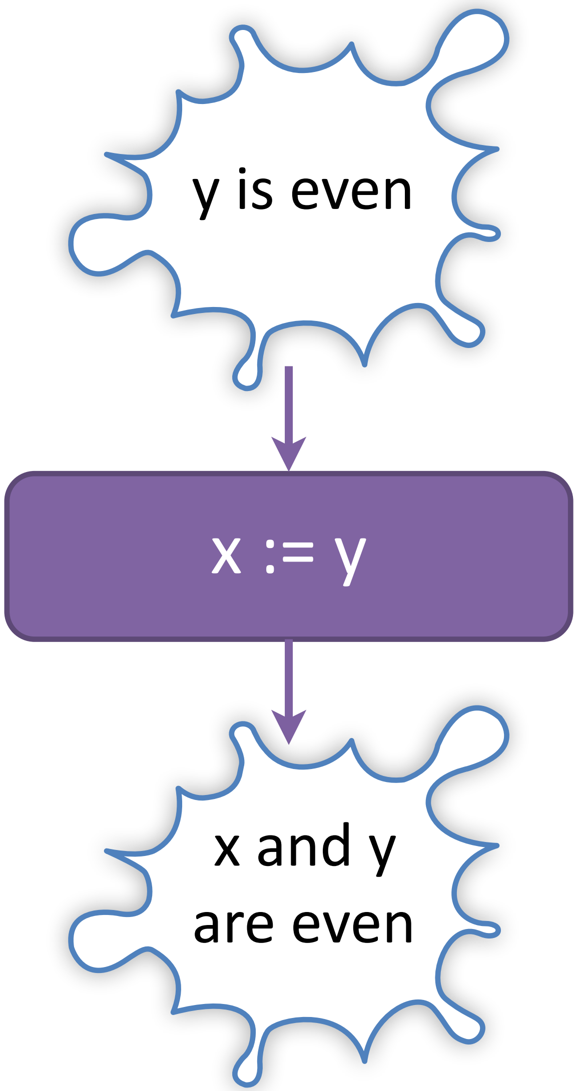
```
1   y := x
2   z := 1
3   while (y > 0) {
4       z := z * y
5       y := y − 1
    }
6   y := 0
7   return y + z
```



Transfer Functions

Given a program statement *S*, we can define a **transfer function** T_S that relates the properties that are true before the statement to the properties that are true after the statement.

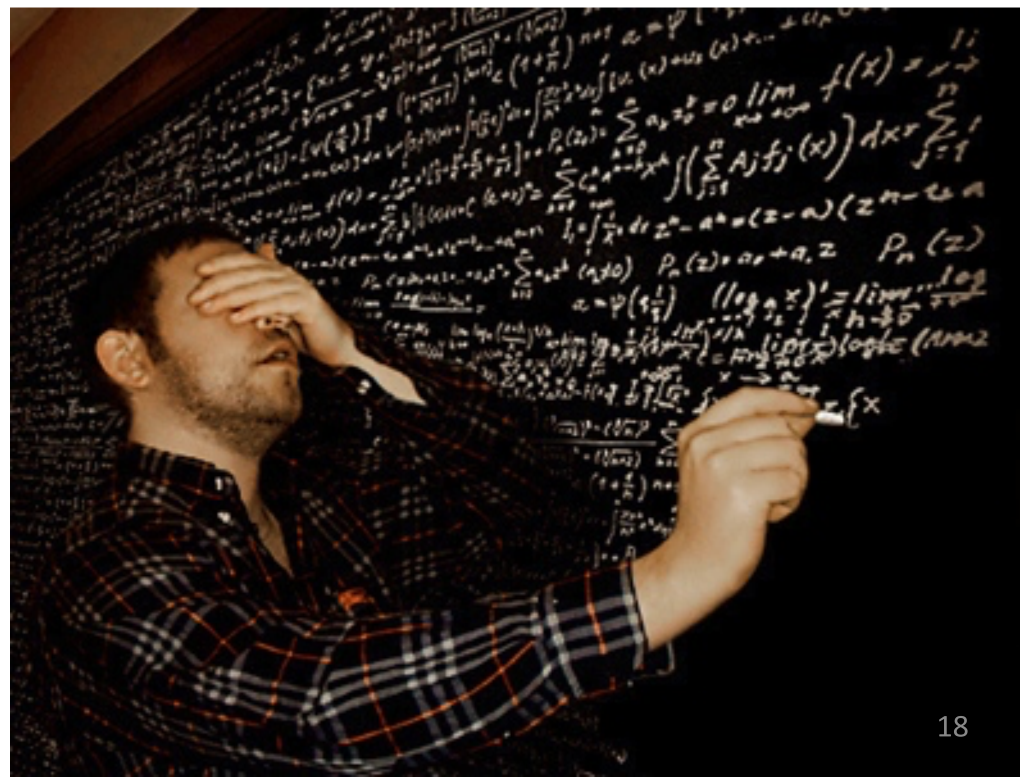
$$T_{x:=y}\begin{bmatrix} x\ (?) \\ y\ \text{even} \end{bmatrix} = \begin{bmatrix} x\ \text{even} \\ y\ \text{even} \end{bmatrix}$$



Time for Some

Math

- Partial Orders
- Upper and Lower Bounds
- Lattices



Partial Orders

- Set P
- Binary relation $\sqsubseteq$ such that $\forall x,y,z \in$ P:
  - $x \sqsubseteq x$ (reflexive)
  - $x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x = y$ (asymmetric)
  - $x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$ (transitive)
- Can use partial order to define
  - Upper and lower bounds
  - Least upper bound
  - Greatest lower bound

Upper Bounds

- For $S \subseteq P$:
  - $x \in P$ is an *upper bound* of S if $\forall y \in S.\ y \sqsubseteq x$
  - $x \in P$ is the *least upper bound* of S if
    - x is an upper bound of S, and
    - $x \sqsubseteq z$ for all upper bounds z of S
  - $\sqcup$ – join, least upper bound, lub, supremum, sup
    - $\sqcup S$ is the least upper bound of S
    - $x \sqcup y = \sqcup\{x,y\}$
  - (Often written as $\vee$ as well)

Lower Bounds

- For $S \subseteq P$:
  - $x \in P$ is a *lower bound* of S if $\forall y \in S.\ x \sqsubseteq y$
  - $x \in P$ is the *greatest lower bound* of S if
    - x is an greatest lower bound of S, and
    - $z \sqsubseteq x$ for all lower bounds z of S
  - $\sqcap$ – meet, greatest lower bound, glb, infimum, inf
    - $\sqcap S$ is the greatest lower bound of S
    - $x \sqcap y = \sqcap\{x,y\}$
  - (Often written as $\wedge$ as well)

Covering

- $x \sqsubset y$ if $x \sqsubseteq y$ and $x \neq y$
- x is *covered* by y (y *covers* x) if
  - $x \sqsubset y$, and
  - no z such that $x \sqsubset z \sqsubset y$
- Conceptually,
  - y covers x if there are no elements between x and y

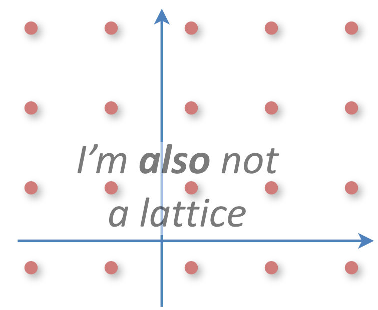
e.g. for $P = \mathbb{Z}$, $\sqsubseteq\ = \leq$

5 covers 4

5 does not cover 3

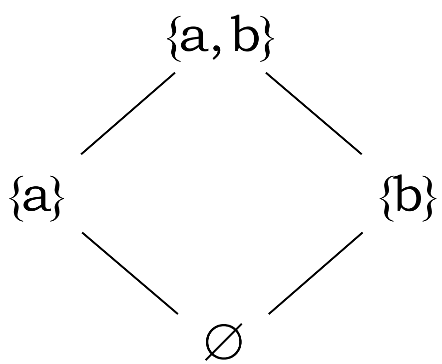
Lattices

- Partially ordered set P
  - If $x \sqcup y$ and $x \sqcap y$ exist for all $x,y \in P$ then P is a *lattice*
  - If $\sqcup S$ and $\sqcap S$ exist for all $S \subseteq P$ then P is a *complete lattice*
- Theorem: all finite lattices are complete.
- Example of a lattice that is not complete:
  - Integers $\mathbb{Z}$
  - $\sqcup$ = max, $\sqcap$ = min
  - But $\sqcup\mathbb{Z}$ and $\sqcap\mathbb{Z}$ do not exist $\Rightarrow$ **not** complete
  - Conversely, $\mathbb{Z} \cup \{+\infty,-\infty\}$ **is** a complete lattice



Example

- $P = \{ \varnothing, \{a\}, \{b\}, \{a,b\} \}$ $= \mathcal{P}(\{a,b\})$
- $x \sqsubseteq y \iff x \subseteq y$ (called a *power-set lattice*)



$\varnothing \sqsubseteq \{a\} \sqsubseteq \{a,b\}$

$\varnothing \sqsubseteq \{b\} \sqsubseteq \{a,b\}$

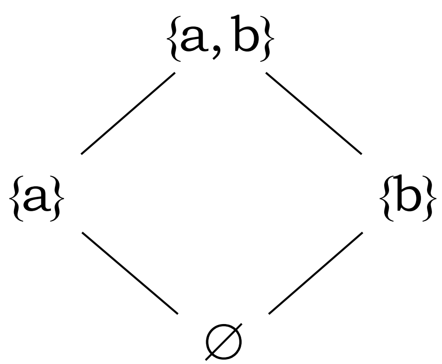
Hasse Diagram

If y covers x:

- Line from y to x
- y above x in diagram

Example

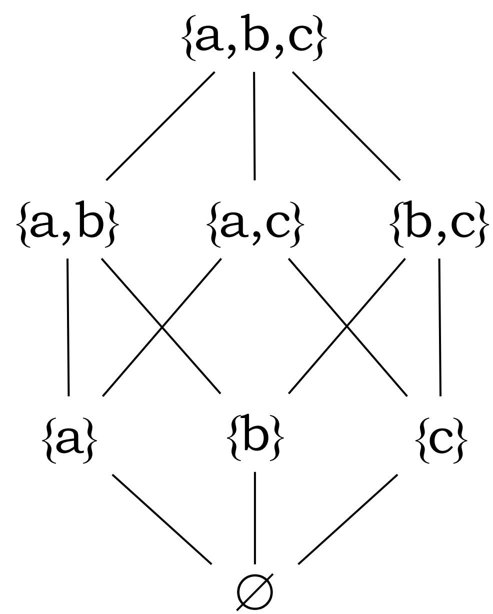
- $P = \{\ \varnothing, \{a\}, \{b\}, \{a,b\}\ \}$
- $x \sqsubseteq y \iff x \subseteq y$



$\varnothing \sqsubseteq \{a\} \sqsubseteq \{a,b\}$

$\varnothing \sqsubseteq \{b\} \sqsubseteq \{a,b\}$

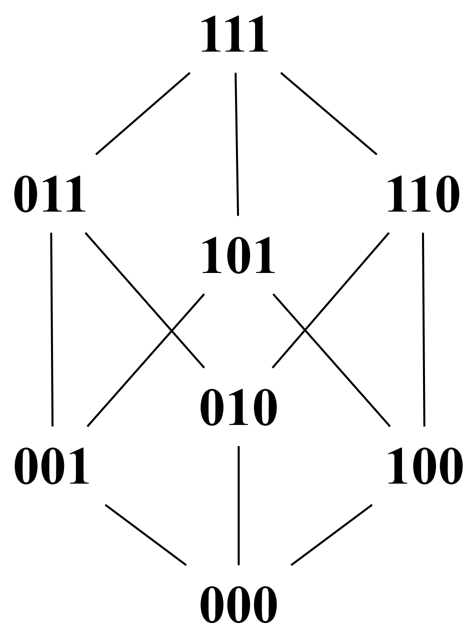
$P = \{\ \varnothing, \{a\}, \{b\}, \{c\},$
$\{a,b\}, \{a,c\}, \{b,c\},$
$\{a,b,c\}\ \}$



Example

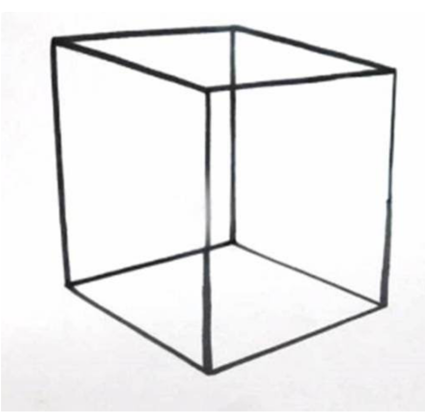
- $P = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- $x \sqsubseteq y \;\Leftrightarrow\; (x \,\&\, y) = x$ where & is bitwise 'and'

(standard boolean lattice, also called *hypercube*)



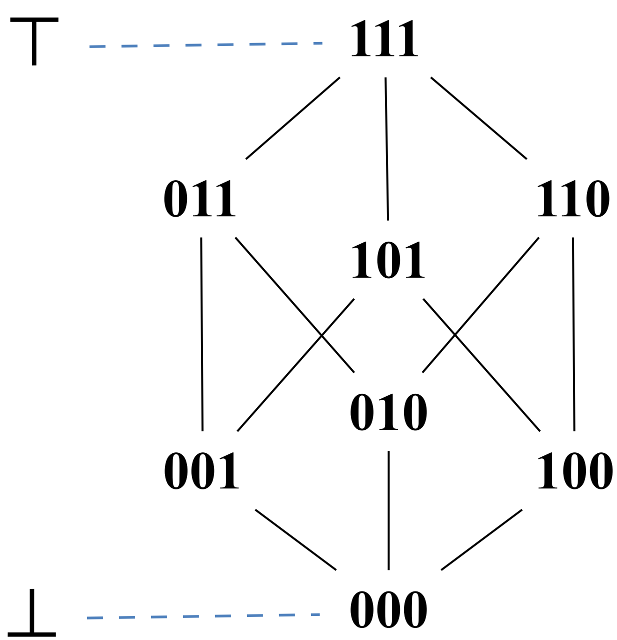
$x \sqcup y = x \mid y$

$x \sqcap y = x \,\&\, y$



Top and Bottom

- Greatest element of P (if it exists) is *top* ($\top$)
- Least element of P (if it exists) is *bottom* ($\bot$)



$\cap \quad \cup$

$$\top = \sqcup P$$

$$x \sqcup y = \boxed{x \mid y}$$

$$x \sqcap y = x \,\&\, y$$

$$\bot = \sqcap P$$

Product Latices

- Given two latices L and Q, the product can easily be made a latice

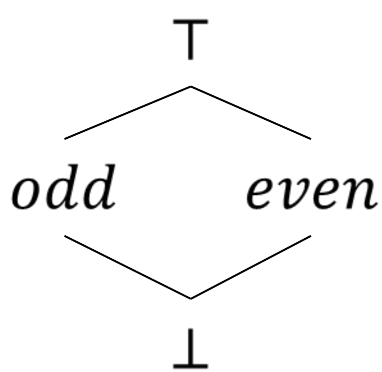
$$(l_1, q_1) \sqsubseteq (l_2, q_2) \Leftrightarrow l_1 \sqsubseteq l_2 \; and \; q_1 \sqsubseteq q_2$$

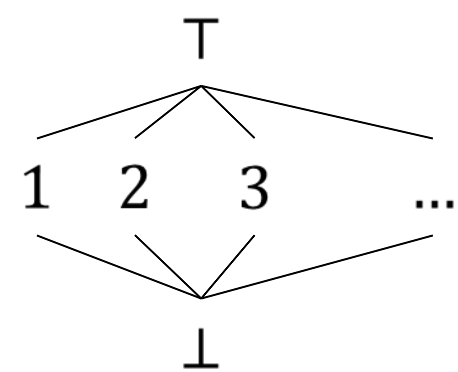
For vectors of L, defining a latice is also easy

$$\langle l_1, l_2, \dots, l_k \rangle \sqsubseteq \langle t_1, t_2, \dots, t_k \rangle \Leftrightarrow \forall_{i \in [1,k]} \, l_i \sqsubseteq t_i$$

Lattices of Program Properties

- Properties of interest can often be arranged into a lattice
- <u>Example</u>: Lattices of values –



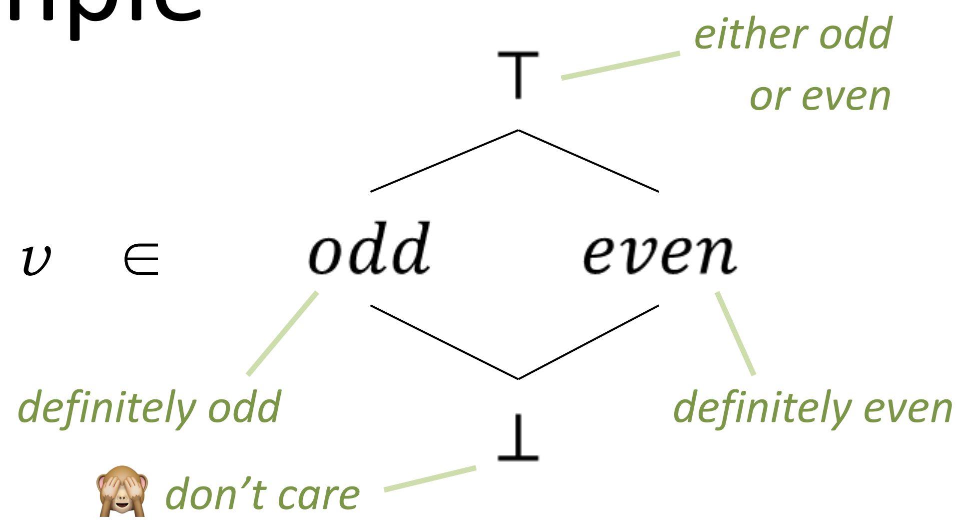


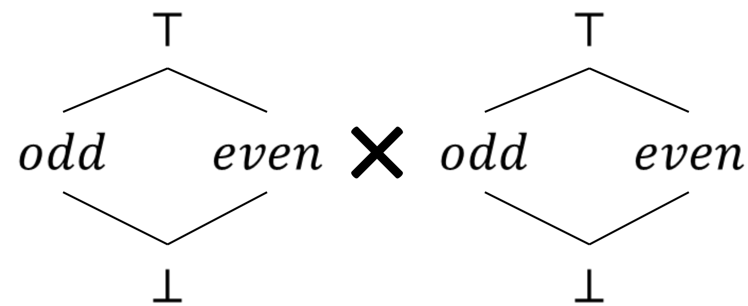
- When the value of each variable is a lattice, the state of the program is a product lattice of the states of all variables.

Example

```
x := 0;
y := 6;
while (x < 10) {
    x := x + 2;
    y := y + x;
}
```

assert (y is even);



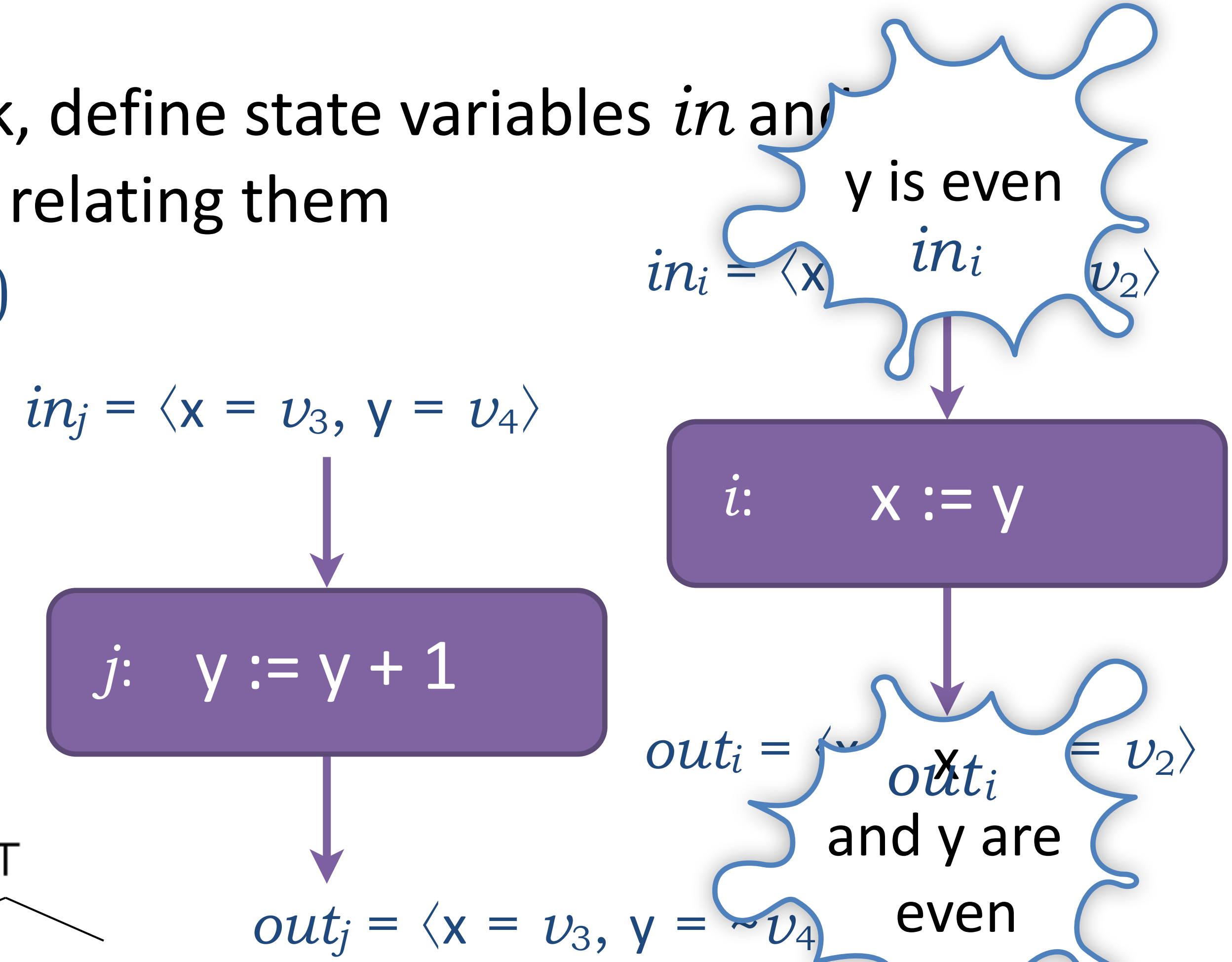


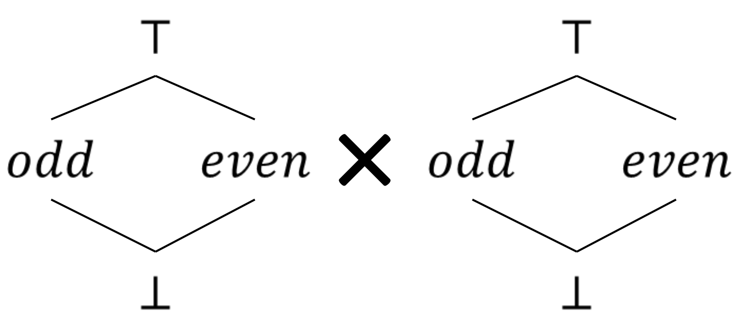
Product lattice of two individual lattices, one per variable

- $\langle \mathsf{x} = \{\perp, even, odd, \top\}, \mathsf{y} = \{\perp, even, odd, \top\}\rangle$
  - *e.g.* $\langle \mathsf{x} = even, \mathsf{y} = odd\rangle \sqsubseteq \langle \mathsf{x} = \top, \mathsf{y} = odd\rangle$
    $\sqsubseteq \langle \mathsf{x} = \top, \mathsf{y} = \top\rangle$

Where were we… ah, yes, **Transfer Functions**

- For every block, define state variables in and out and a function relating them
  - $out_i = T_i(in_i)$





$\sim odd = even \qquad \sim\top = \top$

$\sim even = odd \qquad \sim\bot = \bot$

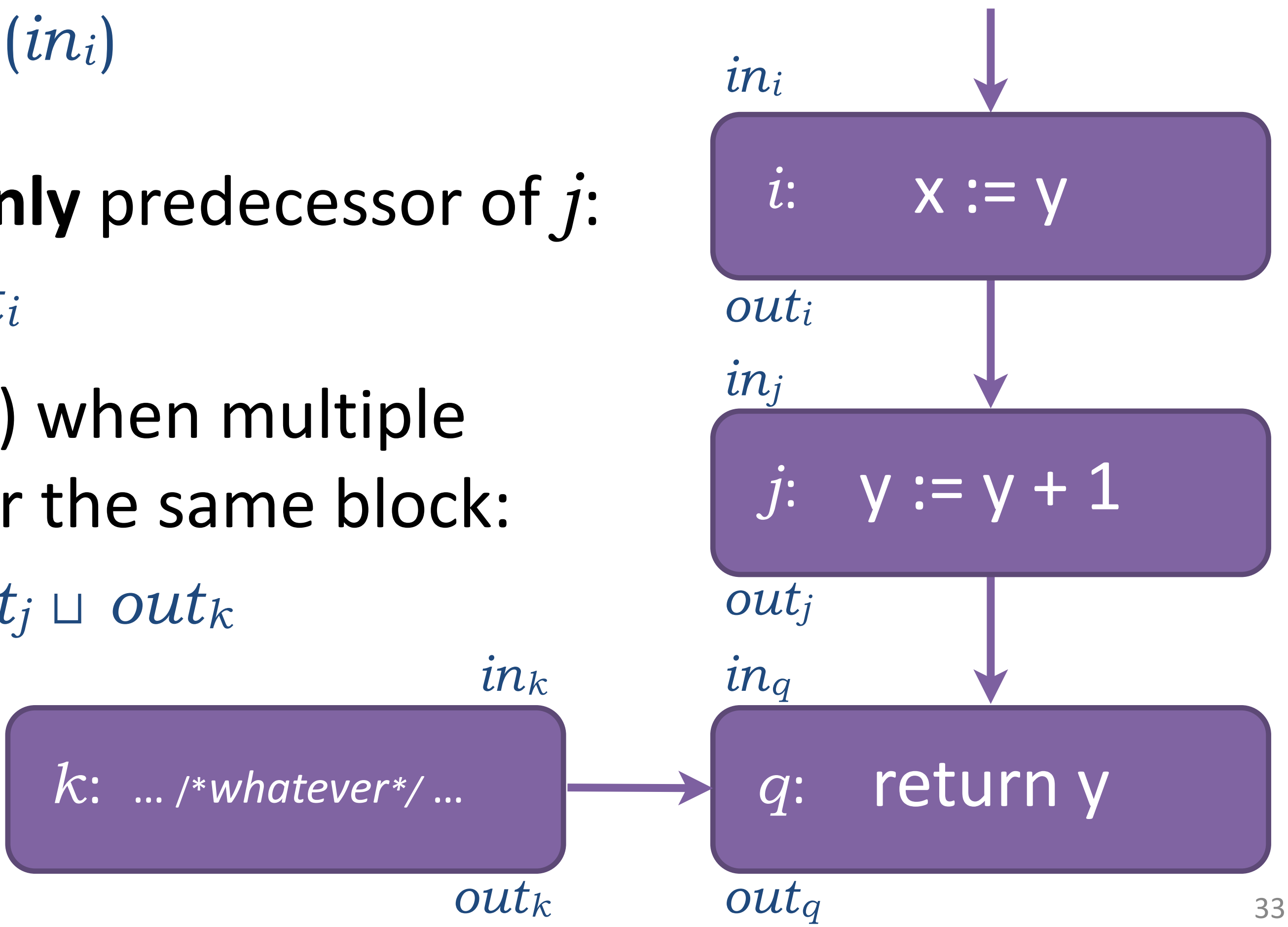
Computing the Transfer Function

- We must hard-code a transfer function specific to the lattice
  - Occasionally, there would be a trade-off between how precise the transfer functions are and how easy it is to compute them

- We can build lattices for arbitrary facts about the program
  - Need to make sure our transfer functions are “well behaved” (we will define “good” behavior later)

From CFG to Equations

- For every block, define state variables in and out
  - $out_i = T_i(in_i)$
- If i is the **only** predecessor of j:
  - $in_j = out_i$
- Use join ($\sqcup$) when multiple edges enter the same block:
  - $in_q = out_j \sqcup out_k$



Back to Reaching Definitions

Domain Lattice

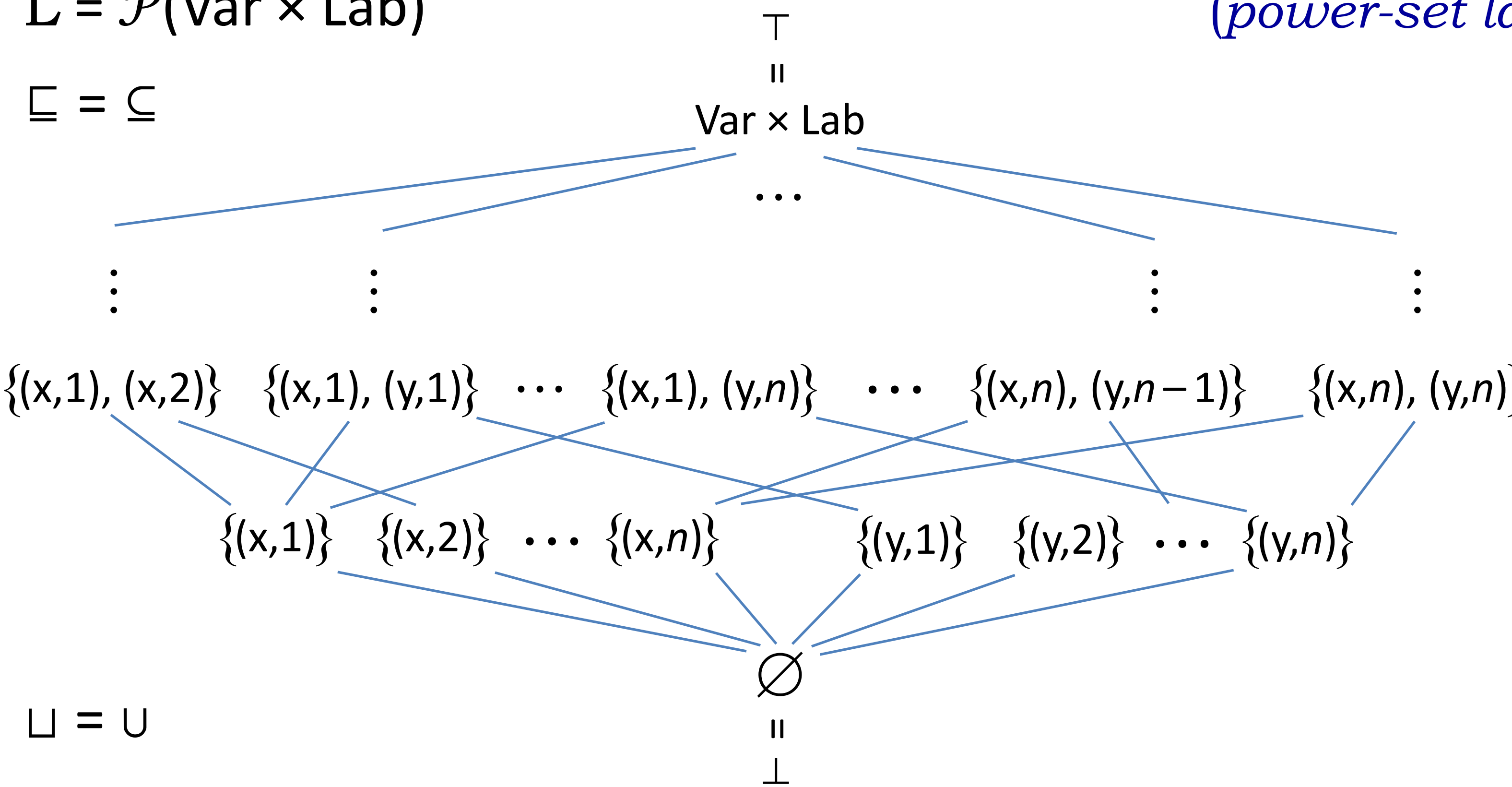
- For every program point, we compute the set of variable definitions that reach it.

$L = \mathcal{P}(\mathrm{Var} \times \mathrm{Lab})$

$\sqsubseteq \; = \; \subseteq$

$\sqcup \; = \; \cup$

(power-set lattice)

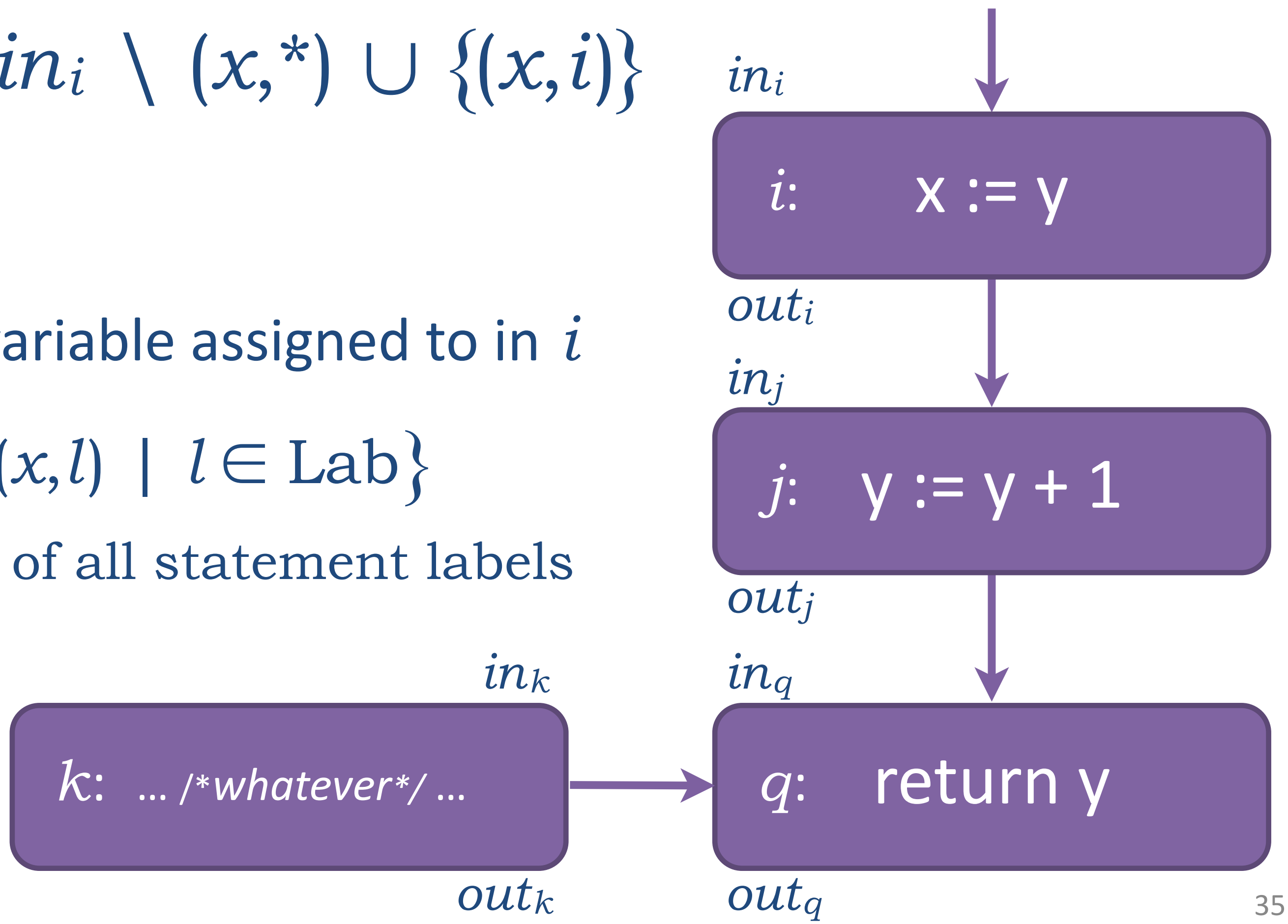


Back to Reaching Definitions

Transfer Functions

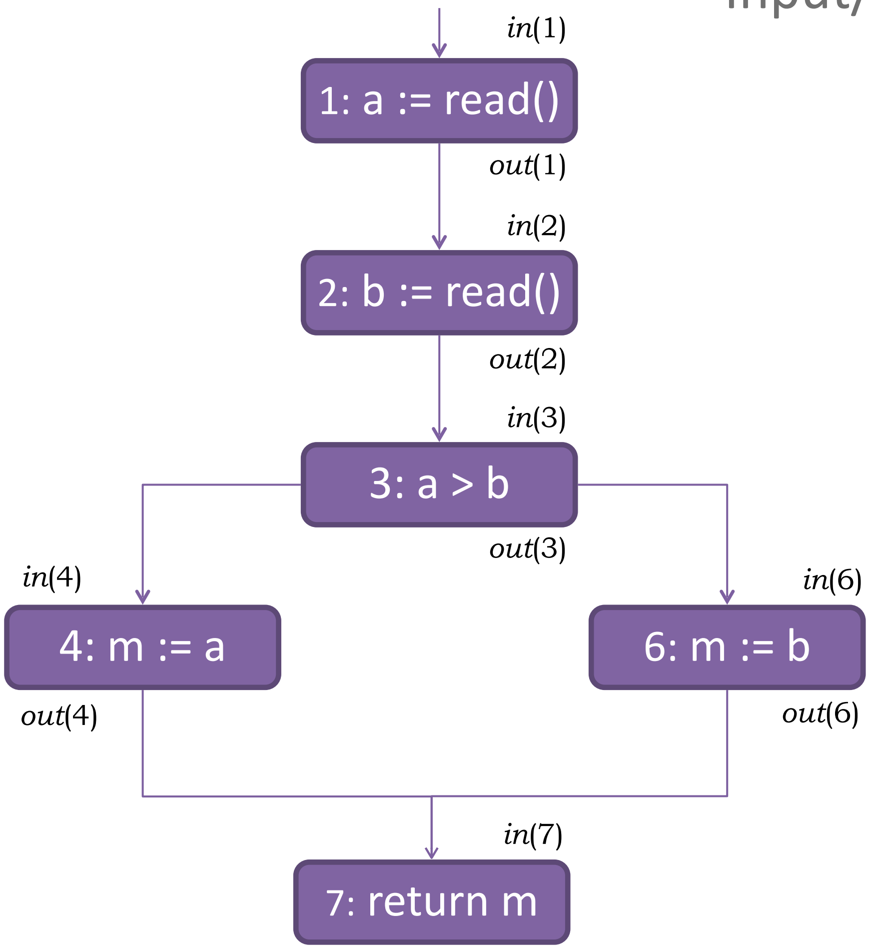
- We define the following transfer function:
  - $out_i = in_i \setminus (x,*) \cup \{(x,i)\}$
- where
  - x is the variable assigned to in i
  - $(x,*) = \{(x,l) \mid l \in \text{Lab}\}$

    Lab = set of all statement labels



Simple Example

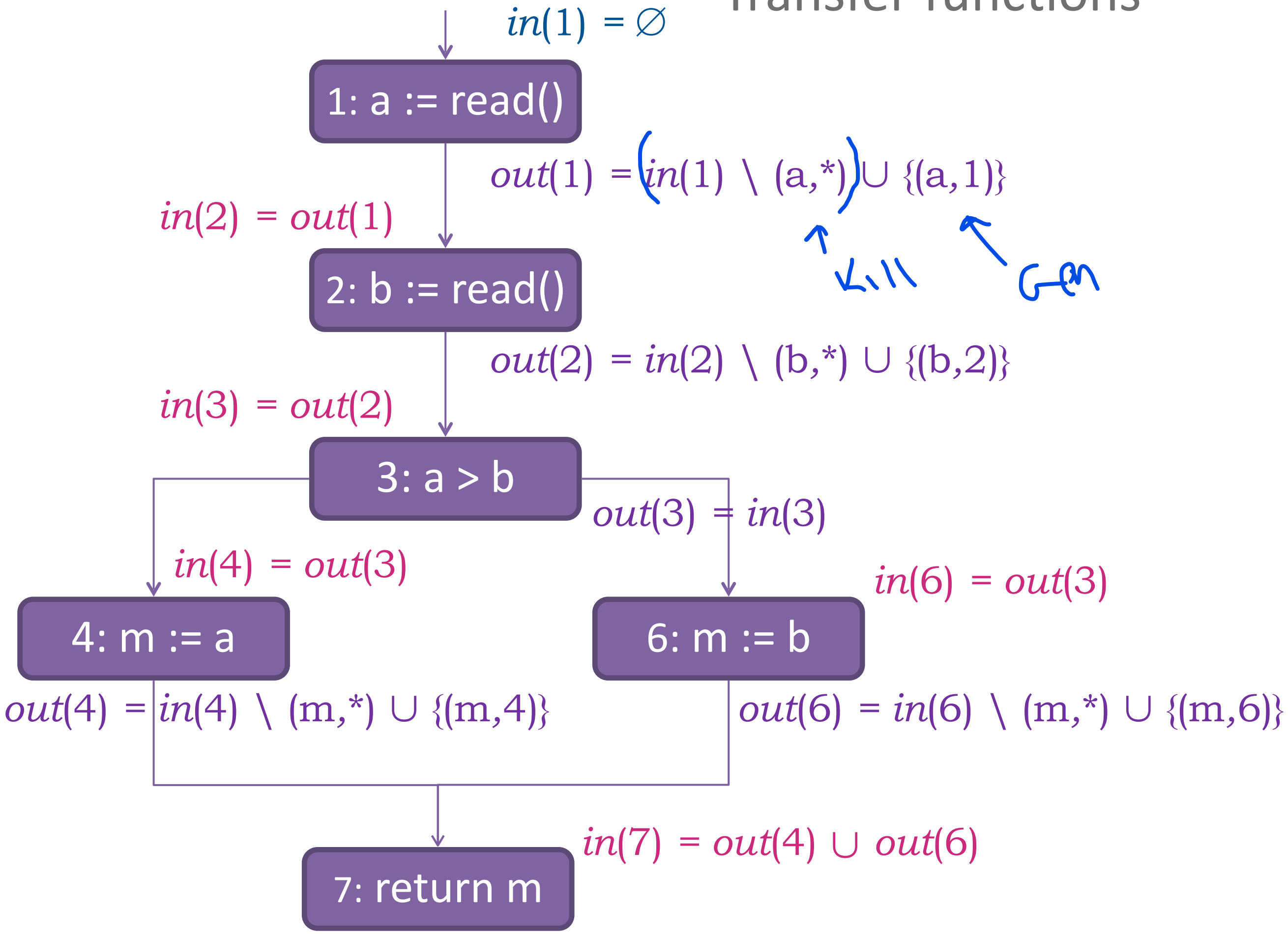
Input/output sets




```
1   a := read()
2   b := read()
3   if (a > b)
4       m := a
5   else
6       m := b
7   return m
```

Simple Example

Transfer functions



Simple Example

Transfer functions

$in(1) = \emptyset$

$out(1) = in(1) \setminus (a,*) \cup \{(a,1)\}$

$in(2) = out(1)$

$out(2) = in(2) \setminus (b,*) \cup \{(b,2)\}$

$in(3) = out(2)$

$out(3) = in(3)$

$in(4) = out(3)$

$in(6) = out(3)$

$out(4) = in(4) \setminus (m,*) \cup \{(m,4)\}$

$out(6) = in(6) \setminus (m,*) \cup \{(m,6)\}$

$in(7) = out(4) \cup out(6)$

Simple Example

System of equations

v_0 — $in(1) = \varnothing$

v_1 — $out(1) = in(1) \setminus (a,*) \cup \{(a,1)\}$

v_2 — $in(2) = out(1)$

v_3 — $out(2) = in(2) \setminus (b,*) \cup \{(b,2)\}$

v_4 — $in(3) = out(2)$

v_5 — $out(3) = in(3)$

v_6 — $in(4) = out(3)$

v_7 — $out(4) = in(4) \setminus (m,*) \cup \{(m,4)\}$

v_8 — $in(6) = out(3)$

v_9 — $out(6) = in(6) \setminus (m,*) \cup \{(m,6)\}$

v_{10} — $in(7) = out(4) \cup out(6)$

Simple Example

$$
\begin{aligned}
v_0 &= \varnothing \\
v_1 &= v_0 \setminus (a,*) \cup \{(a,1)\} \\
v_2 &= v_1 \\
v_3 &= v_2 \setminus (b,*) \cup \{(b,2)\} \\
v_4 &= v_3 \\
v_5 &= v_4 \\
v_6 &= v_5 \\
v_7 &= v_6 \setminus (m,*) \cup \{(m,4)\} \\
v_8 &= v_5 \\
v_9 &= v_8 \setminus (m,*) \cup \{(m,6)\} \\
v_{10} &= v_7 \cup v_9
\end{aligned}
$$

$\bar{\mathrm{v}}$ $\qquad$ $F(\bar{\mathrm{v}})$

$$F(\langle v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10} \rangle) =$$

$$
\begin{aligned}
\langle\ v_0' &= \varnothing, \\
v_1' &= v_0 \setminus (a,*) \cup \{(a,1)\}, \\
v_2' &= v_1, \\
v_3' &= v_2 \setminus (b,*) \cup \{(b,2)\}, \\
v_4' &= v_3, \\
v_5' &= v_4, \\
v_6' &= v_5, \\
v_7' &= v_6 \setminus (m,*) \cup \{(m,4)\}, \\
v_8' &= v_5, \\
v_9' &= v_8 \setminus (m,*) \cup \{(m,6)\}, \\
v_{10}' &= v_7 \cup v_9\ \rangle
\end{aligned}
$$

$\bar{\mathrm{v}}$ *is a solution* $\Leftrightarrow$ $\bar{\mathrm{v}} = F(\bar{\mathrm{v}})$

System of Equations

Representation as an n-ary function

$F(\langle v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\rangle) =$

$$
\begin{aligned}
\langle\ & v_0' = \varnothing, \\
& v_1' = v_0 \setminus (a,*) \cup \{(a,1)\}, \\
& v_2' = v_1, \\
& v_3' = v_2 \setminus (b,*) \cup \{(b,2)\}, \\
& v_4' = v_3, \\
& v_5' = v_4, \\
& v_6' = v_5, \\
& v_7' = v_6 \setminus (m,*) \cup \{(m,4)\}, \\
& v_8' = v_5, \\
& v_9' = v_8 \setminus (m,*) \cup \{(m,6)\}, \\
& v_{10}' = v_7 \cup v_9\ \rangle
\end{aligned}
$$

- The flow equations define a function over 11 variables $v_0 \cdots v_{10}$
- Each variable v_i represents a value from our lattice, $L = \mathcal{P}(\text{Var} \times \text{Lab})$

$F: (\mathcal{P}(\text{Var} \times \text{Lab}))^{11} \longrightarrow (\mathcal{P}(\text{Var} \times \text{Lab}))^{11}$

$\bar{\text{v}}$ *is a solution* $\Leftrightarrow$ $\bar{\text{v}} = F(\bar{\text{v}})$

Solving the Equations

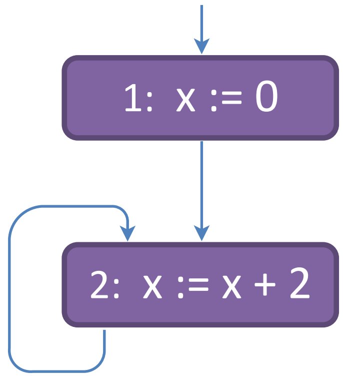
- Fixed Point Problem
  - Given a function F: L $\rightarrow$ L, find $x \in$ L such that $\boxed{F(x) = x}$

- With transfer functions, you will often find that there is more than one such solution…
  - Specifically, when the program has loops
  - We would like the ***most precise*** solution

e.g., F = identity

Solving the Equations

x := 0
while (true)
    x := x + 2



$$in(1) = \langle x \mapsto \top \rangle \qquad v_0 = \langle x \mapsto \top \rangle$$

$$out(1) = in(1)[x \mapsto \mathrm{E}] \qquad v_1 = v_0[x \mapsto \mathrm{E}]$$

$$in(2) = out(1) \sqcup out(2) \qquad v_2 = v_1 \sqcup v_3$$

$$out(2) = in(2) \qquad v_3 = v_2$$

least fixed point

$$\begin{array}{l} v_0 = \langle x \mapsto \top \rangle \\ v_1 = \langle x \mapsto \mathrm{E} \rangle \\ v_2 = \langle x \mapsto \mathrm{E} \rangle \\ v_3 = \langle x \mapsto \mathrm{E} \rangle \end{array} \quad \sqsubseteq \quad \begin{array}{l} v_0 = \langle x \mapsto \top \rangle \\ v_1 = \langle x \mapsto \mathrm{E} \rangle \\ v_2 = \langle x \mapsto \top \rangle \\ v_3 = \langle x \mapsto \top \rangle \end{array}$$

Solution #1 Solution #2

Knaster-Tarski Theorem

- Order preserving (monotonic) function:

$$x \sqsubseteq y \;\Rightarrow\; F(x) \sqsubseteq F(y)$$

- Let L be a complete lattice and F: L $\rightarrow$ L a monotonic function. Then the set of fixed points of F is also a complete lattice.

- *Definition.* the ***least fixed point*** (*lfp*) $x_\perp$ is a fixed point $\left(F(x_\perp) = x_\perp\right)$,
  such that for any x, if $F(x) = x$, then $x_\perp \sqsubseteq x$

$x_\perp$ is the minimal element ($\perp$) of the lattice from Knaster-Tarski.

Kleene Fixed-point Theorem

- Order preserving (monotonic) function:

$$x \sqsubseteq y \;\Rightarrow\; F(x) \sqsubseteq F(y)$$

- The least fixed point satisfies: $x_\perp = \sqcup\{F^n(\perp) \mid n = 0,1,2,\ldots\}$

- Proof. Let $x_i = F^i(\perp)$.
  - by induction, $x_i \sqsubseteq x_{i+1}$
  - also, $x_i \sqsubseteq x_\perp$
  - *(finite case)*
    if for some i, $x_i \underset{}{=} \overset{= F(x_i)}{x_{i+1}} \Rightarrow$ x_i is a fixed point $\Rightarrow$ $x_\perp \sqsubseteq x_i \sqsubseteq x_\perp$ $\Rightarrow$ $x_i = x_\perp$

$x_0,\ x_1,\ x_2,\ \ldots$

is called the Kleene chain of F.

BTW, same trick works for computing greatest fixed point

- in that case, start with $x_0 = \top$

Chains

- A set $S \subseteq L$ is a *chain* if

$$\forall x, y \in S.\ y \sqsubseteq x \text{ or } x \sqsubseteq y$$

- L *has no infinite chains* if every chain in L is finite.

- In that case, we are ***guaranteed*** to find the least fixed point in a finite number of steps.

Solving the Equations

F: $(\mathcal{P}(\mathsf{Var} \times \mathsf{Lab}))^{11} \longrightarrow (\mathcal{P}(\mathsf{Var} \times \mathsf{Lab}))^{11}$

$F(\langle v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\rangle) =$

$$
\begin{aligned}
\langle\ & v_0' = \varnothing, \\
& v_1' = v_0 \setminus (\mathrm{a},*) \cup \{(\mathrm{a},1)\}\ , \\
& v_2' = v_1\ , \\
& v_3' = v_2 \setminus (\mathrm{b},*) \cup \{(\mathrm{b},2)\}\ , \\
& v_4' = v_3\ , \\
& v_5' = v_4\ , \\
& v_6' = v_5\ , \\
& v_7' = v_6 \setminus (\mathrm{m},*) \cup \{(\mathrm{m},4)\}\ , \\
& v_8' = v_5\ , \\
& v_9' = v_8 \setminus (\mathrm{m},*) \cup \{(\mathrm{m},6)\}\ , \\
& v_{10}' = v_7 \cup v_9\ \rangle
\end{aligned}
$$

	$\perp$	$F(\perp)$	$F(F(\perp))$	$F(F(F(\perp)))$	$F(F(F(F(\perp))))$	$F(F(F(F(F(\perp)))))$
v_0 = in(1)	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
v_1 = out(1)	$\varnothing$	{(a,1)}	{(a,1)}	{(a,1)}	{(a,1)}	{(a,1)}
v_2 = in(2)	$\varnothing$	$\varnothing$	{(a,1)}	{(a,1)}	{(a,1)}	{(a,1)}
v_3 = out(2)	$\varnothing$	{(b,2)}	{(b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
v_4 = in(3)	$\varnothing$	$\varnothing$	{(b,2)}	{(b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
v_5 = out(3)	$\varnothing$	$\varnothing$	$\varnothing$	{(b,2)}	{(b,2)}	{(a, 1), (b,2)}
v_6 = in(4)	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	{(b,2)}	{(b,2)}
v_7 = out(4)	$\varnothing$	{(m,4)}	{(m,4)}	{(m,4)}	{(m,4)}	{(b,2), (m,4)}
v_8 = in(6)	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	{(b,2)}	{(b,2)}
v_9 = out(6)	$\varnothing$	{(m,6)}	{(m,6)}	{(m,6)}	{(m,6)}	{(b,2), (m,6)}
v_{10} = in(7)	$\varnothing$	$\varnothing$	{(m,4), (m,6)}	{(m,4), (m,6)}	{(m,4), (m,6)}	{(m,4), (m,6)}

F: $(\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{11} \rightarrow (\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{11}$

$$F(\langle v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10} \rangle) =$$

$$\langle \overset{v_0'}{\varnothing}, \overset{v_1'}{v_0 \setminus (a,*) \cup \{(a,1)\}}, \overset{v_2'}{v_1}, \overset{v_3'}{v_2 \setminus (b,*) \cup \{(b,2)\}}, \overset{v_4'}{v_3}, \overset{v_5'}{v_4}, \overset{v_6'}{v_5},$$
$$\underset{v_7'}{v_6 \setminus (m,*) \cup \{(m,4)\}}, \underset{v_8'}{v_5}, \underset{v_9'}{v_8 \setminus (m,*) \cup \{(m,6)\}}, \underset{v_{10}'}{v_7 \cup v_9} \rangle$$

	$F^5(\perp)$	$F^6(\perp)$	$F^7(\perp)$	$F^8(\perp)$
υ_0 = in(1)	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
υ_1 = out(1)	{(a,1)}	{(a,1)}	{(a,1)}	{(a,1)}
υ_2 = in(2)	{(a,1)}	{(a,1)}	{(a,1)}	{(a,1)}
υ_3 = out(2)	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
υ_4 = in(3)	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
υ_5 = out(3)	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
υ_6 = in(4)	{(b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
υ_7 = out(4)	{(b,2), (m,4)}	{(b,2), (m,4)}	{(a, 1), (b,2), (m,4)}	{(a, 1), (b,2), (m,4)}
υ_8 = in(6)	{(b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}	{(a, 1), (b,2)}
υ_9 = out(6)	{(b,2), (m,6)}	{(b,2), (m,6)}	{(a, 1), (b,2), (m,6)}	{(a, 1), (b,2), (m,6)}
υ_{10} = in(7)	{(m,4), (m,6)}	{(b,2), (m,4), (m,6)}	{(b,2), (m,4), (m,6)}	{(a, 1), (b,2), (m,4), (m,6)}

$F^9(\perp)$

$= F^8(\perp)$

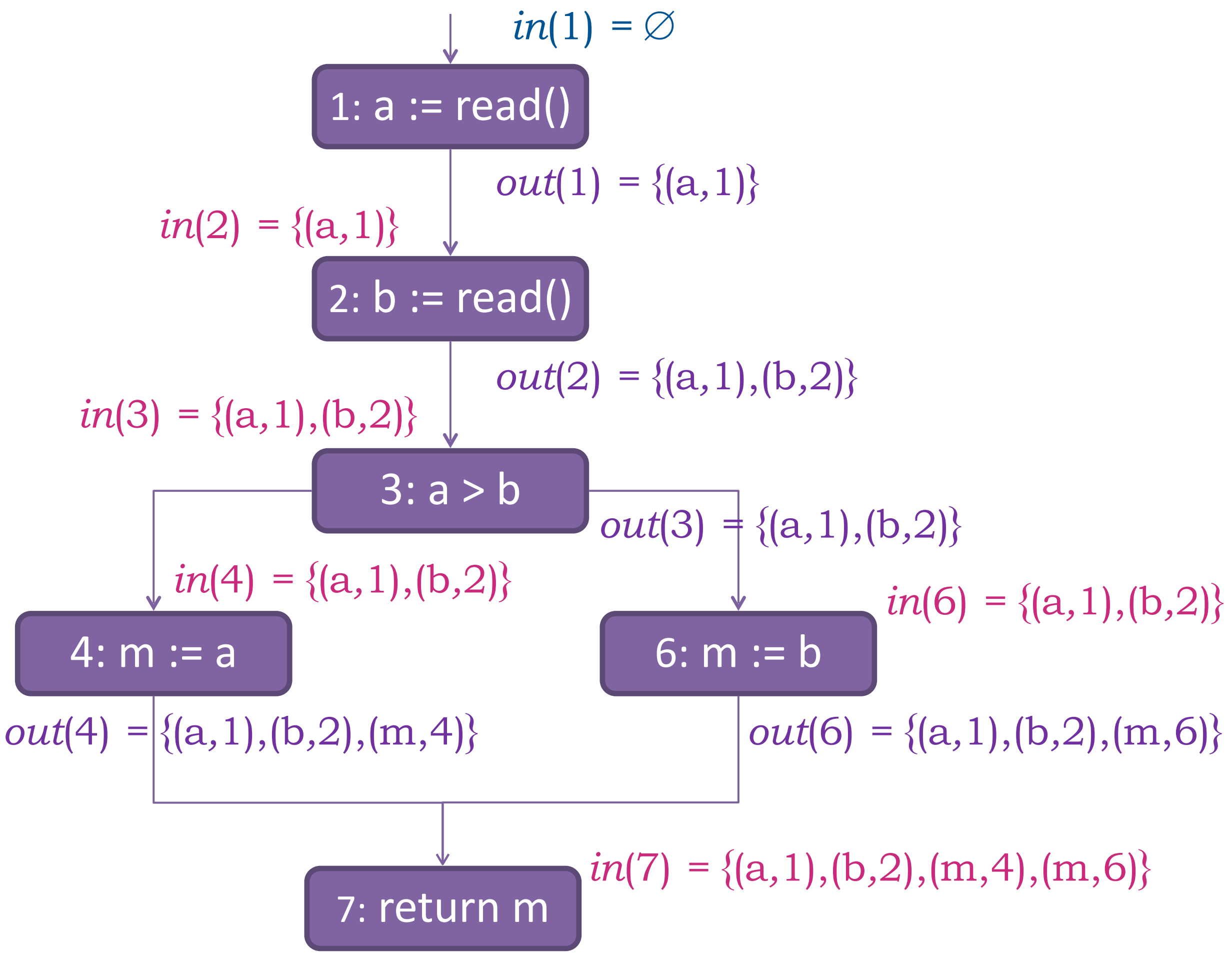
$$F\colon (\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{11} \longrightarrow (\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{11}$$

$$F(\langle \upsilon_0, \upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4, \upsilon_5, \upsilon_6, \upsilon_7, \upsilon_8, \upsilon_9, \upsilon_{10} \rangle) =$$

$$\langle \overset{\upsilon'_0}{\varnothing}, \overset{\upsilon'_1}{\upsilon_0 \setminus (a,*) \cup \{(a,1)\}}, \overset{\upsilon'_2}{\upsilon_1}, \overset{\upsilon'_3}{\upsilon_2 \setminus (b,*) \cup \{(b,2)\}}, \overset{\upsilon'_4}{\upsilon_3}, \overset{\upsilon'_5}{\upsilon_4}, \overset{\upsilon'_6}{\upsilon_5},$$
$$\underset{\upsilon'_7}{\upsilon_6 \setminus (m,*) \cup \{(m,4)\}}, \underset{\upsilon'_8}{\upsilon_5}, \underset{\upsilon'_9}{\upsilon_8 \setminus (m,*) \cup \{(m,6)\}}, \underset{\upsilon'_{10}}{\upsilon_7 \cup \upsilon_9} \rangle$$

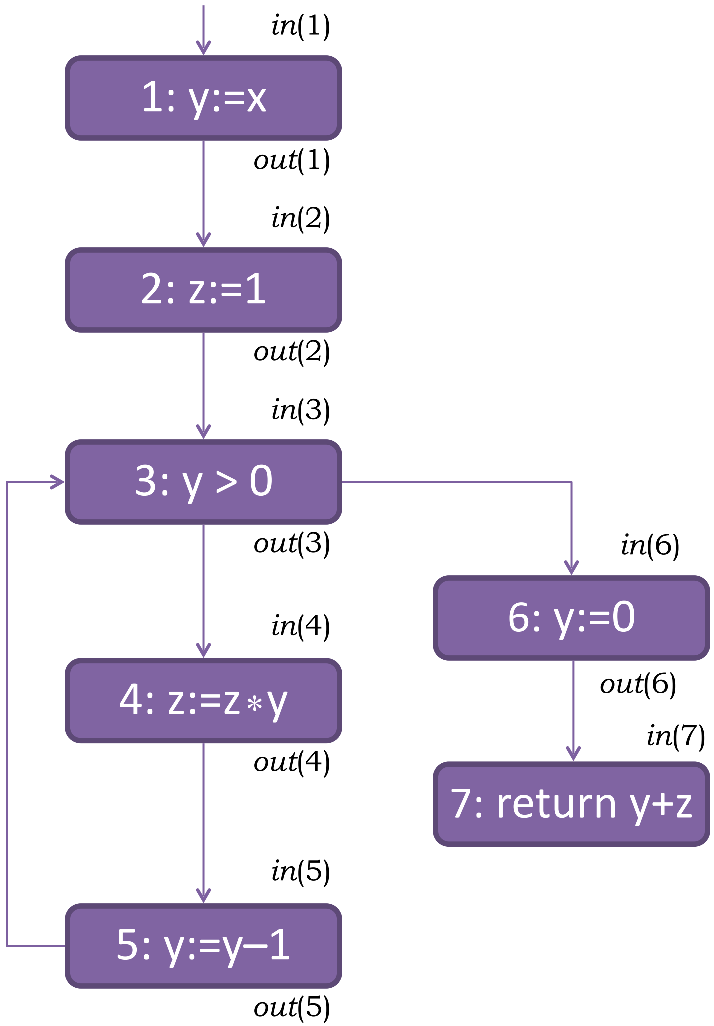
Solving the Equations

Least fixed point solution



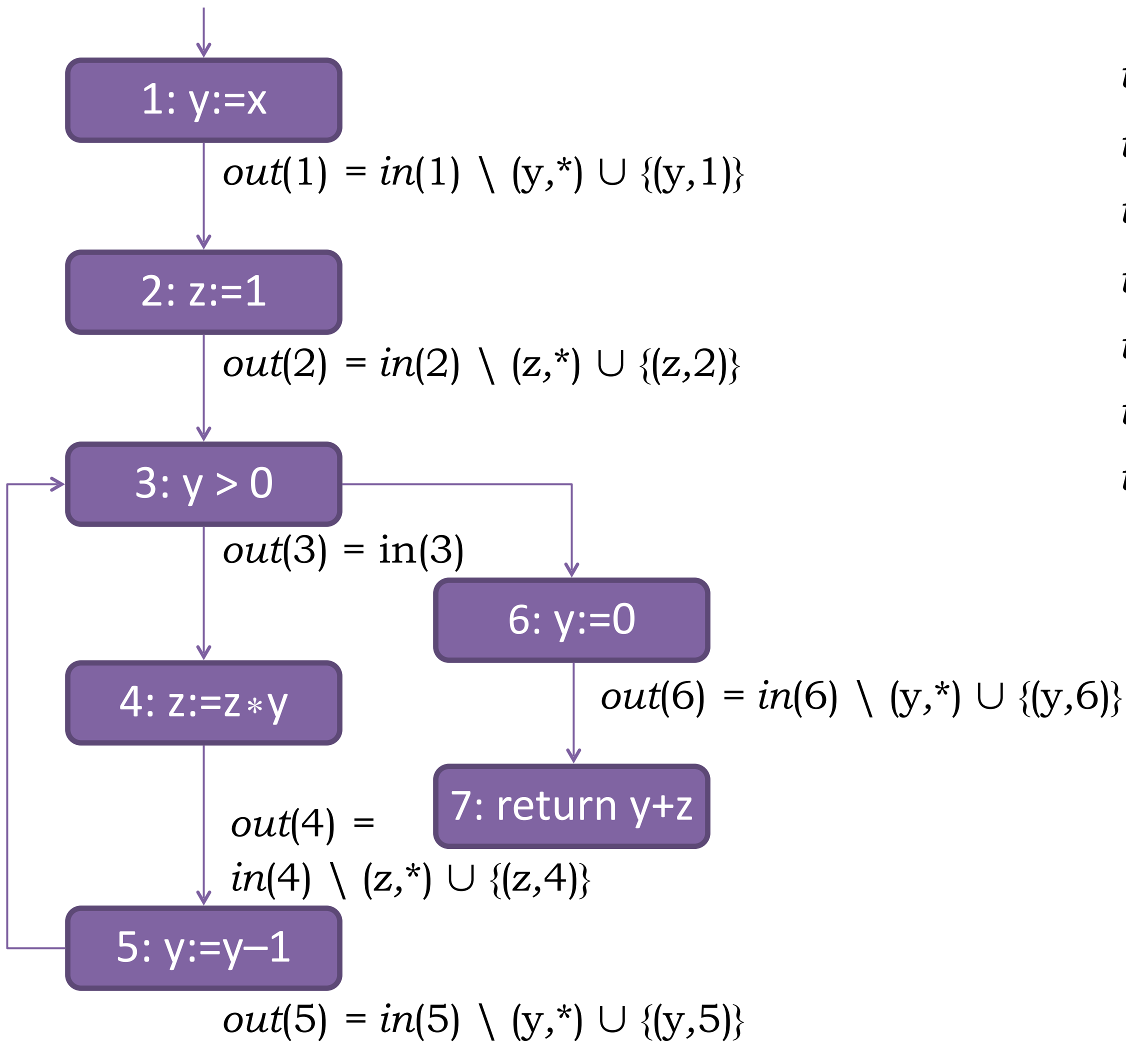

```
a := read()
b := read()
if (a > b)
    m := a
else
    m := b
return m
```

Now, to the example program from before




```
y := x
z := 1
while (y > 0) {
    z := z ∗ y
    y := y – 1
}
y := 0
return y + z
```

Transfer Functions



$in(1) = \varnothing$

$in(2) = out(1)$

$in(3) = out(2) \cup out(5)$

$in(4) = out(3)$

$in(5) = out(4)$

$in(6) = out(3)$

$in(7) = out(6)$

System of Equations

$$F(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}) =$$

v_1	$in(1) = \varnothing$	$\langle\ \varnothing,$
v_2	$in(2) = out(1)$	v_8
v_3	$in(3) = out(2) \cup out(5)$	$v_9 \cup v_{12}$
v_4	$in(4) = out(3)$	v_{10}
v_5	$in(5) = out(4)$	v_{11}
v_6	$in(6) = out(3)$	v_{10}
v_7	$in(7) = out(6)$	v_{13}
v_8	$out(1) = in(1) \setminus (y,*) \cup \{ (y,1) \}$	$v_1 \setminus (y,*) \cup \{ (y,1) \}$
v_9	$out(2) = in(2) \setminus (z,*) \cup \{ (z,2) \}$	$v_2 \setminus (z,*) \cup \{ (z,2) \}$
v_{10}	$out(3) = in(3)$	v_3
v_{11}	$out(4) = in(4) \setminus (z,*) \cup \{ (z,4) \}$	$v_4 \setminus (z,*) \cup \{ (z,4) \}$
v_{12}	$out(5) = in(5) \setminus (y,*) \cup \{ (y,5) \}$	$v_5 \setminus (y,*) \cup \{ (y,5) \}$
v_{13}	$out(6) = in(6) \setminus (y,*) \cup \{ (y,6) \}$	$v_6 \setminus (y,*) \cup \{ (y,6) \}\ \rangle$

System of Equations

Representation as an n-ary function

$$F(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}) =$$

$$\begin{aligned}
\langle\ & \emptyset, \\
& v_8 \\
& v_9 \cup v_{12} \\
& v_{10} \\
& v_{11} \\
& v_{10} \\
& v_{13} \\
& v_1 \setminus (y,*) \cup \{ (y,1) \} \\
& v_2 \setminus (z,*) \cup \{ (z,2) \} \\
& v_3 \\
& v_4 \setminus (z,*) \cup \{ (z,4) \} \\
& v_5 \setminus (y,*) \cup \{ (y,5) \} \\
& v_6 \setminus (y,*) \cup \{ (y,6) \} \rangle
\end{aligned}$$

- These equations define a function over 13 variables ($in(1..7)$, $out(1..6)$)
- Each variable represents a value from our lattice, $L = \mathcal{P}(\text{Var} \times \text{Lab})$

F: $(\mathcal{P}(\text{Var} \times \text{Lab}))^{13} \longrightarrow (\mathcal{P}(\text{Var} \times \text{Lab}))^{13}$

A solution $\bar{v}$ satisfies $F(\bar{v}) = \bar{v}$

	$\perp$	F($\perp$)	F(F($\perp$))	F(F(F($\perp$)))	F(F(F(F($\perp$))))	F(F(F(F(F($\perp$)))))
in(1)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
in(2)	$\emptyset$	$\emptyset$	{(y,1)}	{(y,1)}	{(y,1)}	{(y,1)}
in(3)	$\emptyset$	$\emptyset$	{(z,2),(y,5)}	{(z,2),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
in(4)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{(z,2),(y,5)}	{(z,2),(y,5)}
in(5)	$\emptyset$	$\emptyset$	{(z,4)}	{(z,4)}	{(z,4)}	{(z,4)}
in(6)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{(z,2),(y,5)}	{(z,2),(y,5)}
in(7)	$\emptyset$	$\emptyset$	{(y,6)}	{(y,6)}	{(y,6)}	{(y,6)}
out(1)	$\emptyset$	{(y,1)}	{(y,1)}	{(y,1)}	{(y,1)}	{(y,1)}
out(2)	$\emptyset$	{(z,2)}	{(z,2)}	{(z,2),(y,1)}	{(z,2),(y,1)}	{(z,2),(y,1)}
out(3)	$\emptyset$	$\emptyset$	$\emptyset$	{(z,2),(y,5)}	{(z,2),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
out(4)	$\emptyset$	{(z,4)}	{(z,4)}	{(z,4)}	{(z,4)}	{(z,4)}
out(5)	$\emptyset$	{(y,5)}	{(y,5)}	{(z,4),(y,5)}	{(z,4),(y,5)}	{(z,4),(y,5)}
out(6)	$\emptyset$	{(y,6)}	{(y,6)}	{(y,6)}	{(y,6)}	{(z,2),(y,6)}

F: $(\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{13} \rightarrow (\mathcal{P}(\mathrm{Var} \times \mathrm{Lab}))^{13}$

in(1)=$\emptyset$ in(2)=out(1) in(3)=out(2) U out(5) in(4)=out(3) in(5)=out(4) in(6) = out(3)

out(1) = in(1) \ (y,*) U { (y,1) } out(2) = in(2) \ (z,*) U { (z,2) } in(7) = out(6)

out(3) = in(3) out(4) = in(4) \ (z,*) U { (z,4) }

out(5) = in(5) \ (y,*) U { (y,5) } out(6) = in(6) \ (y,*) U { (y,6) }

	$F(F(F(F(F(\perp)))))$	$F^6(\perp)$	$F^7(\perp)$	$F^8(\perp)$
in(1)	∅	∅	∅	∅
in(2)	{(y,1)}	{(y,1)}	{(y,1)}	{(y,1)}
in(3)	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
in(4)	{(z,2),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
in(5)	{(z,4)}	{(z,4)}	{(z,4),(y,1),(y,5)}	{(z,4),(y,1),(y,5)}
in(6)	{(z,2),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
in(7)	{(y,6)}	{(y,6)}	{(y,6)}	{(y,6)}
out(1)	{(y,1)}	{(y,1)}	{(y,1)}	{(y,1)}
out(2)	{(z,2),(y,1)}	{(z,2),(y,1)}	{(z,2),(y,1)}	{(z,2),(y,1)}
out(3)	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}	{(z,2),(z,4),(y,1),(y,5)}
out(4)	{(z,4)}	{(z,4),(y,1),(y,5)}	{(z,4),(y,1),(y,5)}	{(z,4),(y,1),(y,5)}
out(5)	{(z,4),(y,5)}	{(z,4),(y,5)}	{(z,4),(y,5)}	{(z,4),(y,5)}
out(6)	{(z,2),(y,6)}	{(z,2),(y,6)}	{(z,2),(y,6)}	{(z,2),(y,6)}

…

$$F: (\mathcal{P}(\text{Var} \times \text{Lab}))^{13} \rightarrow (\mathcal{P}(\text{Var} \times \text{Lab}))^{13}$$

in(1)=∅ in(2)=out(1) in(3)=out(2) U out(5) in(4)=out(3) in(5)=out(4) in(6) = out(3)

out(1) = in(1) \ (y,*) U { (y,1) }

out(2) = in(2) \ (z,*) U { (z,2) }

in(7) = out(6)

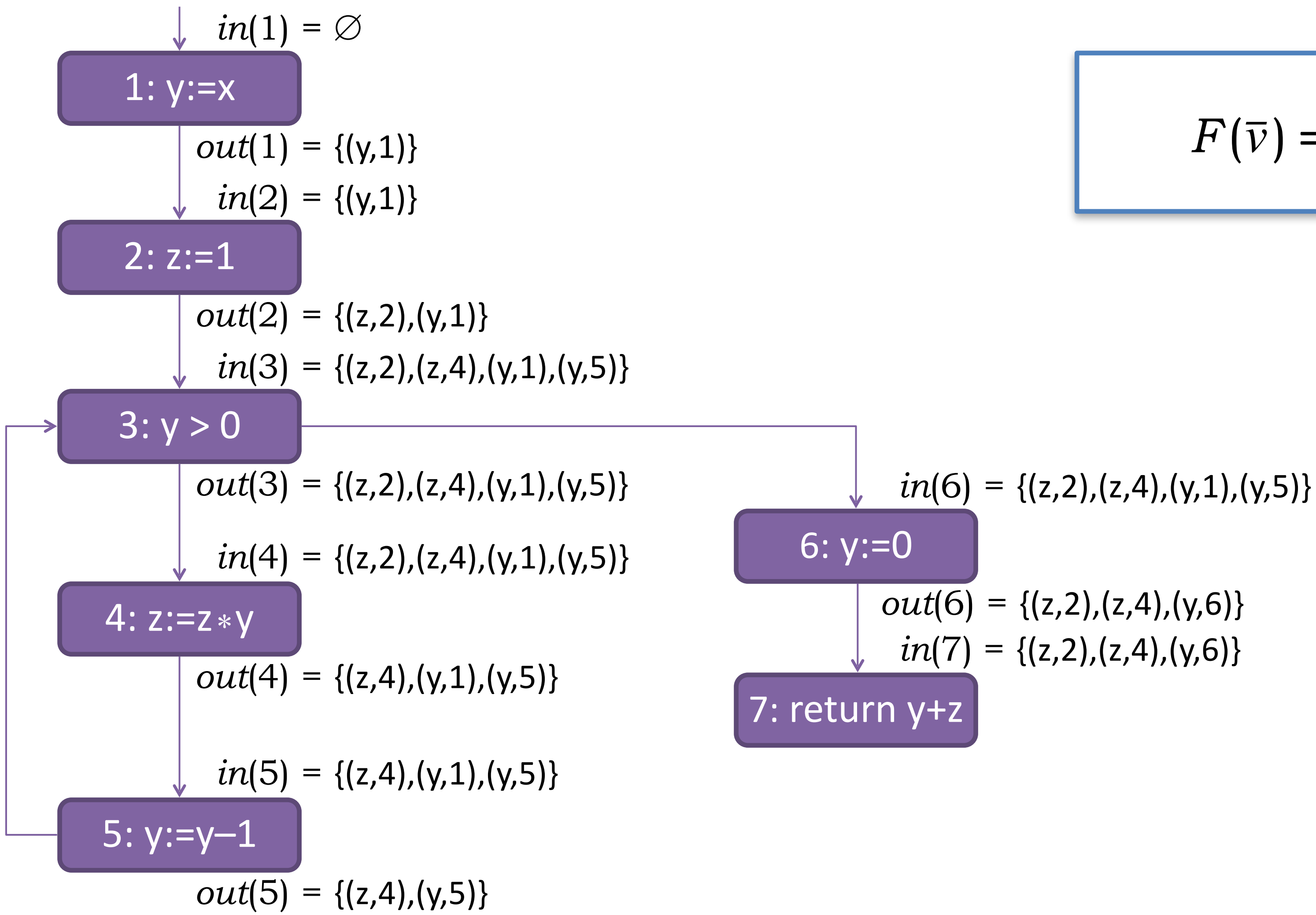
out(3) = in(3)

out(4) = in(4) \ (z,*) U { (z,4) }

out(5) = in(5) \ (y,*) U { (y,5) }

out(6) = in(6) \ (y,*) U { (y,6) }

Least Fixed Point Solution



$$F(\overline{v}) = \overline{v}$$

Chaotic Iterations

- To avoid recomputing values that do not change:
  - Keep a work list of CFG nodes to update
    - start with work list = {*entry*}
  - Pick one node at a time $u \in$ work list
  - Update $out(u)$ from $in(u)$
  - If $out(u)$ has changed, then for all successors v of u;
    - recompute $in(v) = out(u)$
    - add v to the work list
  - Repeat until work list = $\varnothing$

Chaotic Iterations: Example

Initially: work list = {1}

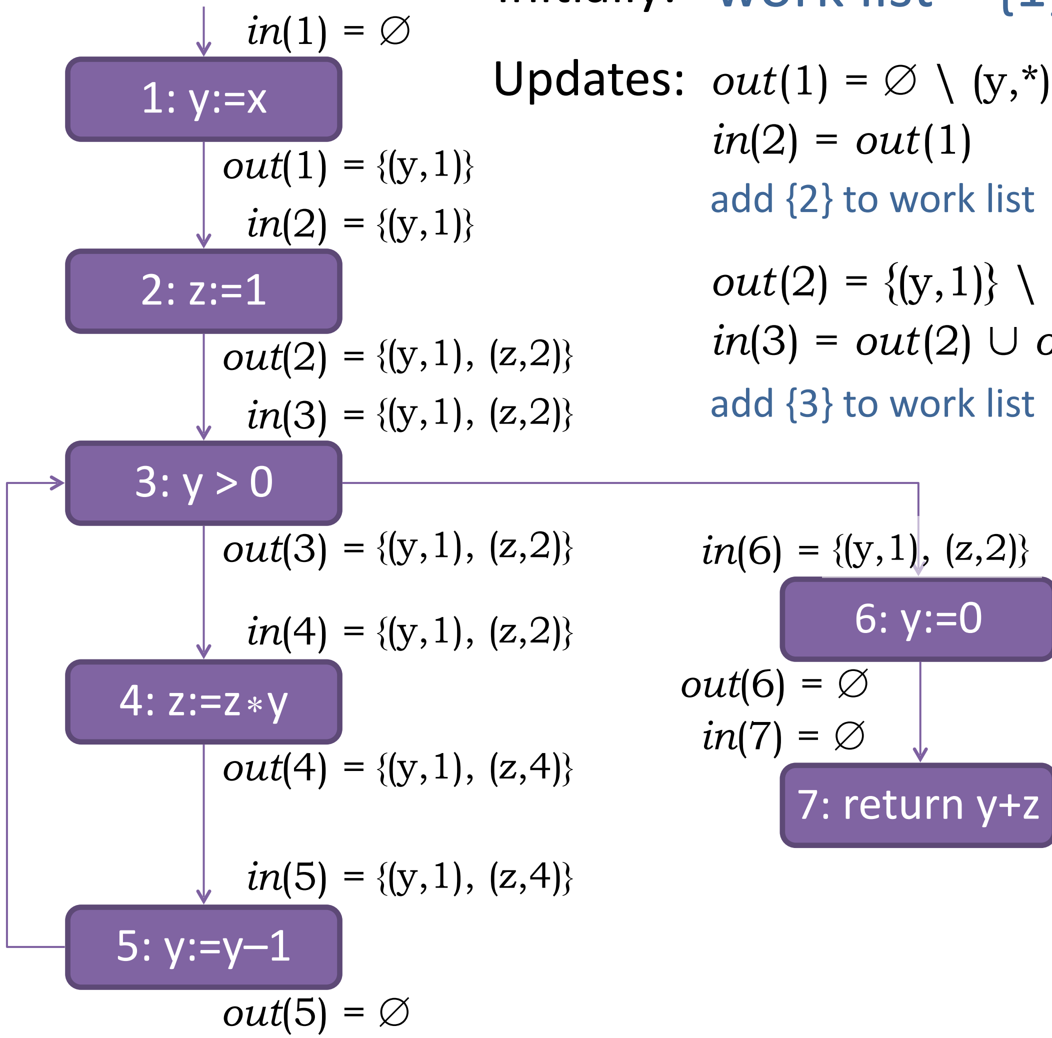
Updates: $out(1) = \emptyset \setminus (y,*) \cup \{(y,1)\}$
$in(2) = out(1)$
add {2} to work list

$out(2) = \{(y,1)\} \setminus (z,*) \cup \{(z,2)\}$
$in(3) = out(2) \cup out(5)$
add {3} to work list

$out(3) = in(3)$
$in(4) = out(3)$
$in(6) = out(3)$
add {4,6} to work list

$out(4) = \{(y,1),(z,2)\} \setminus (z,*) \cup \{(z,4)\}$
$in(5) = out(4)$
add {5} to work list

…



$out(1) = in(1) \setminus (y,*) \cup \{ (y,1) \}$
$out(2) = in(2) \setminus (z,*) \cup \{ (z,2) \}$
$out(3) = in(3)$
$out(4) = in(4) \setminus (z,*) \cup \{ (z,4) \}$
$out(5) = in(5) \setminus (y,*) \cup \{ (y,5) \}$
$out(6) = in(6) \setminus (y,*) \cup \{ (y,6) \}$

Using Reaching-Definitions Information

- <u>Remember:</u> this is an over-approximation
  - A definition ***may*** be reaching use
  - We may err, but **always on the safe side**
    - We may say that a definition ***may*** reach a program point when it doesn't
    - We **never miss** a definition that ***may*** reach a point
- Usage examples
  - detecting **possible** use before any definition
  - very simple *constant folding*
  - transforming IR to SSA form (*e.g.* for LLVM)
  - *useful for debugging in IDEs*

by setting **initial** state to { (x,**?**) | x ∈ Vars }

Using Reaching-Definitions Information

detecting possible use before any definition

```
1   y := x                    ------------------ in(1) = {(x,?), (y,?), (z,?)}
2   while (y > 0) {
3       z := z * y            ------- in(3) = {(y,1), (y,4),(x,?), (z,?), (z,3)}
4       y := y – 1
    }
5   y := 0
6   return y + z
```

use of z

When a definition (*v*,?) for some v reaches any use of *v* in the program, issue a warning

Using Reaching-Definitions Information

very simple *constant folding*

```
1   y := x                    ------------------ in(1) = ∅
2   z := x := 1
3   while (y > 0) {
4       z := z * y
5       y := y − x            ------ in(5) = {(y,1), (y,4), (x,2), (z,4)}
    }
6   y := 0
7   return y + z
```

use of x

When the **only** definition (v,i) of some v that reaches some use of v in the program is a constant assignment, the use of v can be replaced by the constant

Using Reaching-Definitions Information

transforming IR to SSA form (*e.g.* for LLVM)

```
cond:
   %b = icmp slt i32 %i, %j
   br i1 %b, label %then,
             label %else
then:
   %max = or i32 0, %j
   br label %exit

else:
   %max = or i32 0, %i
   br label %exit

exit:
   ret i32 %max
```

$in(\text{exit}) = \{(\text{b, cond}), (\text{max, then}), (\text{max, else})\}$

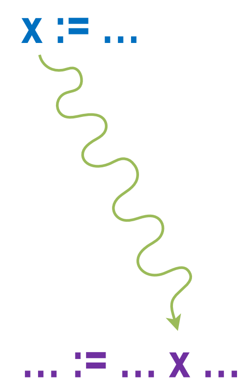
```
cond:
   %b = icmp slt i32 %i, %j
   br i1 %b, label %then,
             label %else
then:
   %max_then = or i32 0, %j
   br label %exit

else:
   %max_else = or i32 0, %i
   br label %exit

exit:
   %max_exit =
     phi i32 [ %max_then, %then ],
             [ %max_else, %else ]
   ret i32 %max_exit
```

Live Variables

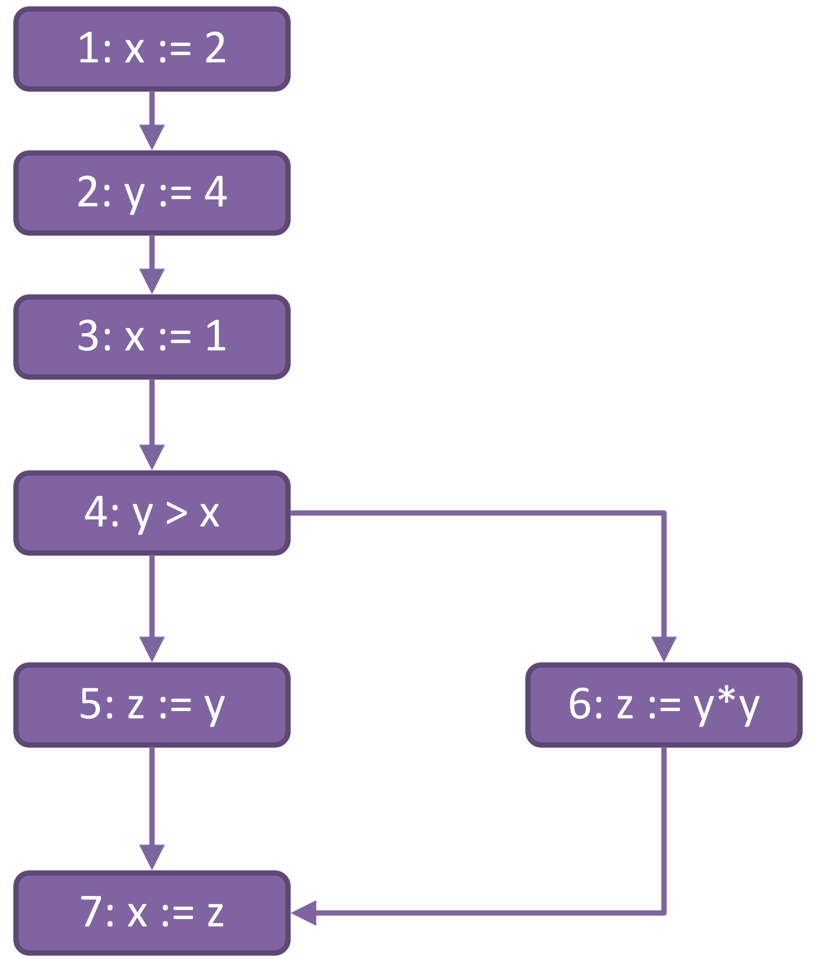
```
                     - - - - - - - - -  x dead,  y dead,  z dead
1:  x := 2;
                     - - - - - - - - -  x dead,  y dead,  z dead
2:  y := 4;
                     - - - - - - - - -  x dead,  y live,  z dead
3:  x := 1;
                     - - - - - - - - -  x live,  y live,  z dead
4:  if y > x
                     - - - - - - - - -  x dead,  y live,  z dead
5:        then z := y
6:        else  z := y * y;
7:  x := z
```



For each program point, which variables may be live (*i.e.*, has some future use before re-definition, along some path) at the exit from that point.

Live Variables

```
1:  x := 2;
2:  y := 4;
3:  x := 1;
4:  if y > x
5:          then z := y
6:          else  z := y * y;
7:  x := z
```



- Backward Analysis (!)

Live Variables

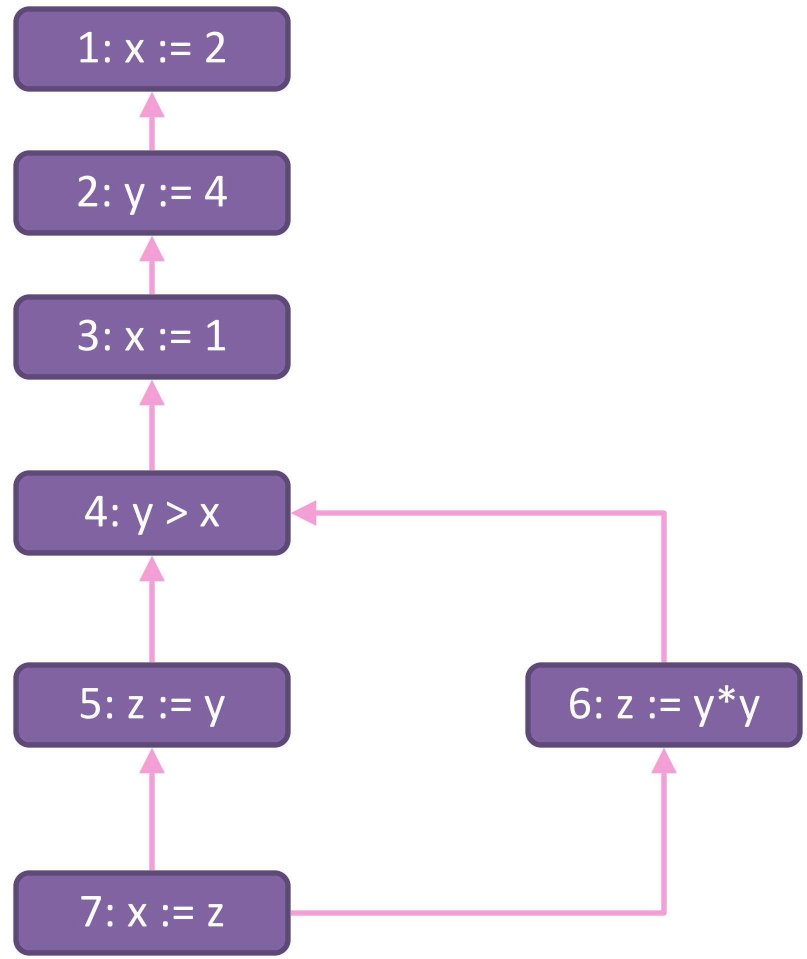
```
1:  x := 2;
2:  y := 4;
3:  x := 1;
4:  if y > x
5:        then z := y
6:        else z := y * y;
7:  x := z
```

FV: Expr $\longrightarrow \mathcal{P}$(Var)

- Variables used in an expression

Stmt	out(ℓ)
x := *expr*	in(ℓ) \ { x } ∪ FV(*expr*)
if *cond*	in(ℓ) ∪ FV(*cond*)

Transfer functions



Live Variables — Solution

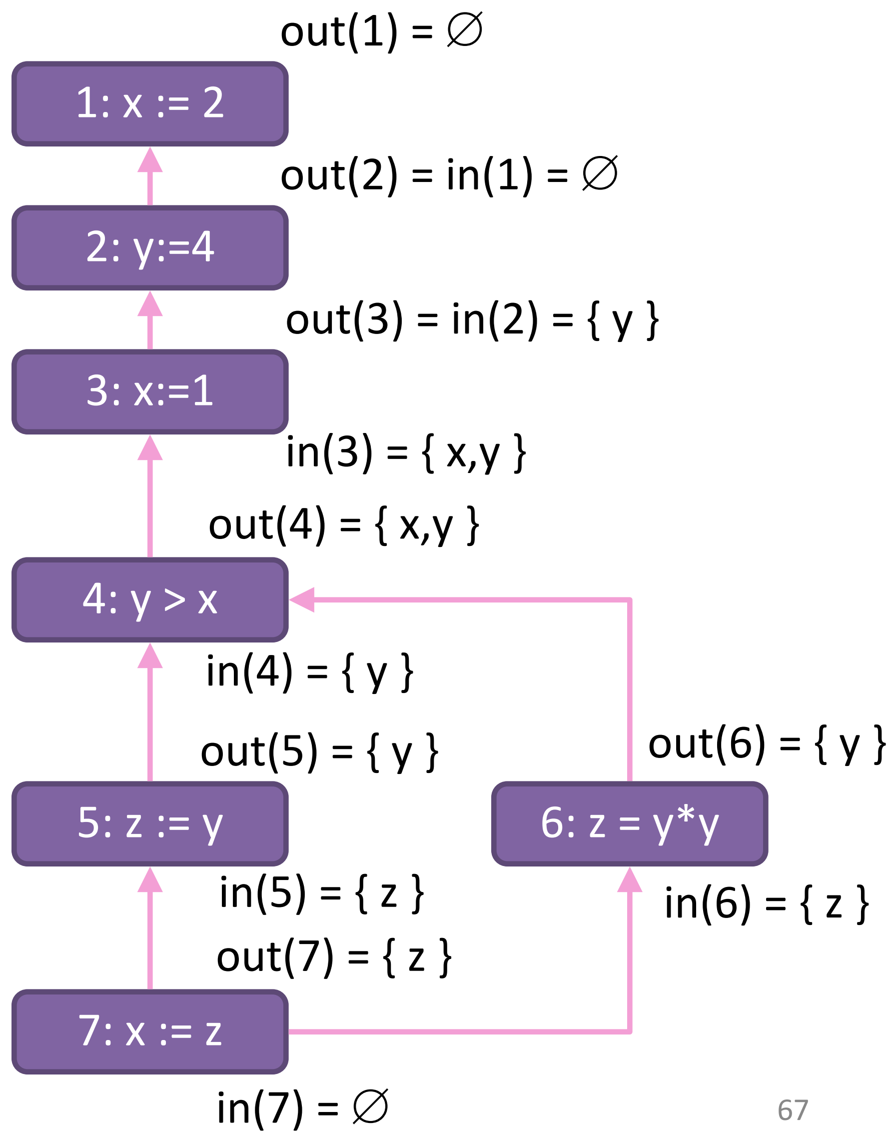
1: x := 2;
2: y := 4;
3: x := 1;
4: if y > x
5: then z := y
6: else z := y * y;
7: x := z

FV: Expr $\longrightarrow \mathcal{P}$(Var)

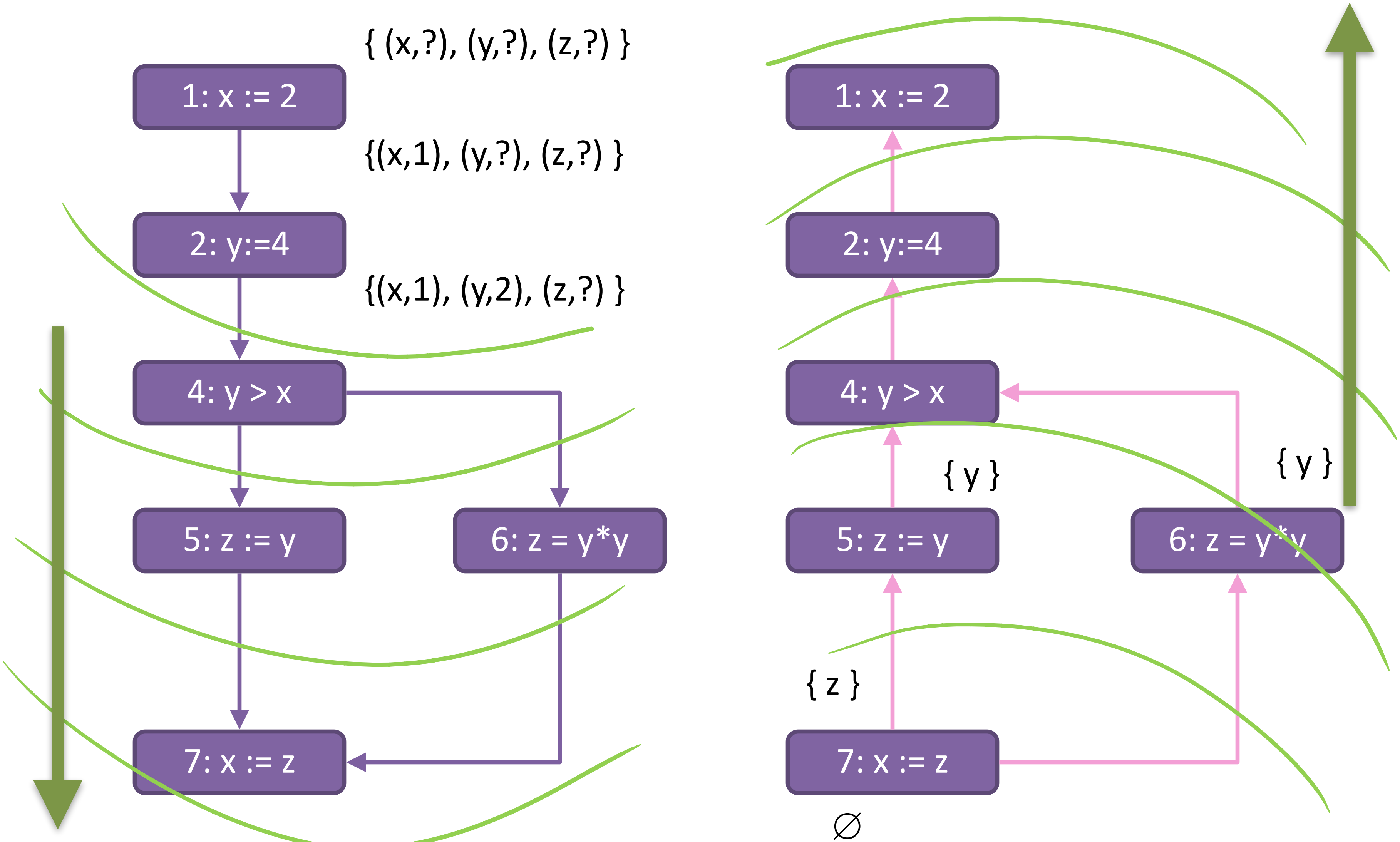
- Variables used in an expression

Stmt	out(ℓ)
x := *expr*	in(ℓ) \ { x } ∪ FV(*expr*)
if *cond*	in(ℓ) ∪ FV(*cond*)

Transfer functions



Forward vs. Backward Analyses



Kill/Gen

Statement	out(ℓ)
x := *expr*	in(ℓ) \ { x } ∪ FV(*expr*)
skip	in(ℓ)
if *cond*	in(ℓ) ∪ FV(*cond*)

Statement	kill	gen
x := *expr*	{ x }	FV(*expr*)
skip	∅	∅
if *cond*	∅	FV(*cond*)

$$\text{out}(\ell) = \text{in}(\ell) \setminus \text{kill}(B^\ell) \cup \text{gen}(B^\ell)$$

B^ℓ = statement (or block) at label ℓ

Available Expressions Analysis

```
1   x = a + b
2   y = a * b
3   while (y > a + b) {
4       a = a + 1
5       x = a + b
    }
```

(a + b) always available
at label 3

For each program point, find which **expressions** must have already been computed, and not later modified, on all paths leading to that program point

Some Required Notation

- Classes of expressions:
  - AExpr – arithmetic expressions
  - BExpr – boolean expressions
- FV: (AExpr ∪ BExpr) $\longrightarrow \mathcal{P}$(Var)
  - Variables used in an expression
- AExpr(*e*) = all (non-atomic) arithmetic sub-expressions of an expression *e*

Available Expressions Analysis

- Property domain
  - $\mathrm{L} = \mathcal{P}(\text{AExpr})$; $\sqsubseteq\ =\ \supseteq$; $\sqcup\ =\ \cap$
  - in, out: Lab $\rightarrow \mathrm{L}$
    Map a statement label to set of arithmetic expressions that are available at (before, after) that statement

- Dataflow equations
  - Flow equations – how to join incoming dataflow facts
  - Effect equations – given an input set of expressions $in(i)$, what is the effect of the statement at i

Available Expressions Analysis

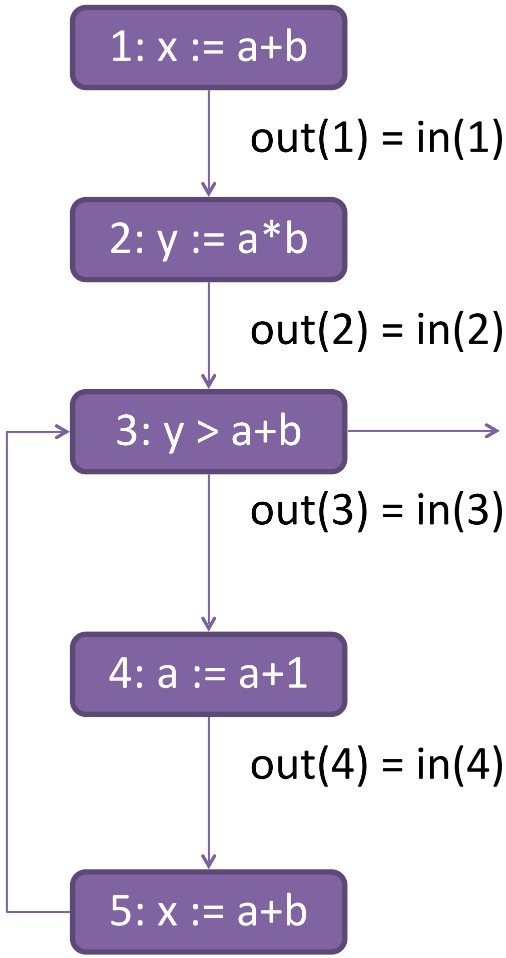
- in(ℓ) =
  - $\varnothing$ when ℓ is the initial label
  - $\bigcap$ {out(ℓ') | $\ell' \in$ pred(ℓ)} otherwise

As dictated by the monotone framework

- out(ℓ) =

Statement	out(ℓ)
x = *expr*	in(ℓ) \ $\underbrace{\{ e \in \text{AExpr} \mid x \in FV(e) \}}_{\text{כל הביטויים האריתמטיים שבהם x נמצא מצד ימין}}$ ∪ $\underbrace{\{ e \in \text{AExpr}(expr) \mid x \notin FV(e) \}}_{\text{כל התת ביטויים של צד ימין כך ש x לא נמצא בהם}}$
skip	in(ℓ)
if *cond*	in(ℓ) ∪ AExpr(*cond*)

Transfer Functions



$out(1) = in(1) \setminus \varnothing \cup \{ a+b \}$

$out(2) = in(2) \setminus \varnothing \cup \{ a*b \}$

$out(3) = in(3) \setminus \varnothing \cup \{ a+b \}$

$out(4) = in(4) \setminus \{ a+b, a*b, a+1 \} \cup \varnothing$

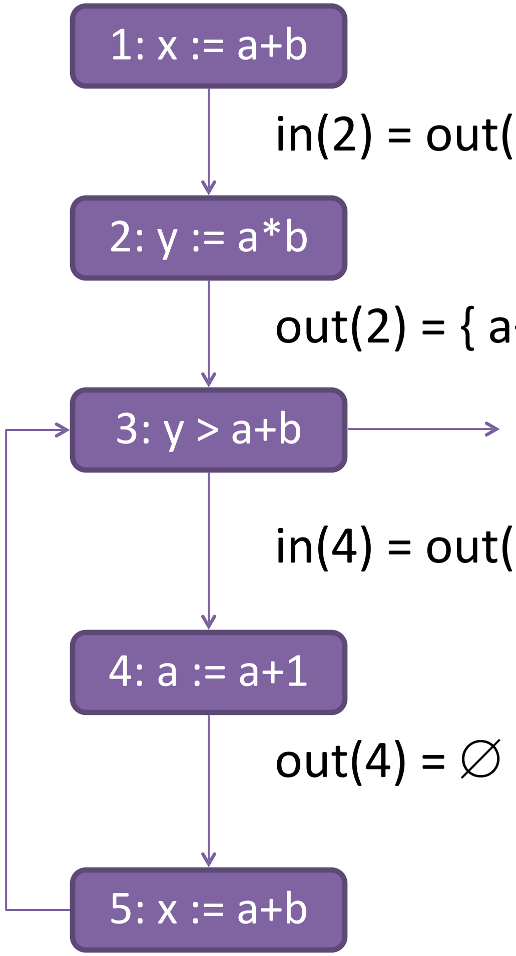
$out(5) = in(5) \setminus \varnothing \cup \{ a+b \}$

$in(1) = \varnothing$
$in(2) = out(1)$
$in(3) = out(2) \cap out(5)$
$in(4) = out(3)$
$in(5) = out(4)$

```
1   x := a + b
2   y := a * b
3   while (y > a + b) {
4       a := a + 1
5       x := a + b
    }
```

Solution

in(1) = ∅



in(2) = out(1) = { a + b }

out(2) = { a+b, a*b } in(3) = { a + b }

in(4) = out(3) = { a+b }

out(4) = ∅

out(5) = { a+b }

Kill/Gen

Statement	out (ℓ)
x := *expr*	in(ℓ) \ { *e* ∈ AExpr | x ∈ FV(*e*) } U { e ∈ AExpr(*expr*) | x ∉ FV(*e*) }
skip	in(ℓ)
if *cond*	in(ℓ) U AExpr(*cond*)

Statement	kill	gen
x := *expr*	{ *e* ∈ AExpr | x ∈ FV(*e*) }	{ e ∈ AExpr(*expr*) | x ∉ FV(*e*) }
skip	∅	∅
if *cond*	∅	AExpr(*cond*)

out(ℓ) = in(ℓ) \ kill(B^ℓ) U gen(B^ℓ)

B^ℓ = statement (or block) at label ℓ

Reaching Definitions Revisited

Statement	**out(ℓ)**
x := *expr*	in(ℓ) \ { (x,i) | $i \in$ Lab } U { (x,ℓ) }
skip	in(ℓ)
if *cond*	in(ℓ)

Statement	**kill**	**gen**
x := *expr*	{ (x,i) | $i \in$ Lab }	{ (x,ℓ) }
skip	$\varnothing$	$\varnothing$
if *cond*	$\varnothing$	$\varnothing$

out(ℓ) = in(ℓ) \ kill(B^ℓ) U gen(B^ℓ)

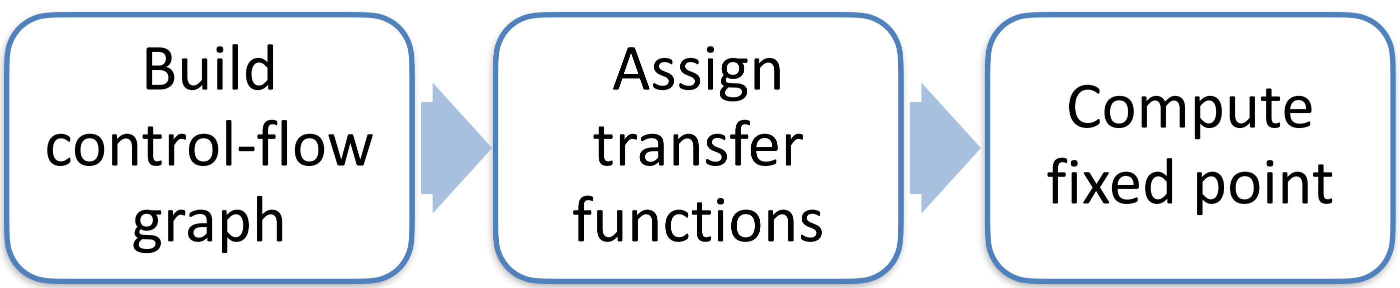
B^ℓ = statement (or block) at label ℓ

Analyses Summary

	(may) Reaching Definitions	(must) Available Expressions	(may) Live Variables
L	$\mathcal{P}(\text{Var} \times \text{Lab})$	$\mathcal{P}(\text{AExp})$	$\mathcal{P}(\text{Var})$
$\sqsubseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cup$	$\cap$	$\cup$
$\top$	Var × Lab	$\varnothing$	Var
$\perp$	$\varnothing$	AExp	$\varnothing$
Initial	$\{(x,?) \mid x \in \text{Globals}\}$	$\varnothing$	$\varnothing$
Entry labels	{ init }	{ init }	final
Direction	forward	forward	backward
f_ℓ	$f_\ell(\text{val}) = (\text{val} \setminus \text{kill}_\ell) \cup \text{gen}_\ell$		

Summary

- Static Analysis
  - ✓ Prove properties of a program at compile time
  - ✓ Over-approximate possible program behaviors
- Dataflow Analysis



- Monotone Framework
  - ✓ Can be used to express many useful analyses, in particular kill/gen-type analyses

Coming Up