Another example: constant prop
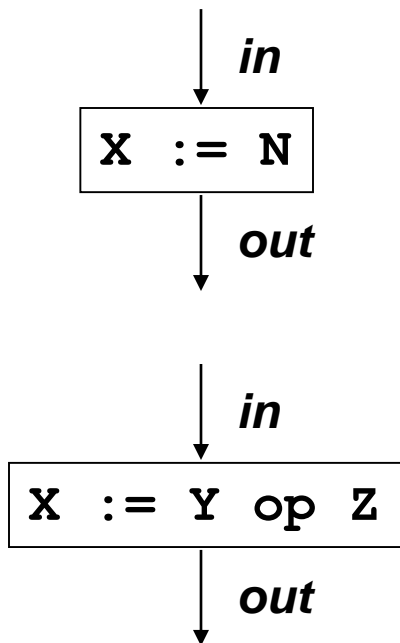
- Set D = $\mathcal{P}(\{ x \rightarrow N \mid x \in Var, N \in Int \})$

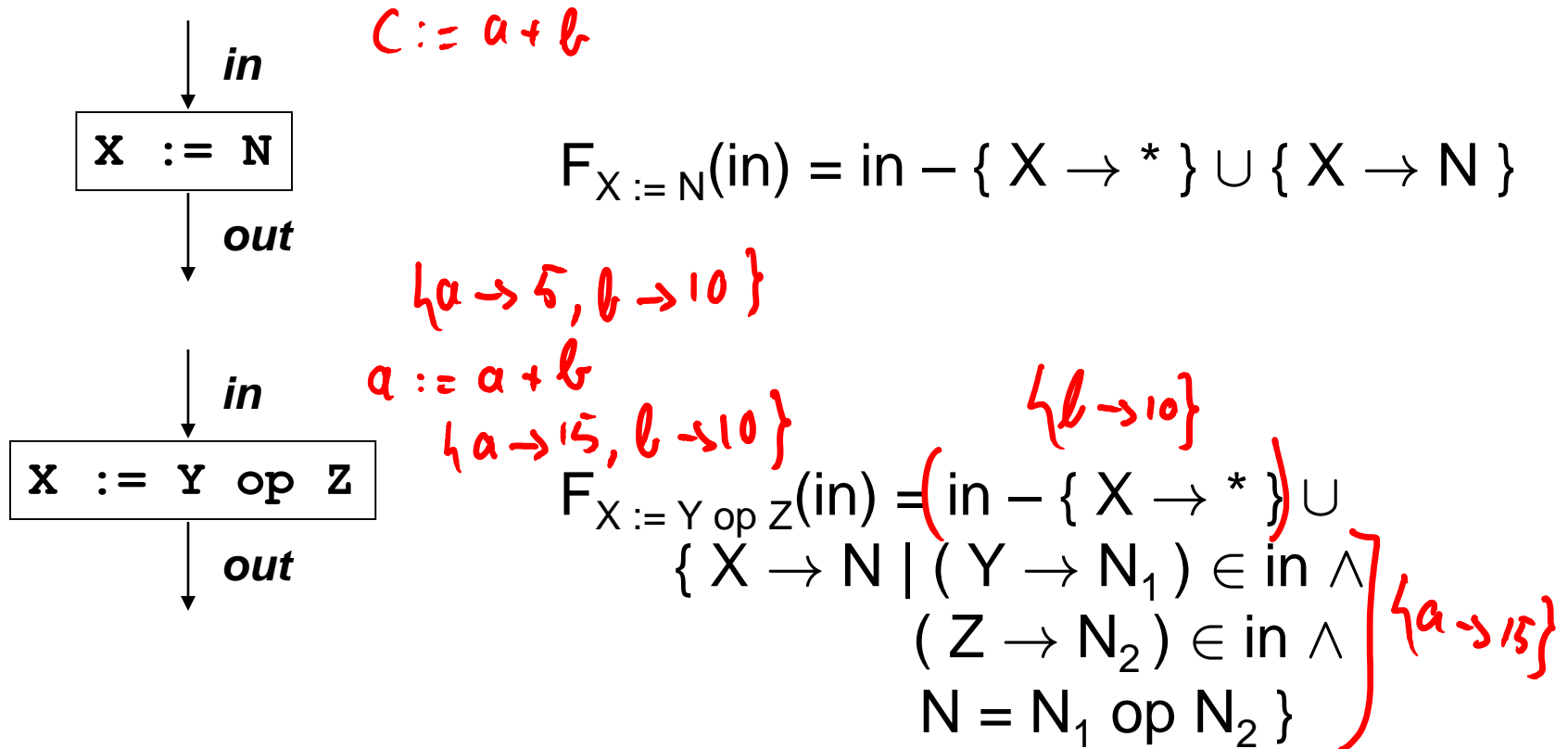$$F_{X := N}(in) = in - \{ x \rightarrow * \} \cup \{ x \rightarrow N \}$$

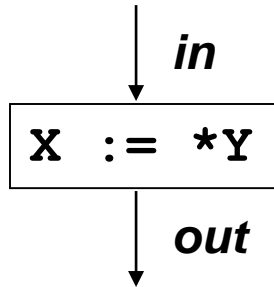$$F_{X := Y \, op \, Z}(in) = in - \{ x \rightarrow * \} \cup \{ x \rightarrow "N_1 \, op \, N_2" \mid Y \rightarrow N_1 \in in \wedge Z \rightarrow N_2 \in in \}$$

Another example: constant prop

- Set $D = 2^{\{\, x \rightarrow N \mid x \in \text{Vars} \wedge N \in Z \,\}}$

in

$C := a + b$

X := N

out

$$F_{X := N}(in) = in - \{\, X \rightarrow * \,\} \cup \{\, X \rightarrow N \,\}$$

$\{a \rightarrow 5, b \rightarrow 10\}$

in

$a := a + b$

$\{a \rightarrow 15, b \rightarrow 10\}$

$\{b \rightarrow 10\}$

X := Y op Z

$$F_{X := Y \text{ op } Z}(in) = \left(in - \{\, X \rightarrow * \,\} \right) \cup$$
$$\{\, X \rightarrow N \mid (\, Y \rightarrow N_1 \,) \in in \wedge$$
$$(\, Z \rightarrow N_2 \,) \in in \wedge$$
$$N = N_1 \text{ op } N_2 \,\}$$

$\{a \rightarrow 15\}$

out

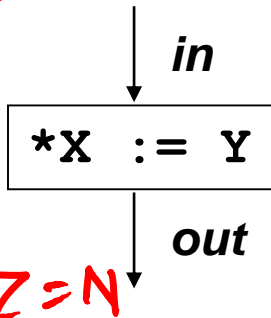# Another example: constant prop

$$F_{X := *Y}(in) = in - \{X \to *\} \cup$$

$$\{X \to N \mid \forall V \in may\text{-}pt(Y) \quad V \to N \in in\}$$

may-pt(Y)

$$F_{*X := Y}(in) = in - \{Z \to * \mid Z \in mpt(x)\} \cup$$

Z = M   Y = N

Z = N

Another example: constant prop

in

$\boxed{\texttt{X := *Y}}$

out
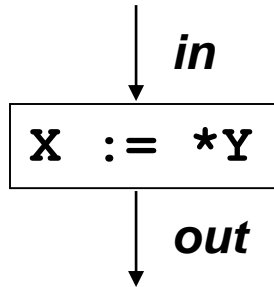
$F_{X := *Y}(in) = in - \{ X \rightarrow * \}$
$\cup \{ X \rightarrow N \mid \forall Z \in \text{may-point-to}(Y) .$
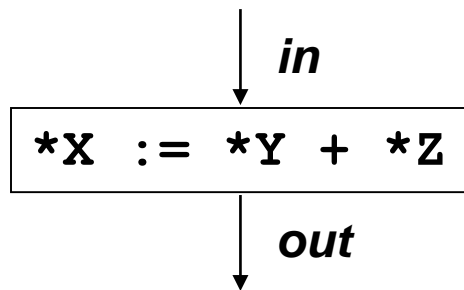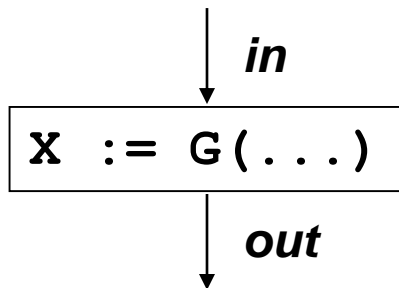$(Z \rightarrow N) \in in \}$

in

$\boxed{\texttt{*X := Y}}$

out
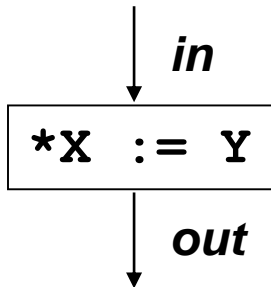
$F_{*X := Y}(in) = in - \{ Z \rightarrow * \mid Z \in \text{may-point}(X) \}$
$\cup \{ Z \rightarrow N \mid Z \in \text{must-point-to}(X) \wedge$
$Y \rightarrow N \in in \}$
$\cup \{ Z \rightarrow N \mid (Y \rightarrow N) \in in \wedge$
$(Z \rightarrow N) \in in \}$

Another example: constant prop

$$*X := *Y + *Z$$

$$F_{*X := *Y + *Z}(\text{in}) =$$

in

out

$$X := G(\ldots)$$

$$F_{X := G(\ldots)}(\text{in}) =$$

in

out

Another example: constant prop

$$\text{*X := *Y + *Z}$$

in

out

$$F_{\text{*X := *Y + *Z}}(in) = F_{\text{a := *Y;b := *Z;c := a + b; *X := c}}(in)$$

$$\text{X := G(...)}$$

in

out

$$F_{\text{X := G(...)}}(in) = \emptyset$$

Another example: constant prop

```
          ↓ in
┌──────────────────┐
│ s: if (...)      │
└──────────────────┘
 out[0] /        \ out[1]
       ↓          ↓



  x->5          x->10


 in[0] \        / in[1]
        ↓      ↓
┌──────────────────┐
│      merge       │   ∩
└──────────────────┘
        ↓ out
```

Lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$

  $|, \supseteq, AS, \emptyset, \cap, \cup$

  2AS

  $\emptyset$

  $\{ \wedge \} \sqcup$

  AS
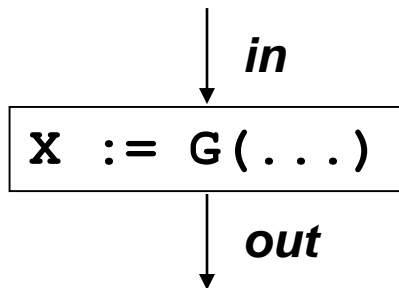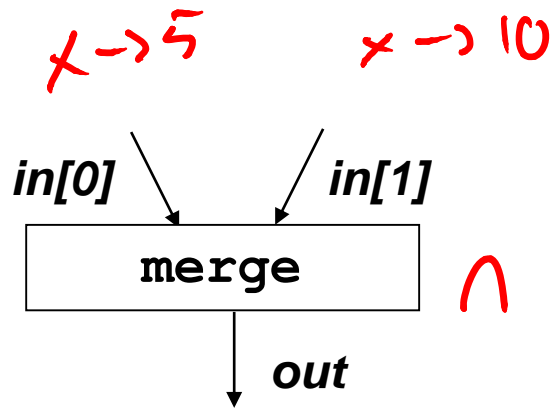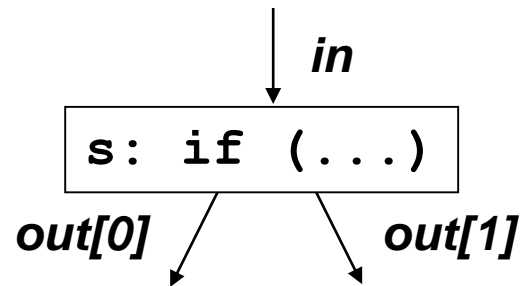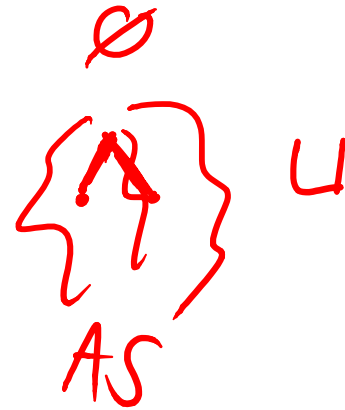
  $AS = \{ x \rightarrow N \mid x \in \text{Var}, N \in \text{Int} \}$

Lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
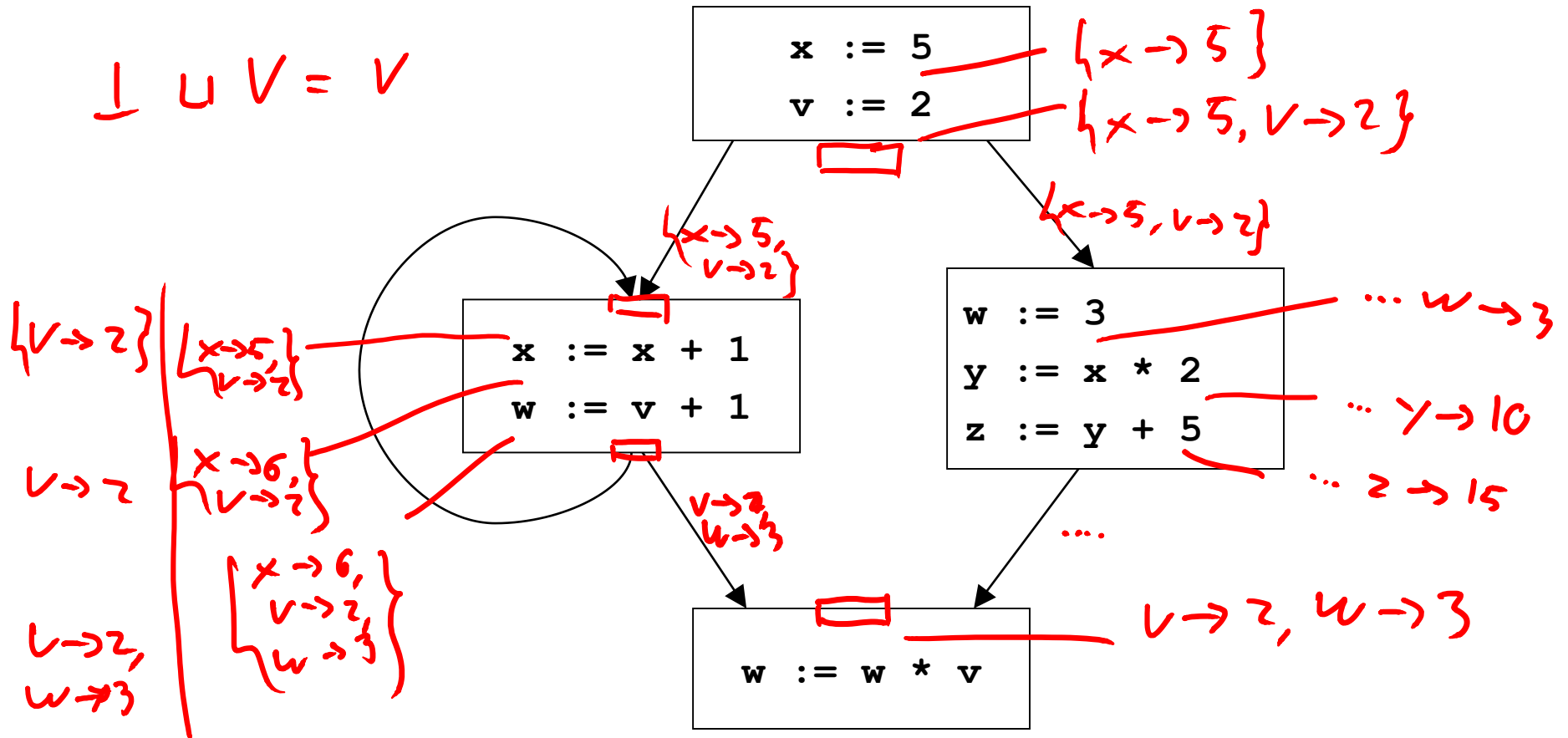 where $A = \{\, x \to N \mid x \in \text{Vars} \wedge N \in Z \,\}$

Example

$T = \emptyset$

$\perp \sqcup V = V$

```
x := 5
v := 2
```

$\{x \rightarrow 5\}$

$\{x \rightarrow 5, V \rightarrow 2\}$

$\{x \rightarrow 5, V \rightarrow 2\}$

```
x := x + 1
w := v + 1
```

$\{x \rightarrow 5, V \rightarrow 2\}$

```
w := 3
y := x * 2
z := y + 5
```

$\cdots w \rightarrow 3$

$\cdots Y \rightarrow 10$

$\cdots z \rightarrow 15$

$\{V \rightarrow 2\}$

$\{x \rightarrow 5, V \rightarrow 2\}$

$\{x \rightarrow 6, V \rightarrow 2\}$

$V \rightarrow 2$

$\{x \rightarrow 6, V \rightarrow 2, w \rightarrow 3\}$

$V \rightarrow 2, w \rightarrow 3$

$V \rightarrow 2, w \rightarrow 3$

```
w := w * v
```

$V \rightarrow 2, w \rightarrow 3$

Another Example

∅

```
    x := 5
a  := x + 10
```

$x \rightarrow 5$

$\{x \rightarrow 5, a \rightarrow 15\}$

```
x := x + 1
x := x - 1
```

1

$\{x \rightarrow 5, a \rightarrow 15\}$

$\{x \rightarrow 6, a \rightarrow 15\}$

$\{x \rightarrow 5, a \rightarrow 15\}$

```
b := x + 10
```

$x \rightarrow 5, a \rightarrow 15$

Another Example starting at top

x := 5
a := x + 10

x → 5

{x → 5, a → 15}

z := 10

x := x + 1
x := x - 1
z := 10

∅

0

0

0

b := x + 10

Back to lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
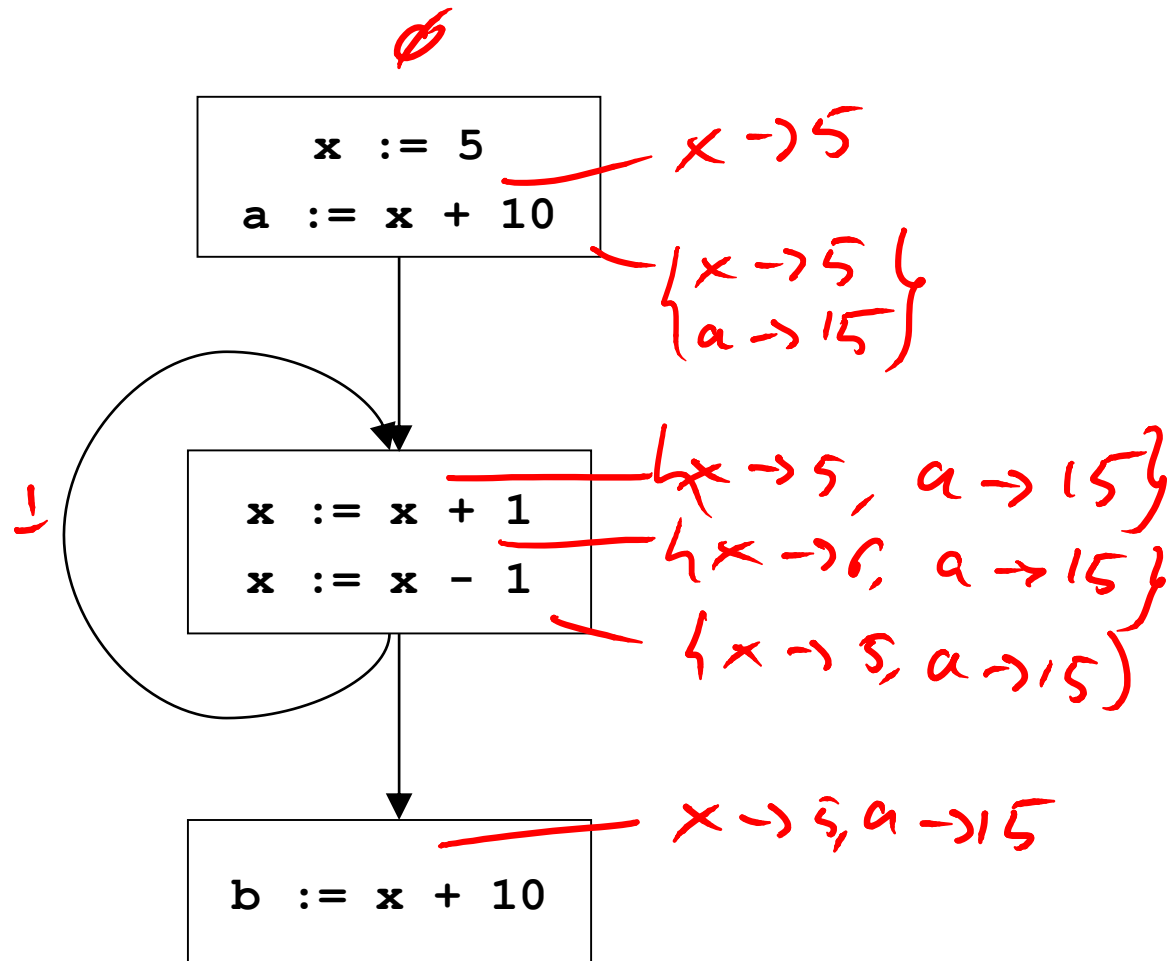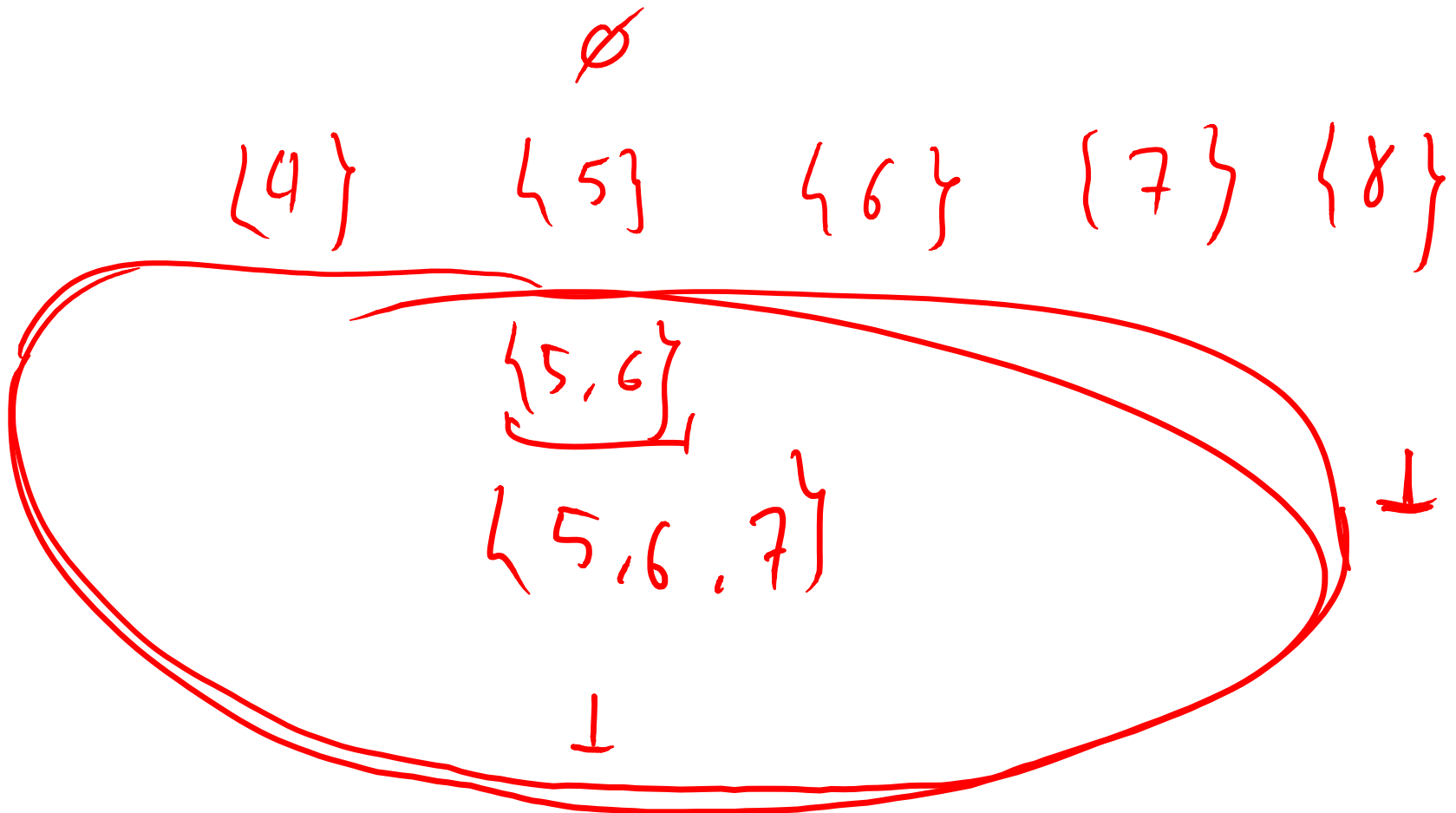 where $A = \{ x \to N \mid x \in \text{Vars} \wedge N \in Z \}$
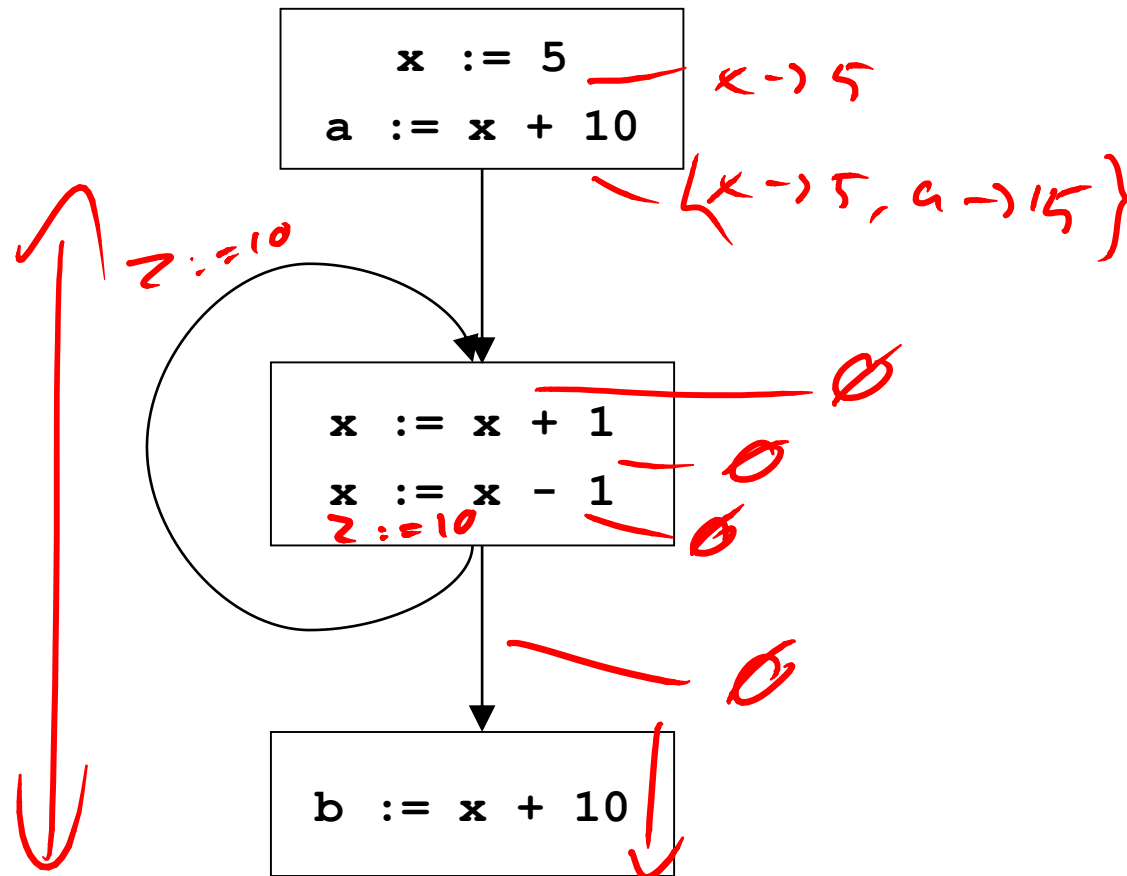
- What's the problem with this lattice?

Back to lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
  where $A = \{ x \rightarrow N \mid x \in Vars \wedge N \in Z \}$

- What's the problem with this lattice?

- Lattice is infinitely high, which means we can't guarantee termination

Better lattice
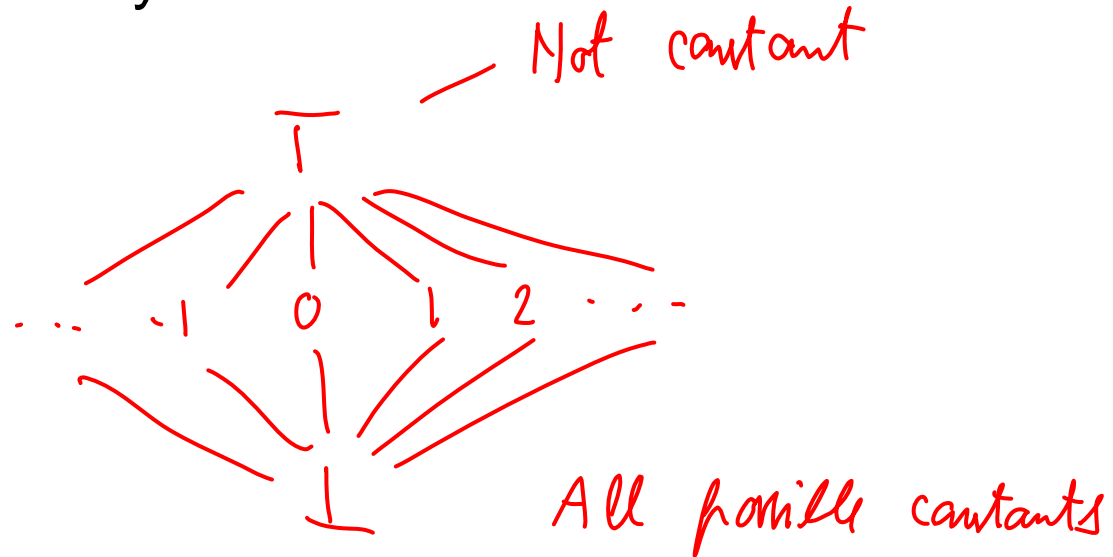
- Suppose we only had one variable

Better lattice

- Suppose we only had one variable

Not constant

All possible constants

- $D = \{\bot, \top\} \cup Z$

- $\forall i \in Z . \bot \sqsubseteq i \wedge i \sqsubseteq \top$

- height = 3

For all variables

- Two possibilities

- Option 1: Tuple of lattices

- Given lattices $(D_1, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ ... $(D_n, \sqsubseteq_n, \bot_n, \top_n, \sqcup_n, \sqcap_n)$ create:

    tuple lattice $D^n =$

For all variables

(⊤, ⊥, ⊥)

- Two possibilities

- Option 1: Tuple of lattices

(α, ⊥, ⊥)

(⊥, ⊥, ⊥)

- Given lattices $(D_1, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1) \ldots (D_n, \sqsubseteq_n, \bot_n, \top_n, \sqcup_n, \sqcap_n)$ create:

    tuple lattice $D^n = ((D_1 \times \ldots \times D_n), \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ where
    $\bot = (\bot_1, \ldots, \bot_n)$
    $\top = (\top_1, \ldots, \top_n)$
    $(a_1, \ldots, a_n) \sqcup (b_1, \ldots, b_n) = (a_1 \sqcup_1 b_1, \ldots, a_n \sqcup_n b_n)$
    $(a_1, \ldots, a_n) \sqcap (b_1, \ldots, b_n) = (a_1 \sqcap_1 b_1, \ldots, a_n \sqcap_n b_n)$
    $height = height(D_1) + \ldots + height(D_n)$

For all variables

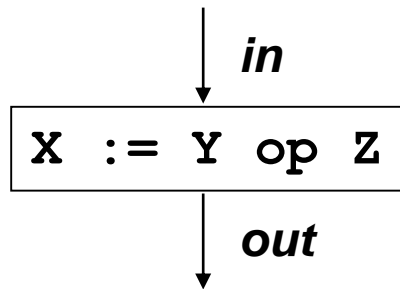- Option 2: Map from variables to single lattice

- Given lattice $(D, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and a set V, create:

map lattice $V \to D = (V \to D, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$

$$\bot = \lambda v . \bot,$$

$$a \sqcup b = \lambda v . a(v) \sqcup_1 b(v)$$

$$[a \mapsto \bot, b \mapsto \bot]$$

$$[a \mapsto 5, b \mapsto \bot]$$

$$[a \mapsto$$

Back to example

in

X := Y op Z

out

$$F_{X := Y \text{ op } Z}(\text{in}) = \text{in} \left[X \mapsto \text{in}(Y) \; \hat{op} \; \text{in}(z) \right]$$

$$0 \; \hat{*} \; \bot$$

$\hat{op}$	$\bot$	b	$\top$
$\bot$	$\bot$	$\top$	$\top$
a	$\top$	$a \text{ op } b$	$\top$
$\top$	$\top$	$\top$	$\top$

Back to example

$$\boxed{\texttt{X := Y op Z}}$$

in → (into box), *out* ↓ (out of box)
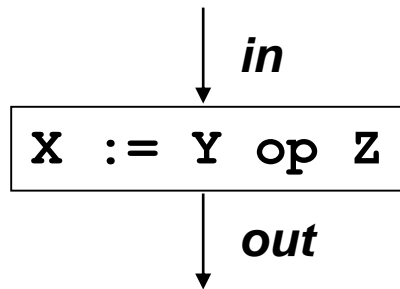
$F_{X := Y \text{ op } Z}(\text{in}) = \text{in} [\, X \rightarrow \text{in}(Y) \, \widehat{\text{op}} \, \text{in}(Z) \,]$

where a $\widehat{\text{op}}$ b =

$\widehat{\text{op}}$	$\perp$	d_1	$\top$
$\perp$	$\perp$	$\perp$	$\top$
d_2	$\perp$	$d_1 \text{ op } d_2$	$\top$
$\top$	$\top$	$\top$	$\top$

General approach to domain design

- ## Simple lattices:
  - boolean logic lattice
  - powerset lattice
  - incomparable set: set of incomparable values, plus top and bottom (eg const prop lattice)
  - two point lattice: just top and bottom

- ## Use combinators to create more complicated lattices
  - tuple lattice constructor
  - map lattice constructor

May vs Must

- Has to do with definition of computed info

$$x \rightarrow \{ a \rightarrow b, \ a \rightarrow c \}$$

- Set of x → y must-point-to pairs $\{ a \rightarrow d \}$
  - if we compute x → y, then, then during program execution, x must point to y

- Set of x→ y may-point-to pairs
  - if during program execution, it is possible for x to point to y, then we must compute x → y

May vs must

$\{y \rightarrow 10\}$

$* \ x := 5$

$\{x \rightarrow a, x \rightarrow e\}$

$\text{print}(y)$

$\sqsubseteq 1$

	May	Must
most optimistic (bottom)	$\emptyset$	FS
most conservative (top)	$x \rightarrow y, \ y \rightarrow x \ldots$	$\emptyset$
safe	add	remove
merge	$\cup$	$\cap$

May vs must

	May	Must
most optimistic (bottom)	empty set	full set
most conservative (top)	full set	empty set
safe	overly big	overly small
merge	$\cup$	$\cap$

Common Sub-expression Elim

- Want to compute when an expression is available in a var

- Domain:

Common Sub-expression Elim

- Want to compute when an expression is available in a var

- Domain:

$$S = \{ X \to E \mid X \in Var, E \in Expr \}$$

$$D = 2^S$$

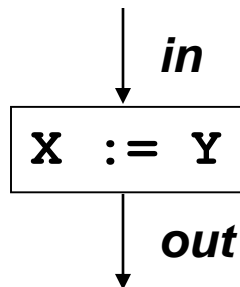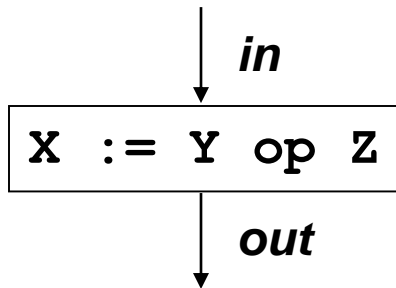$$\bot = S$$

$$\top = \emptyset$$

$$\sqcup = \cap$$

Flow functions

$a := b + c$ $\{a = b + c\}$

$b := e + f$

$\longrightarrow \{a \to b+c, b \to e+f\}$

$b+c \to a$

$$F_{X := Y \text{ op } Z}(\text{in}) = \text{in} - \{X \to *\}$$
$$\{X \to \cdots X \cdots \}$$

$$\cup \{X \to Y \text{ op } Z\}$$
$$\wedge Y \neq X \wedge Z \neq X$$

$$F_{X := Y}(\text{in}) = \text{in} - \{X \to *\}$$

$$\cup \{X \to \bar{E} \mid Y \to \bar{E} \in \text{in}\}$$

Flow functions
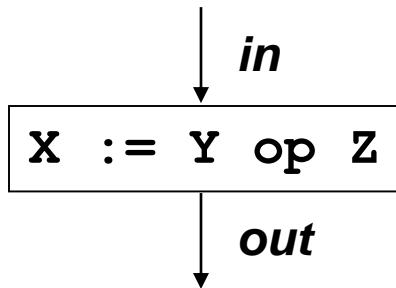
$$F_{X := Y \text{ op } Z}(\text{in}) = \text{in} - \{ X \to * \}$$
$$- \{ * \to \dots X \dots \} \cup$$
$$\{ X \to Y \text{ op } Z \mid X \neq Y \wedge X \neq Z \}$$
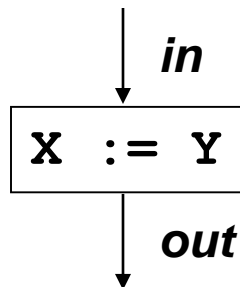
$$F_{X := Y}(\text{in}) = \text{in} - \{ X \to * \}$$
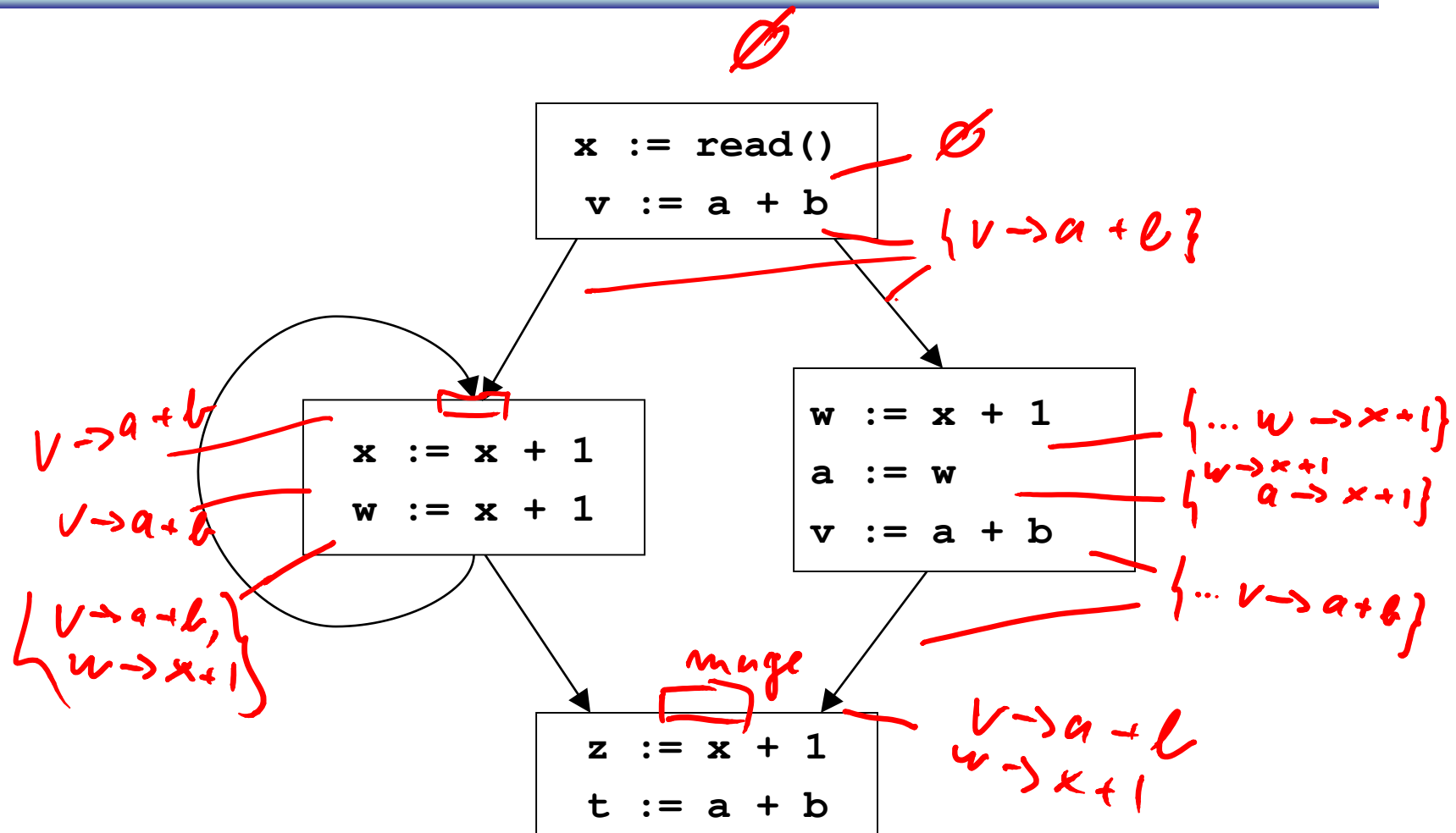$$- \{ * \to \dots X \dots \} \cup$$
$$\{ X \to E \mid Y \to E \in \text{in} \}$$

Example

$$T = 0$$
$$\underline{1} = FS$$

$\emptyset$

```
x := read()
v := a + b
```
$\emptyset$

$\{V \rightarrow a + b\}$

```
x := x + 1
w := x + 1
```

$V \rightarrow a + b$

$V \rightarrow a + b$

$\left\{ \begin{array}{l} V \rightarrow a + b, \\ w \rightarrow x + 1 \end{array} \right\}$

```
w := x + 1
a := w
v := a + b
```

$\{ \ldots w \rightarrow x + 1 \}$

$\left\{ \begin{array}{l} w \rightarrow x + 1 \\ a \rightarrow x + 1 \end{array} \right\}$

$\{ \ldots V \rightarrow a + b \}$

```
z := x + 1
t := a + b
```

merge

$V \rightarrow a + b$
$w \rightarrow x + 1$

Direction of analysis

- Although constraints are not directional, flow functions are

- All flow functions we have seen so far are in the forward direction

- In some cases, the constraints are of the form
  in = F(out)

- These are called backward problems.

- Example: live variables
  - compute the set of variables that may be live

Live Variables

- A variable is live at a program point if it will be used before being redefined

- A variable is dead at a program point if it is redefined before being used

$int\ x;$

$x := 5$

$x := 1$

$x := 1$

eax

$print(x)$

$x := 5$

$print(x)$ × Live

$x := 2$

eax

$print(x)$

Example: live variables

- Set D =

- Lattice: (D, $\sqsubseteq$, $\bot$, $\top$, $\sqcup$, $\sqcap$) =

$\varnothing$ Var $\cup$ $\wedge$

$Pow(Var),$

Example: live variables
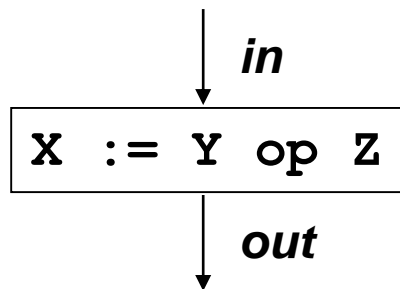
- Set $D = 2^{\text{Vars}}$

- Lattice: $(D, \sqsubseteq, \perp, \top, \sqcup, \sqcap) = (2^{\text{Vars}}, \subseteq, \emptyset, \text{Vars}, \cup, \cap)$

```
        │ in
   ┌────▼────────┐
   │ X := Y op Z │
   └────┬────────┘
        │ out
        ▼
```

$a := b + c$ ⇐

$$F_{X := Y \text{ op } Z}(\text{out}) = \text{out} - \{x\} \cup \{y, z\}$$

Example: live variables

- Set $D = 2^{\text{Vars}}$

- Lattice: $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) = (2^{\text{Vars}}, \subseteq, \emptyset, \text{Vars}, \cup, \cap)$
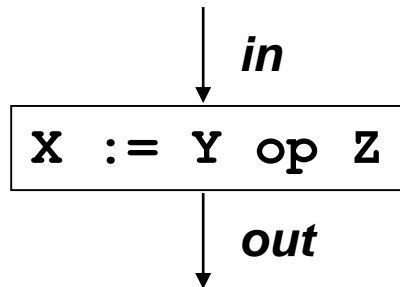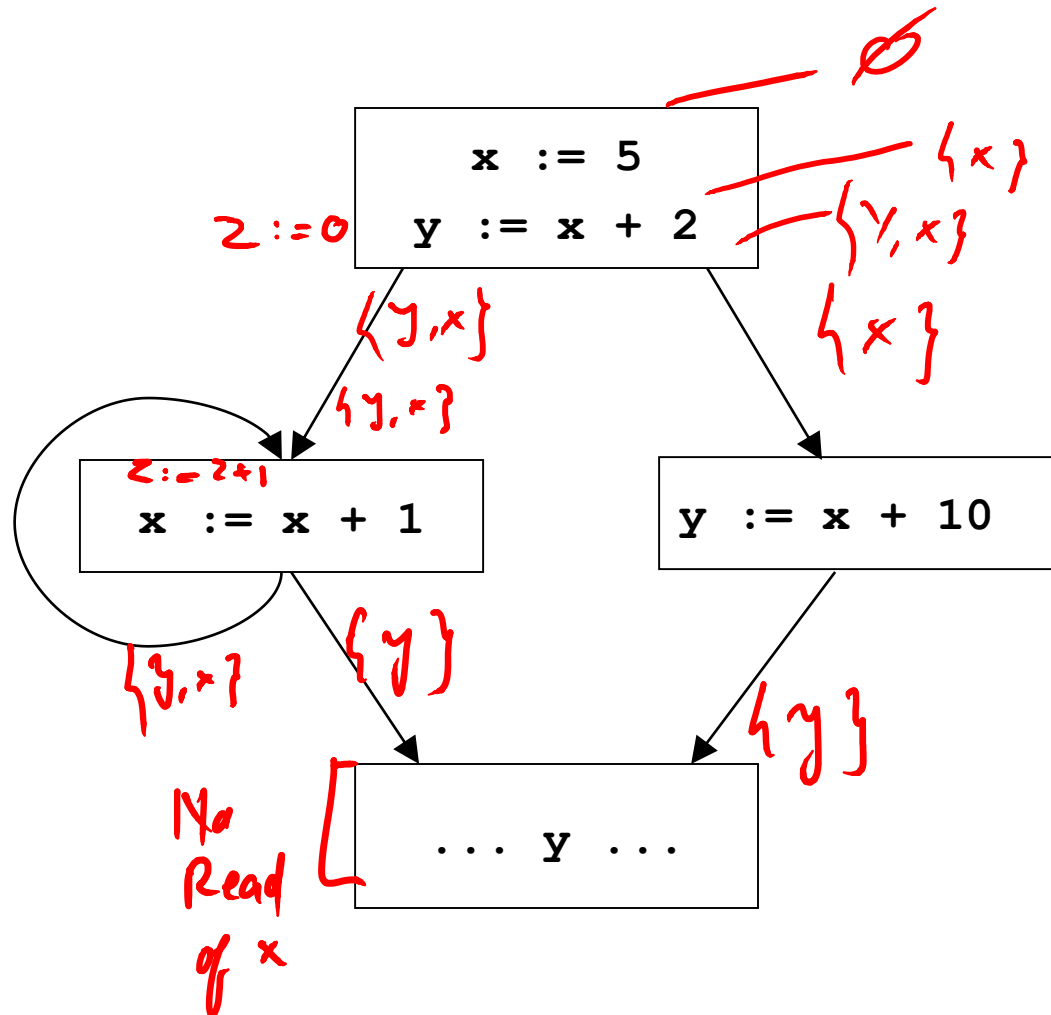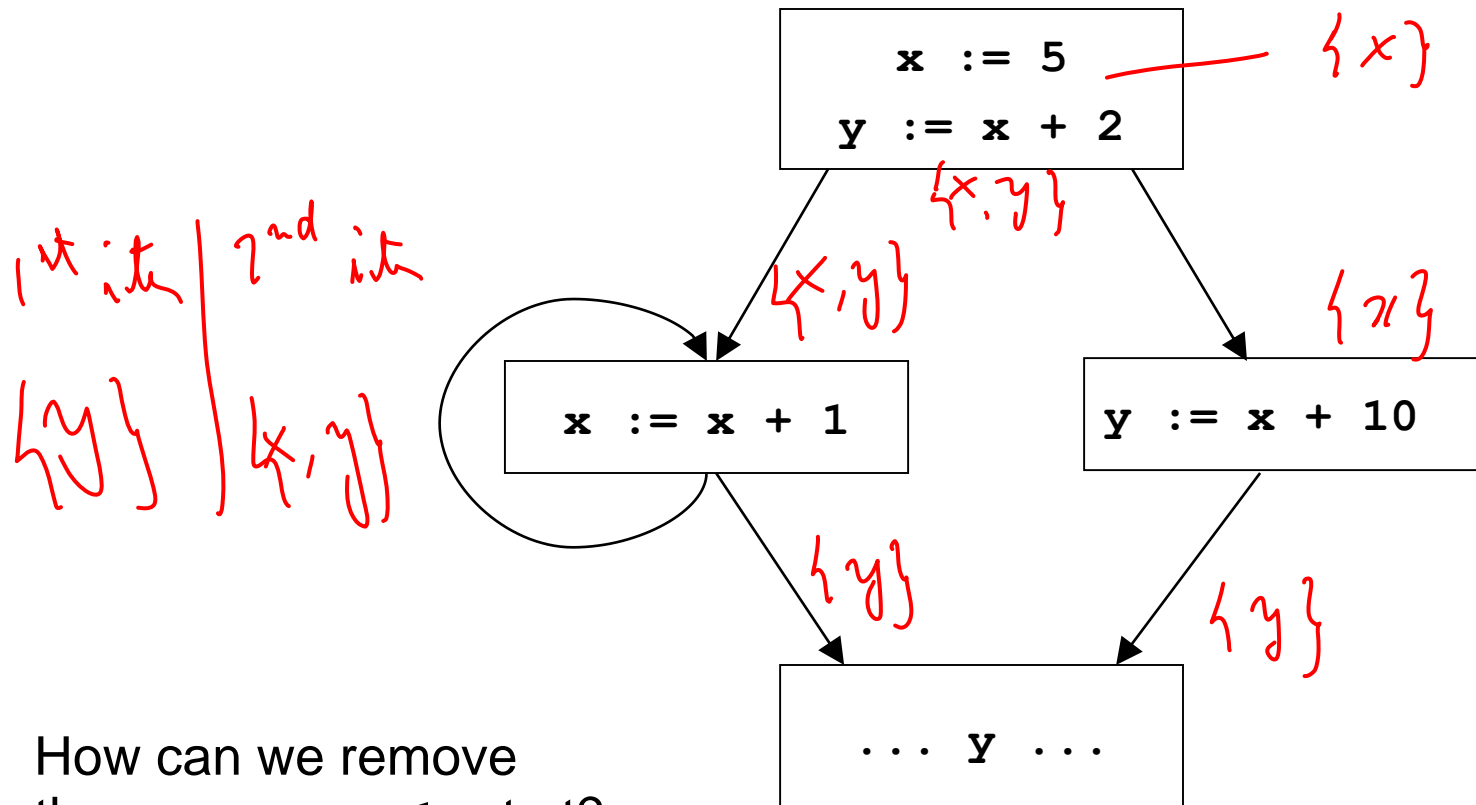
in

```
X := Y op Z
```

out

$$F_{X := Y \text{ op } Z}(\text{out}) = \text{out} - \{\, X \,\} \cup \{\, Y, Z \,\}$$

Example: live variables

x := 5
y := x + 2

z := 0

∅
{x}
{y, x}

{y, x}
{y, x}

{x}

z := z+1
x := x + 1

y := x + 10

{y, x}

{y}

{y}

No
Read
of x

... y ...

Example: live variables

x := 5
y := x + 2   {x}

{x,y}

1st iter | 2nd iter

{y} | {x,y}

{x,y}

{x}

x := x + 1   y := x + 10

{y}   {y}

... y ...
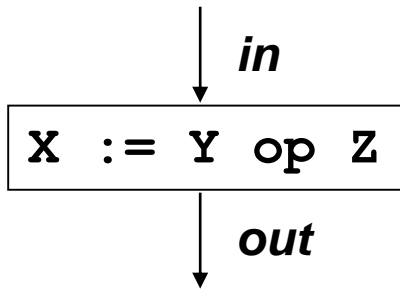
How can we remove
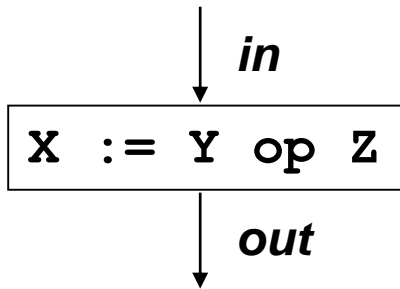the x := x + 1 stmt?

Revisiting assignment

$$F_{X := Y \text{ op } Z}(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z\}$$

Revisiting assignment

$\Rightarrow b := \cdots \Leftarrow$

$\Rightarrow$ ~~a := b + c~~ $- \{d, e\}$

$a := d + e \leftarrow \{d, e\}$

in

```
X := Y op Z
```

out

$F_{X\ :=\ Y\ op\ Z}(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z\}$

$\text{out} - \{x\} \cup$

$x \notin \text{out}? \ \emptyset : \{Y, Z\}$

Theory of backward analyses

- Can formalize backward analyses in two ways

- Option 1: reverse flow graph, and then run forward problem

- Option 2: re-develop the theory, but in the backward direction

Precision

- Going back to constant prop, in what cases would we lose precision?

Precision

- Going back to constant prop, in what cases would we lose precision?

```
x := 5
if (<expr>) {
   x := 6
}
... x ...

where <expr> is
equiv to false
```

```
if (p) {
   x := 5;
} else
   x := 4;
}
...
if (p) {
   y := x + 1
} else {
   y := x + 2
}
... y ...
```

```
if (...) {
   x := -1;
} else
   x := 1;
}
y := x * x;
... y ...
```

Precision

- The first problem: Unreachable code
  - solution: run unreachable code removal before
  - the unreachable code removal analysis will do its best, but may not remove all unreachable code

- The other two problems are path-sensitivity issues
  - Branch correlations: some paths are infeasible
  - Path merging: can lead to loss of precision

MOP: meet over all paths

- Information computed at a given point is the meet of the information computed by each path to the program point

```
if (...) {
    x := -1;
} else
    x := 1;
}
y := x * x;
... y ...
```
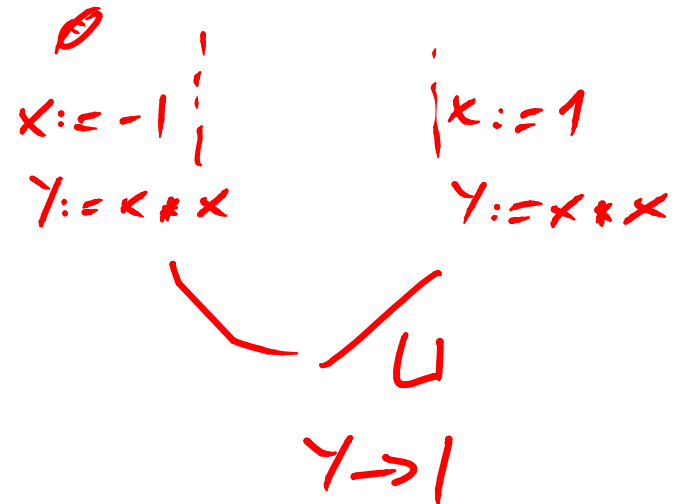
MOP

- For a path p, which is a sequence of statements $[s_1, ..., s_n]$, define: $F_p(in) = F_{s_n}(...F_{s_1}(in) ...)$

- In other words: $F_p = F_{s_1} \circ \cdots \circ F_{s_n}$

- Given an edge e, let paths-to(e) be the (possibly infinite) set of paths that lead to e

- Given an edge e, MOP(e) = $\bigsqcup_{p \,\in\, paths\text{-}to(e)} F_p(\bot)$
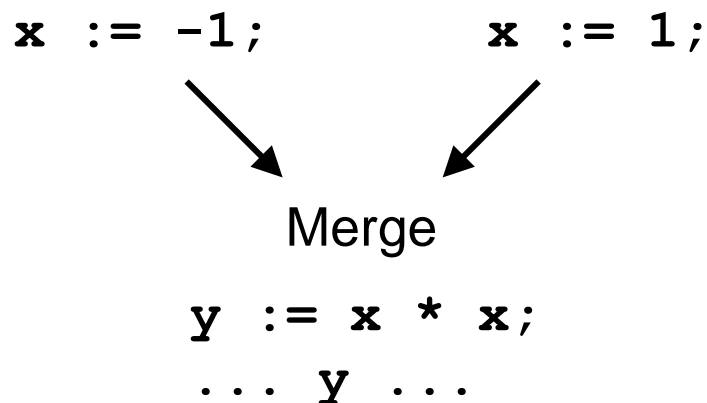
- For us, should be called JOP (ie: join, not meet)

MOP vs. dataflow

- MOP is the "best" possible answer, given a fixed set of flow functions
  - This means that MOP $\sqsubseteq$ dataflow at edge in the CFG

- In general, MOP is not computable (because there can be infinitely many paths)
  - vs dataflow which is generally computable (if flow fns are monotonic and height of lattice is finite)

- And we saw in our example, in general, MOP $\neq$ dataflow

MOP vs. dataflow

- However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?

Dataflow

```
x := -1;        x := 1;
```

          Merge

```
y := x * x;
... y ...
```

MOP

```
x := -1;      x := 1;
y := x * x;   y := x * x;
... y ...      ... y ...
```
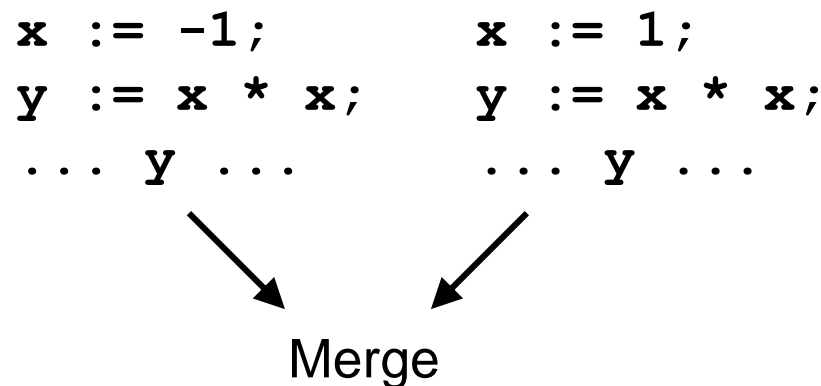
          Merge

MOP vs. dataflow

- However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?

- Distributive problems. A problem is distributive if:

  $\forall$ a, b . F(a $\sqcup$ b) = F(a) $\sqcup$ F(b)

- If flow function is distributive, then MOP = dataflow

Summary of precision

- Dataflow is the basic algorithm

- To basic dataflow, we can add path-separation
  - Get MOP, which is same as dataflow for distributive problems
  - Variety of research efforts to get closer to MOP for non-distributive problems

- To basic dataflow, we can add path-pruning
  - Get branch correlation

- To basic dataflow, can add both:
  - meet over all feasible paths