Lecture 15 – Dataflow Analysis

THEORY OF COMPILATION

Eran Yahav

www.cs.technion.ac.il/~yahave/tocs2011/compilers-lec15.pptx

Static Analysis

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

"The algorithmic discovery of properties of a program by inspection of its source text[1]"
-- Manna, Pnueli

1 Does not have to literally be the source text, just means w/o running it

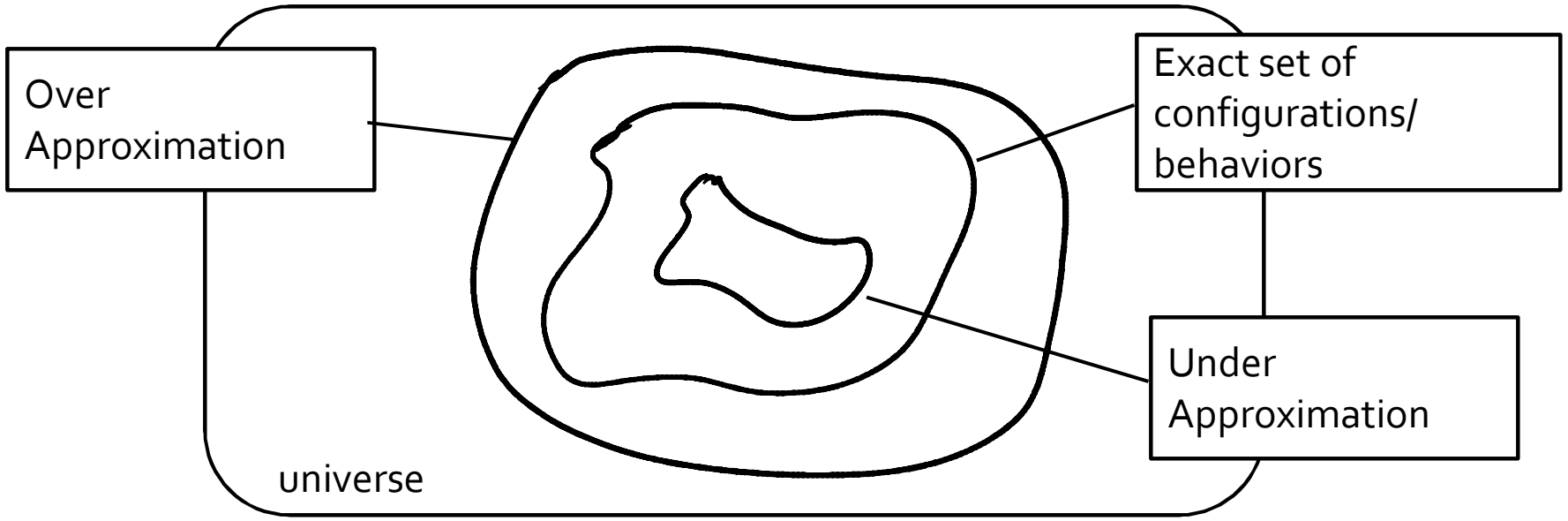
Static Analysis

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

- 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);
```

- Over approximation: assertion may be violated

Precision

```
main(…) {
 printf(“assertion may be vioalted\n”);
}
```

- Lose precision only when required
- Understand where precision is lost

Static Analysis

- Formalize software behavior in a mathematical model (semantics)
- Prove properties of the mathematical model
  - Automatically, typically with approximation of the formal semantics
- Develop theory and tools for program correctness and robustness

Static Analysis

- Spans a wide range from type checking to full verification

- General safety specifications
- Security properties (e.g., information flow)
- Concurrency correctness conditions (e.g., progress, linearizability)
- Correct use of libraries (e.g., typestate)

- Under-approximations useful for bug-finding, test-case generation,…

Static Analysis: Techniques

- Dataflow analysis
- Constraint-based analysis
- Type and effect systems
- Abstract Interpretation
  (advanced course next semester)
- …

The WHILE Language: Syntax

$A \in AExp$ arithmetic expressions
$B \in BExp$ boolean expressions
$S \in Stmt$ statements

Var set of variables
Lab set of labels
Op_a arithmetic operators
Op_b boolean operators
Op_r relational operators

$a ::= x \mid n \mid a_1\ op_a\ a_2$

$b ::= true \mid false \mid not\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$

$S ::= [x := a]^{lab}$
$\mid [skip]^{lab}$
$\mid S_1;S_2$
$\mid if\ [b]^{lab}\ then\ S_1\ else\ S_2$
$\mid while\ [b]^{lab}\ do\ S$

(We are going to abuse syntax later for readability)

The WHILE Language: Structural Operational Semantics

$\sigma \in \text{State} = \text{Var} \rightarrow \text{Z}$

Configuration: <S, σ> or just σ for terminal configuration

A: AExp $\rightarrow$ (State $\rightarrow$ Z)

$A[\![x]\!]\sigma = \sigma(x)$

$A[\![n]\!]\sigma = N[\![n]\!]$

$A[\![a_1 \text{ op } a_2]\!]\sigma = A[\![a_1]\!]\sigma \text{ op } A[\![a_2]\!]\sigma$

B: BExp $\rightarrow$ (State $\rightarrow$ { true, false})

$B[\![\text{not } b]\!]\sigma = \neg B[\![b]\!]\sigma$

$B[\![b_1 \text{ op}_b \ b_2]\!]\sigma = B[\![b_1]\!]\sigma \text{ op}_b B[\![b_2]\!]\sigma$

$B[\![a_1 \text{ op}_r \ a_2]\!]\sigma = A[\![a_1]\!]\sigma \text{ op}_r A[\![a_2]\!]\sigma$

The WHILE Language: Structural Operational Semantics

[ass] $\langle [x := a]^{lab}, \sigma \rangle \rightarrow \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$

[skip] $\langle [skip]^{lab}, \sigma \rangle \rightarrow \sigma$

[seq_1] $$\frac{\langle S_1, \sigma \rangle \rightarrow \langle S'_1, \sigma' \rangle}{\langle S_1; S_2, \sigma \rangle \rightarrow \langle S'_1; S_2, \sigma' \rangle}$$

[seq_2] $$\frac{\langle S_1, \sigma \rangle \rightarrow \sigma'}{\langle S_1; S_2, \sigma \rangle \rightarrow \langle S_2, \sigma' \rangle}$$

[if_1]	$\langle$if [b]^{lab} then S1 else S2, $\sigma \rangle \rightarrow \langle$S1, $\sigma\rangle$	if $\mathcal{B}[\![b]\!]\sigma$ = true
[if_2]	$\langle$if [b]^{lab} then S1 else S2, $\sigma \rangle \rightarrow \langle$S2, $\sigma\rangle$	if $\mathcal{B}[\![b]\!]\sigma$ = false
[wh_1]	$\langle$while [b]^{lab} do S, $\sigma \rangle \rightarrow \langle$(S; while [b]^{lab} do S), $\sigma\rangle$	if $\mathcal{B}[\![b]\!]\sigma$ = true
[wh_1]	$\langle$while [b]^{lab} do S, $\sigma \rangle \rightarrow \sigma$	if $\mathcal{B}[\![b]\!]\sigma$ = false

(Table 2.6 from PPA)

Example of Derivation Sequence

$[y := x]^1;$
$[z := 1]^2;$
$\text{while } [y > 0]^3 \text{ (}$
$[z := z * y]^4;$
$[y := y - 1]^5;$
)
$[y := 0]^6$

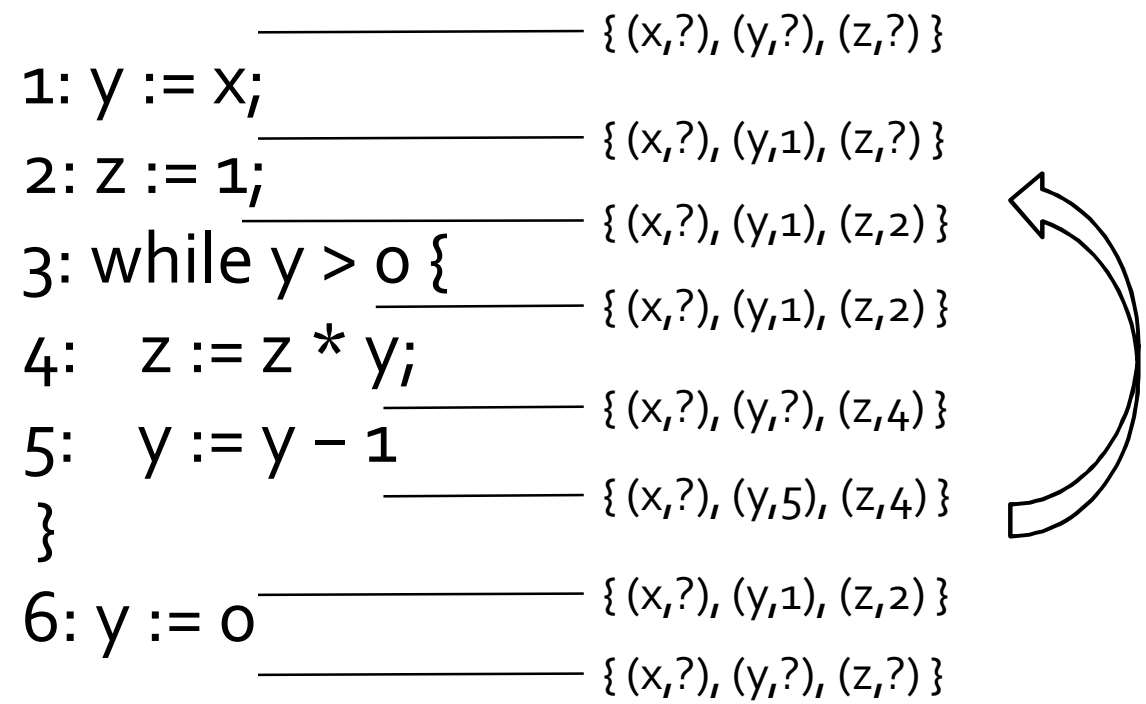
$< [y := x]^1;[z := 1]^2;\text{while } [y > 0]^3 ([z := z * y]^4;[y := y - 1]^5;)[y := 0]^6, \{ x \mapsto 42, y \mapsto 0, z \mapsto 0 \} > \longrightarrow$

$< [z := 1]^2;\text{while } [y > 0]^3 ([z := z * y]^4;[y := y - 1]^5;)[y := 0]^6, \{ x \mapsto 42, y \mapsto 42, z \mapsto 0 \} > \longrightarrow$

$< \text{while } [y > 0]^3 ([z := z * y]^4;[y := y - 1]^5;)[y := 0]^6, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \longrightarrow$

$< ([z := z * y]^4;[y := y - 1]^5;);\text{while } [y > 0]^3 ([z := z * y]^4;[y := y - 1]^5;)[y := 0]^6, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \ldots$

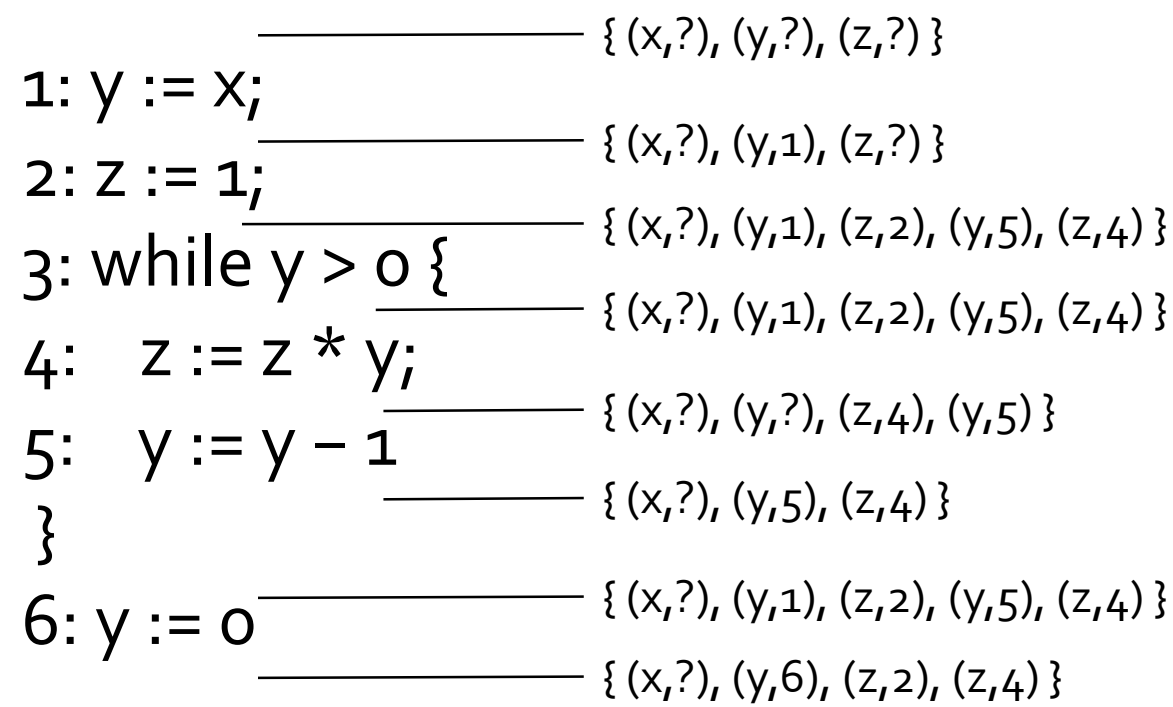
Dataflow Analysis



- Reaching Definitions
  - The assignment **lab: var := exp** reaches **lab'** if there is an execution where **var** was last assigned at **lab**

(adapted from Nielson, Nielson & Hankin)

Dataflow Analysis



- Reaching Definitions
  - The assignment **lab: var := exp** reaches **lab'** if there is an execution where **var** was last assigned at **lab**

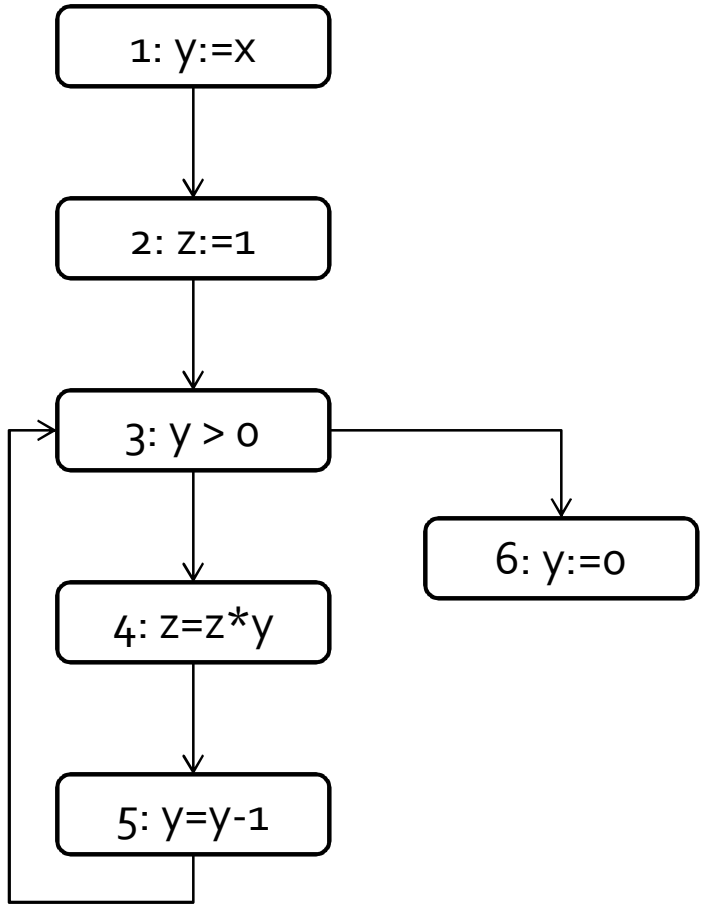
(adapted from Nielson, Nielson & Hankin)

Dataflow Analysis

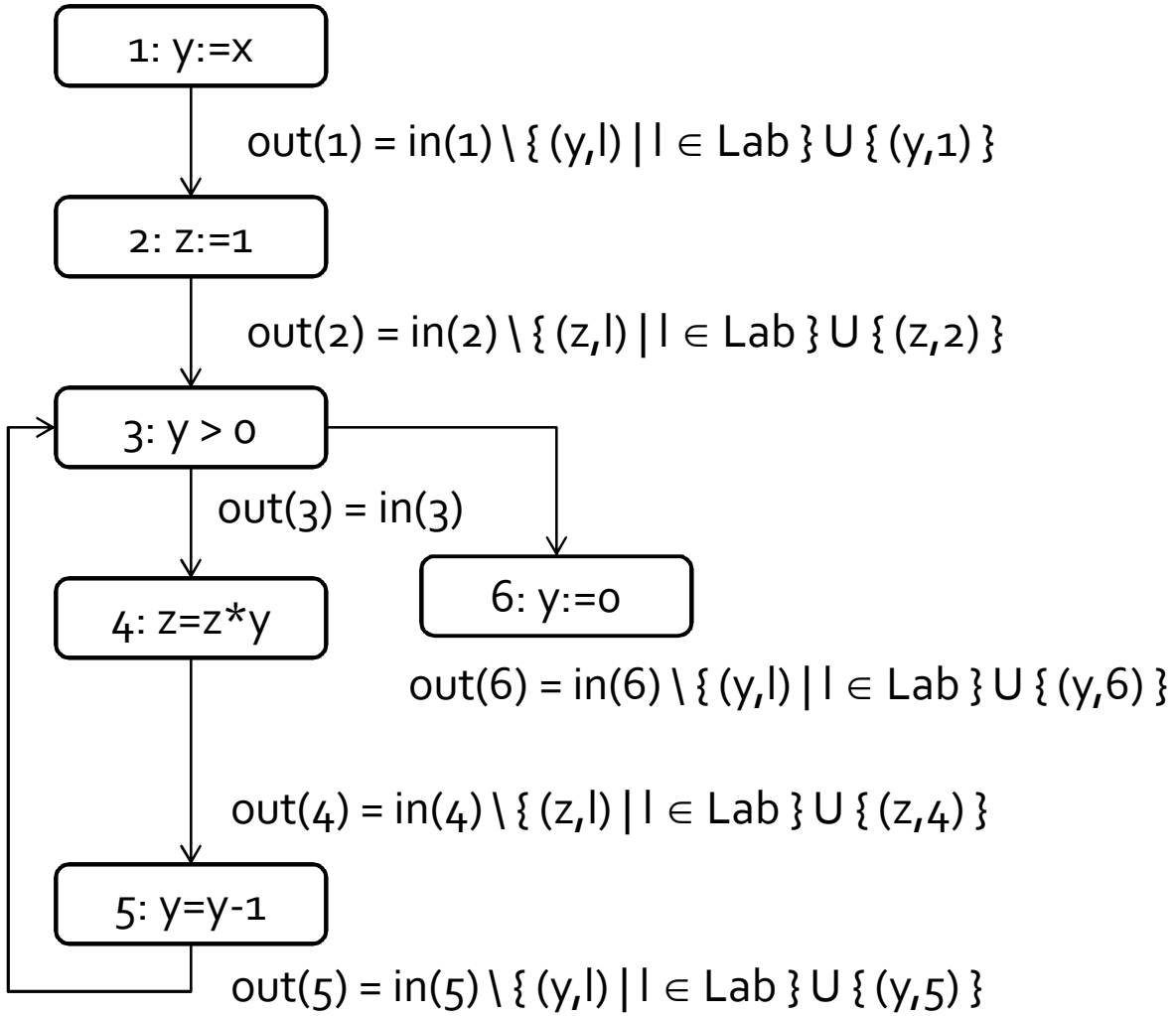
- Build control-flow graph
- Assign transfer functions
- Compute fixed point

Control-Flow Graph

```
1: y := x;
2: z := 1;
3: while y > 0 {
4:    z := z * y;
5:    y := y – 1
 }
6: y := 0
```



Transfer Functions



in(1) = { (x,?), (y,?), (z,?) }
in(2) = out(1)
in(3) = out(2) U out(5)
in(4) = out(3)
in(5) = out(4)
in(6) = out(3)

System of Equations

$in(1) = \{ (x,?), (y,?), (z,?) \}$
$in(2) = out(1)$
$in(3) = out(2) \cup out(5)$
$in(4) = out(3)$
$in(5) = out(4)$
$In(6) = out(3)$
$out(1) = in(1) \setminus \{ (y,l) \mid l \in Lab \} \cup \{ (y,1) \}$
$out(2) = in(2) \setminus \{ (z,l) \mid l \in Lab \} \cup \{ (z,2) \}$
$out(3) = in(3)$
$out(4) = in(4) \setminus \{ (z,l) \mid l \in Lab \} \cup \{ (z,4) \}$
$out(5) = in(5) \setminus \{ (y,l) \mid l \in Lab \} \cup \{ (y,5) \}$
$out(6) = in(6) \setminus \{ (y,l) \mid l \in Lab \} \cup \{ (y,6) \}$

$F: (\wp(Var \times Lab))^{12} \rightarrow (\wp(Var \times Lab))^{12}$

System of Equations

In(1)	∅				
In(2)	∅				
In(3)	∅				
In(4)	∅				
In(5)	∅				
In(6)	∅				
Out(1)	∅				
Out(2)	∅				
Out(3)	∅				
Out(4)	∅				
Out(5)	∅				
Out(6)	∅				

F: $(\wp(\text{Var} \times \text{Lab}))^{12} \rightarrow (\wp(\text{Var} \times \text{Lab}))^{12}$

in(1)={(x,?),(y,?),(z,?)} , 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,l) | l ∈ Lab } U { (y,1) }, out(2) = in(2) \ { (z,l) | l ∈ Lab } U { (z,2) }
out(3) = in(3), out(4) = in(4) \ { (z,l) | l ∈ Lab } U { (z,4) }
out(5) = in(5) \ { (y,l) | l ∈ Lab } U { (y,5) }, out(6) = in(6) \ { (y,l) | l ∈ Lab } U { (y,6) }

System of Equations

	$RD_{\varnothing}$	$F(RD_{\varnothing})$	$F(F(RD_{\varnothing}))$	$F(F(F(RD_{\varnothing})))$	$F(F(F(F(RD_{\varnothing}))))$
In(1)	∅	{(x,?),(y,?),(z,?)}	==	==	==
In(2)	∅	∅	{(y,1)}	{(x,?),(y,1),(z,?)}	==
In(3)	∅	∅	{(z,2),(y,5)}	{(z,2),(y,5)}	{(z,2),(z,4),(y,5)}
In(4)	∅	∅	∅	∅	{(z,2),(y,5)}
In(5)	∅	∅	{(z,4)}	{(z,4)}	{(z,4)}
In(6)	∅	∅	∅	∅	{(z,2),(y,5)}
Out(1)	∅	{(y,1)}	{(x,?),(y,1),(z,?)}	==	==
Out(2)	∅	{(z,2)}	{(z,2)}	{(z,2),(y,1)}	==
Out(3)	∅	∅	∅	{(z,2),(y,5)}	{(z,2),(y,5)}
Out(4)	∅	{(z,4)}	{(z,4)}	{(z,4)}	{(z,4)}
Out(5)	∅	{(y,5)}	{(y,5)}	{(z,4),(y,5)}	{(z,4),(y,5)}
Out(6)	∅	{(y,6)}	{(y,6)}	{(y,6)}	{(y,6)}

...

F: $(\wp(\text{Var} \times \text{Lab}))^{12} \rightarrow (\wp(\text{Var} \times \text{Lab}))^{12}$

$RD \sqsubseteq RD'$ when $\forall i$: $RD(i) \subseteq RD'(i)$

Monotonicity

F: $(\wp(\text{Var} \times \text{Lab}))^{12} \rightarrow (\wp(\text{Var} \times \text{Lab}))^{12}$

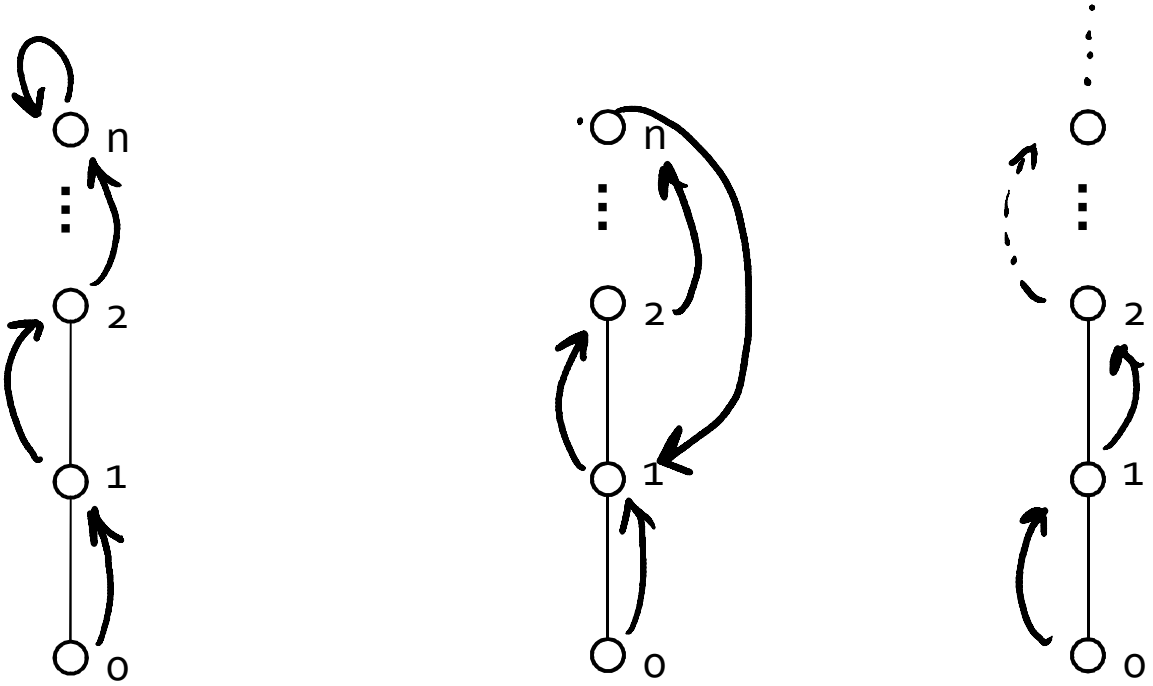
	$RD_\varnothing$	$F(RD_\varnothing)$	$F(F(RD_\varnothing))$	$F(F(F(RD_\varnothing)))$	$F(F(F(F(RD_\varnothing))))$
…					
Out(1)	∅	**{(y,1)}**	**{(x,?),(y,1),(z,?)}**	**{(y,1)}**	**….**
…					

…

out(1) = {(y,1)} when (x,?) ∈ out(1)
in(1) \ { (y,l) | l ∈ Lab } U { (y,1) }, otherwise

(silly example just for illustration)

Convergence



Least Fixed Point

- We will see later why it exists
- For now, mostly informally…

F: ($\wp$(Var x Lab))^{12} $\rightarrow$ ($\wp$(Var x Lab))^{12}

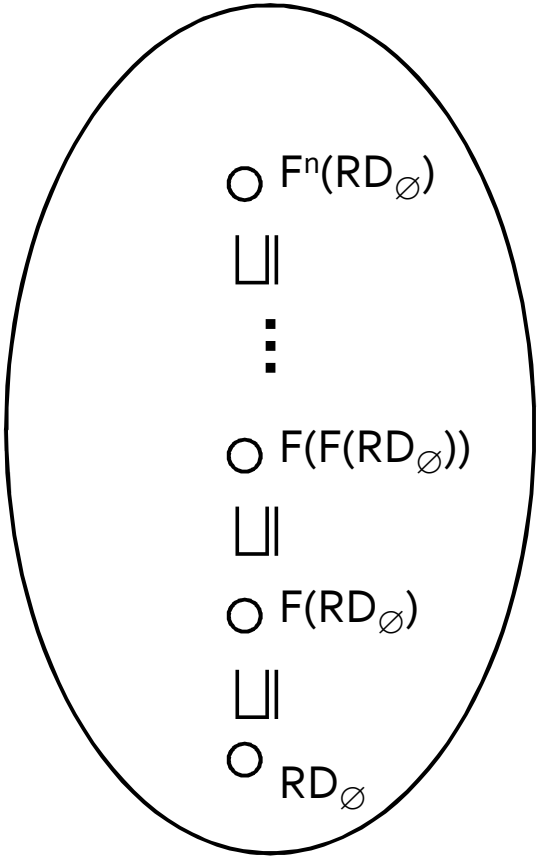
RD $\sqsubseteq$ RD′ when $\forall$i: RD(i) $\subseteq$ RD′(i)

F is monotone:
RD $\sqsubseteq$ RD′ implies that F(RD) $\sqsubseteq$ F(RD′)

$RD_\varnothing$ = ($\varnothing$, $\varnothing$,…,$\varnothing$)

$F(RD_\varnothing)$, $F(F(RD_\varnothing)$, $F(F(F(RD_\varnothing))$, … $F^n(RD_\varnothing)$

$F^{n+1}(RD_\varnothing) = F^n(RD_\varnothing)$



Things that Should Trouble You

- How did we get the transfer functions?
- How do we know these transfer functions are safe (conservative)?
- How do we know that these transfer functions are optimal?

Some required notation

blocks : Stmt $\longrightarrow$ P(Blocks)
blocks($[x := a]^{lab}$) = $\{[x := a]^{lab}\}$
blocks($[skip]^{lab}$) = $\{[skip]^{lab}\}$
blocks(S_1; S_2) = blocks(S_1) $\cup$ blocks(S_2)
blocks(if $[b]^{lab}$ then S_1 else S_2) = $\{[b]^{lab}\} \cup$ blocks(S_1) $\cup$ blocks(S_2)
blocks(while $[b]^{lab}$ do S) = $\{[b]^{lab}\} \cup$ blocks(S)

FV: (BExp $\cup$ AExp)$\longrightarrow$ Var
Variables used in an expression

AExp(a) = all non-unit expressions in the arithmetic expression a
similarly AExp(b) for a boolean expression b

Available Expressions Analysis

```
[x := a+b]1;
[y := a*b]2;
while [y > a+b]3 (
  [a := a + 1]4;
  [x := a + b]5
 )
```

(a+b) always available at label 3

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

Available Expressions Analysis

- Property space
  - in_{AE}, out_{AE}: Lab $\rightarrow \wp$(AExp)
  - Mapping a label to set of arithmetic expressions available at that label

- Dataflow equations
  - Flow equations – how to join incoming dataflow facts
  - Effect equations - given an input set of expressions S, what is the effect of a statement

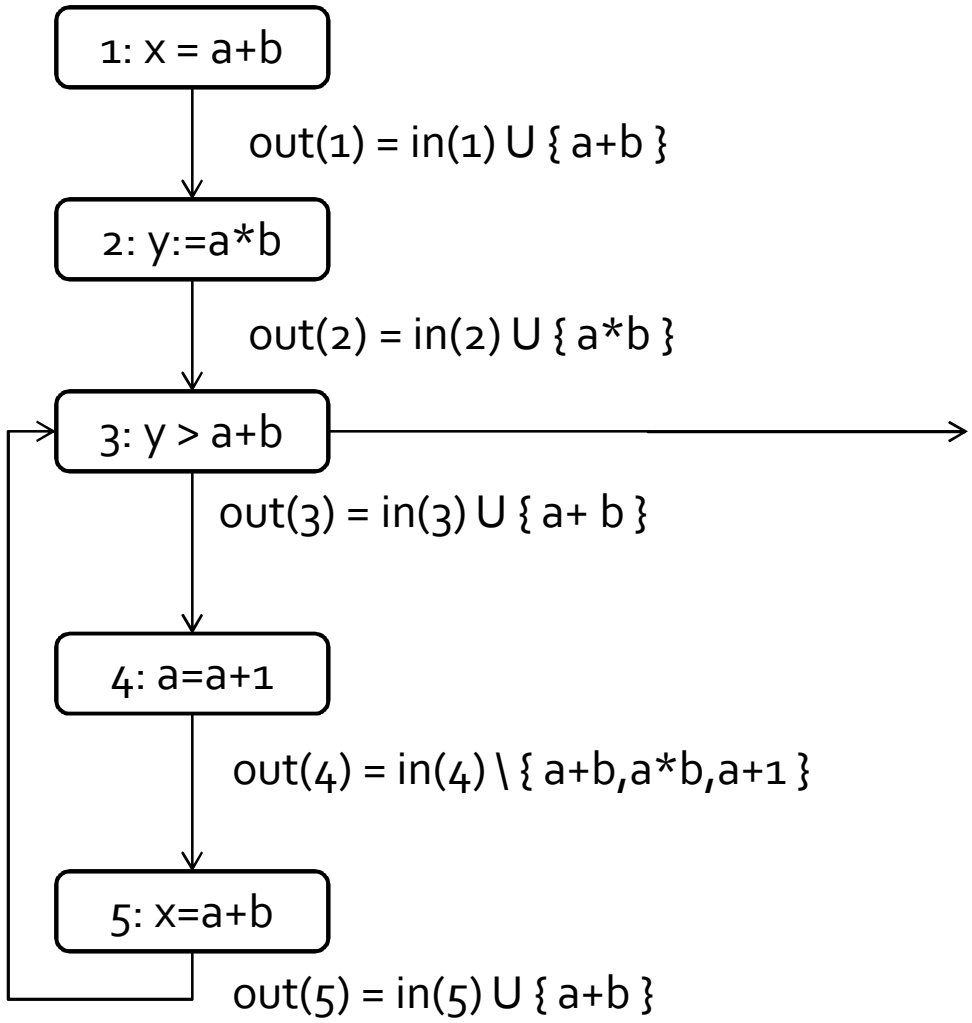
Available Expressions Analysis

- $in_{AE}(lab)$ =
  - $\varnothing$ when lab is the initial label
  - $\bigcap \{ out_{AE}(lab') \mid lab' \in pred(lab) \}$ otherwise
- $out_{AE}(lab)$ = …

Block	**out (lab)**
$[x := a]^{lab}$	$in(lab) \setminus \{ a' \in AExp \mid x \in FV(a') \} \cup \{ a' \in AExp(a) \mid x \notin FV(a') \}$
$[skip]^{lab}$	$in(lab)$
$[b]^{lab}$	$in(lab) \cup AExp(b)$

From now on going to drop the AE subscript when clear from context

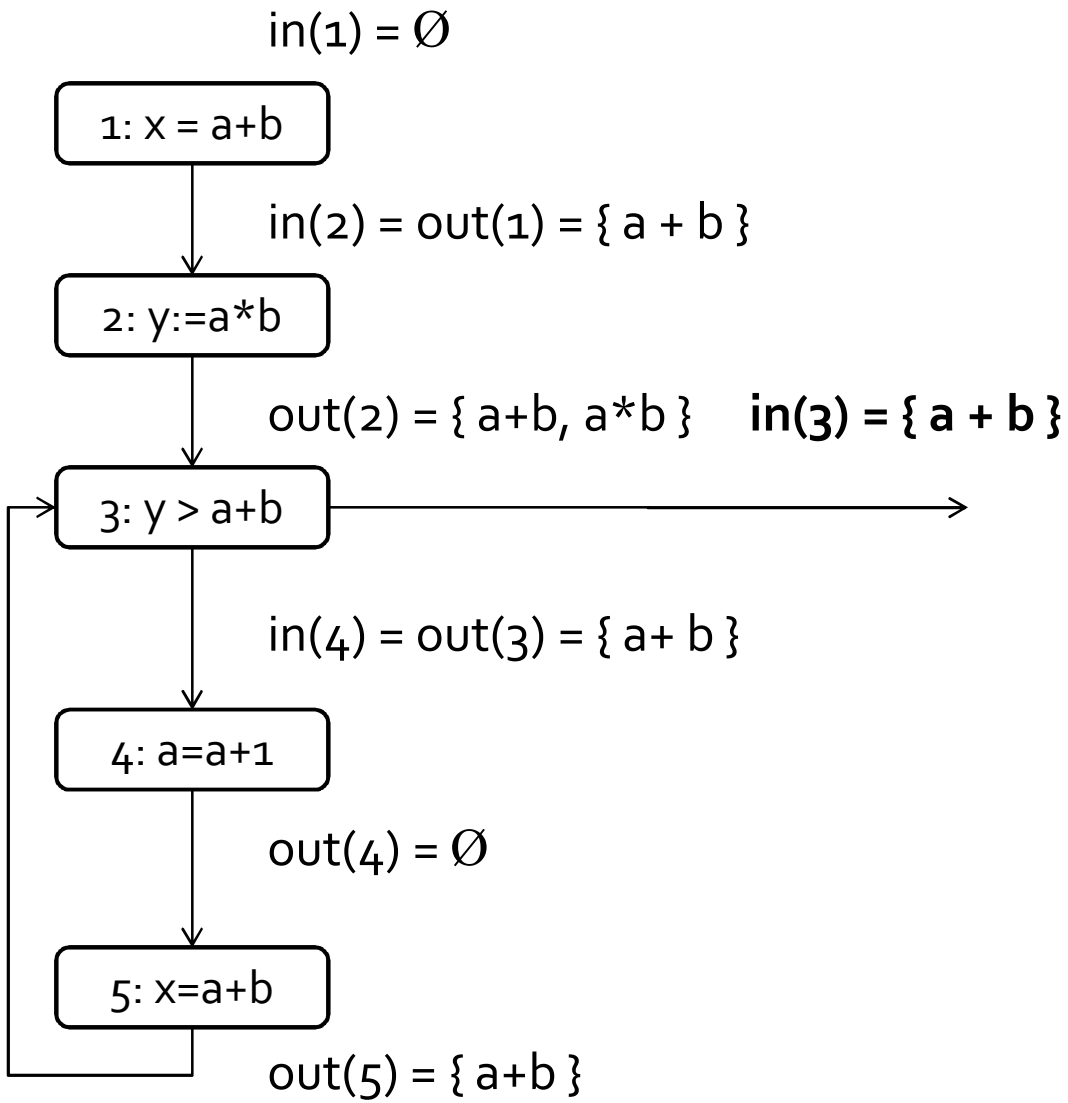
Transfer Functions



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

$[x := a+b]^1$;
$[y := a*b]^2$;
while $[y > a+b]^3$ (
 $[a := a + 1]^4$;
 $[x := a + b]^5$
)

Solution


in(1) = Ø
1: x = a+b
in(2) = out(1) = { a + b }
2: y:=a*b
out(2) = { a+b, a*b }
in(3) = { a + b }
3: y > a+b
in(4) = out(3) = { a+ b }
4: a=a+1
out(4) = Ø
5: x=a+b
out(5) = { a+b }

Kill/Gen

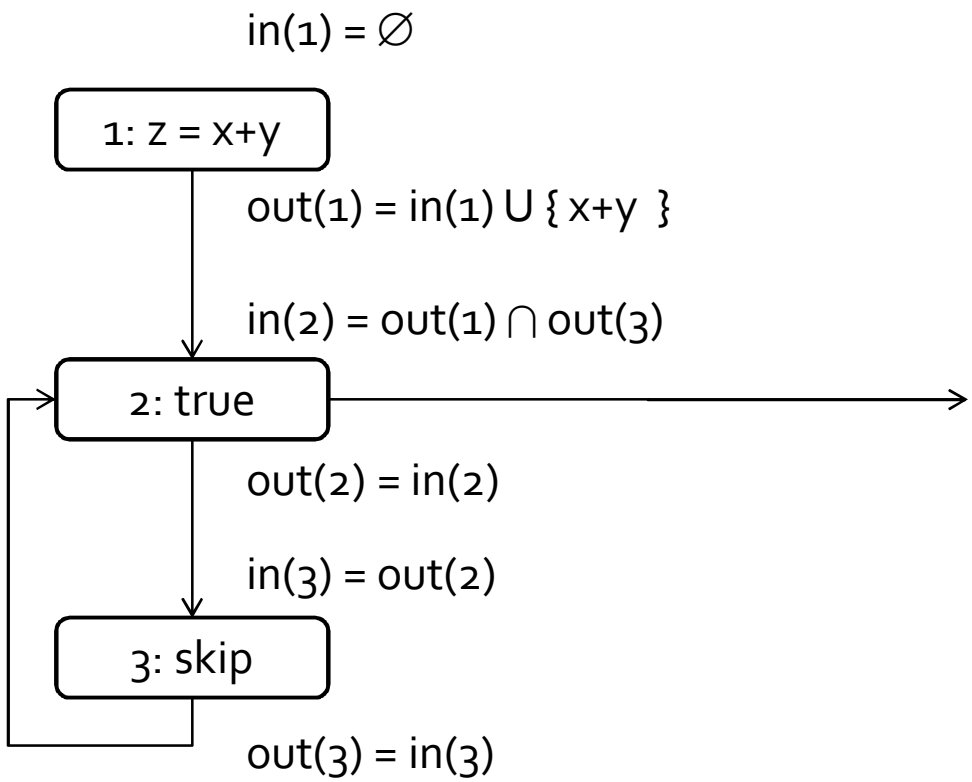
Block	**out (lab)**
$[x := a]^{lab}$	$in(lab) \setminus \{ a' \in AExp \mid x \in FV(a') \} \cup \{ a' \in AExp(a) \mid x \notin FV(a') \}$
$[skip]^{lab}$	$in(lab)$
$[b]^{lab}$	$in(lab) \cup AExp(b)$

Block	**kill**	**gen**
$[x := a]^{lab}$	$\{ a' \in AExp \mid x \in FV(a') \}$	$\{ a' \in AExp(a) \mid x \notin FV(a') \}$
$[skip]^{lab}$	$\emptyset$	$\emptyset$
$[b]^{lab}$	$\emptyset$	$AExp(b)$

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

B^{lab} = block at label lab

Why solution with largest sets?



in(1) = Ø
in(2) = out(1) ∩ out(3)
in(3) = out(2)

[z := x+y]¹;
while [true]² (
  [skip]³;
)

After simplification: in(2) = in(2) ∩ { x+y }

Solutions: {x+y} or ∅

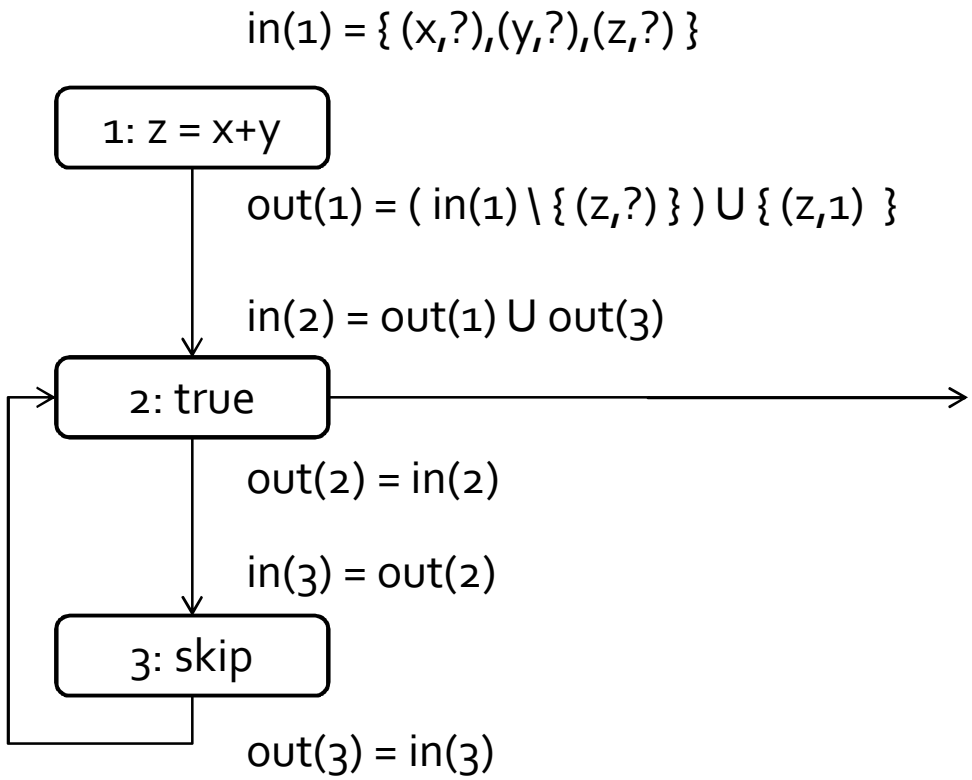
Reaching Definitions Revisited

Block	**out (lab)**
$[x := a]^{lab}$	$in(lab) \setminus \{ (x,l) \mid l \in Lab \} \cup \{ (x,lab) \}$
$[skip]^{lab}$	$in(lab)$
$[b]^{lab}$	$in(lab)$

Block	**kill**	**gen**
$[x := a]^{lab}$	$\{ (x,l) \mid l \in Lab \}$	$\{ (x,lab) \}$
$[skip]^{lab}$	$\emptyset$	$\emptyset$
$[b]^{lab}$	$\emptyset$	$\emptyset$

For each program point, which assignments may have been made and not overwritten, when program execution reaches this point along some path.

Why solution with smallest sets?



in(1) = { (x,?),(y,?),(z,?) }
in(2) = out(1) **U** out(3)
in(3) = out(2)

[z := x+y]^1;
while [true]^2 (
  [skip]^3;
)

After simplification: in(2) = in(2) U { (x,?),(y,?),(z,1) }

Many solutions: any superset of { (x,?),(y,?),(z,1) }

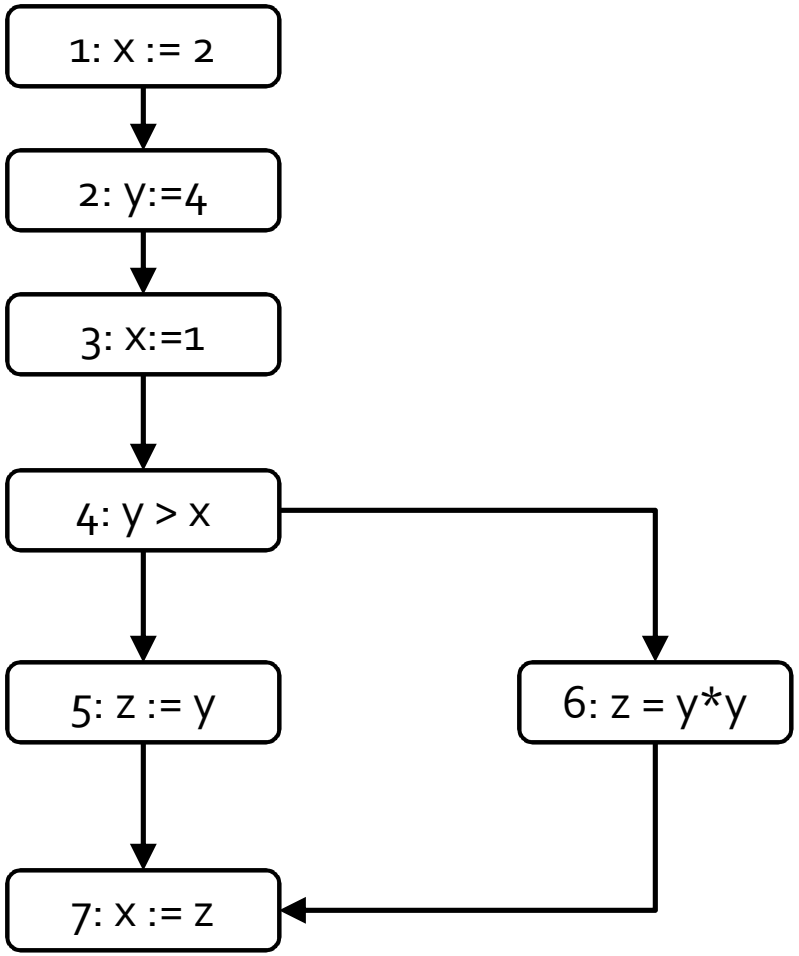
Live Variables

$[x := 2]^1$;
$[y := 4]^2$;
$[x := 1]^3$;
(if $[y > x]^4$ then $[z := y]^5$
else $[z := y * y]^6$);
$[x := z]^7$

For each program point, which variables may be live at the exit from the point.

Live Variables

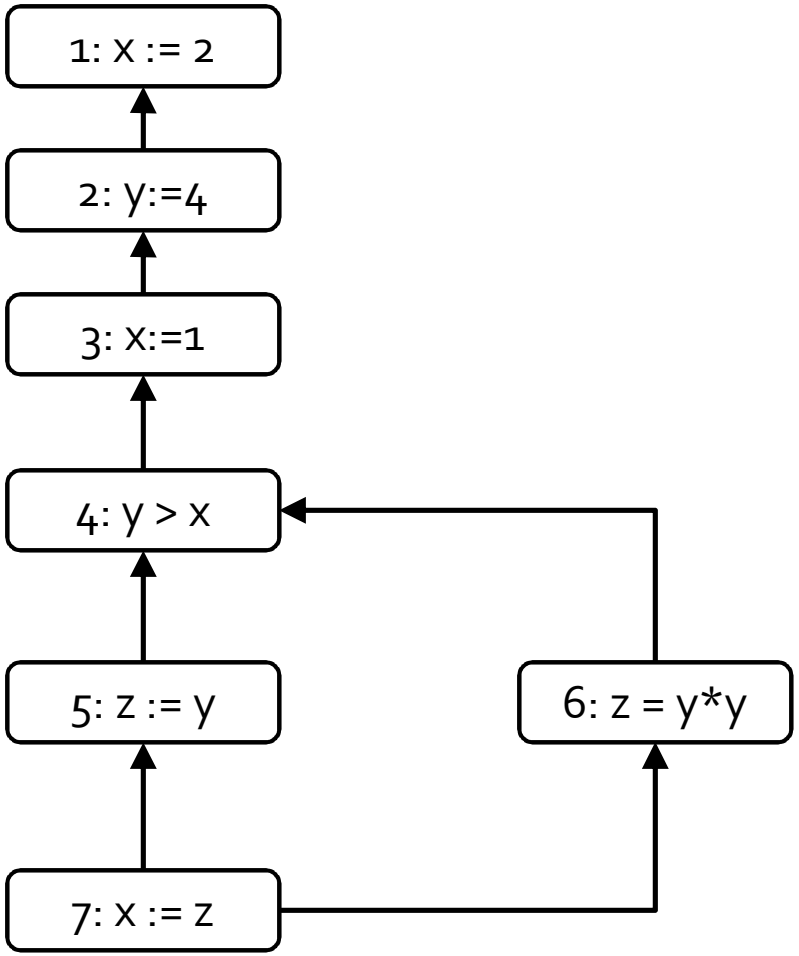
$[x:=2]^1$;
$[y:=4]^2$;
$[x:=1]^3$;
(if $[y>x]^4$ then $[z:=y]^5$
else $[z:=y*y]^6$);
$[x:=z]^7$



Live Variables

$[x := 2]^1$;
$[y := 4]^2$;
$[x := 1]^3$;
(if $[y > x]^4$ then $[z := y]^5$
else $[z := y * y]^6$);
$[x := z]^7$

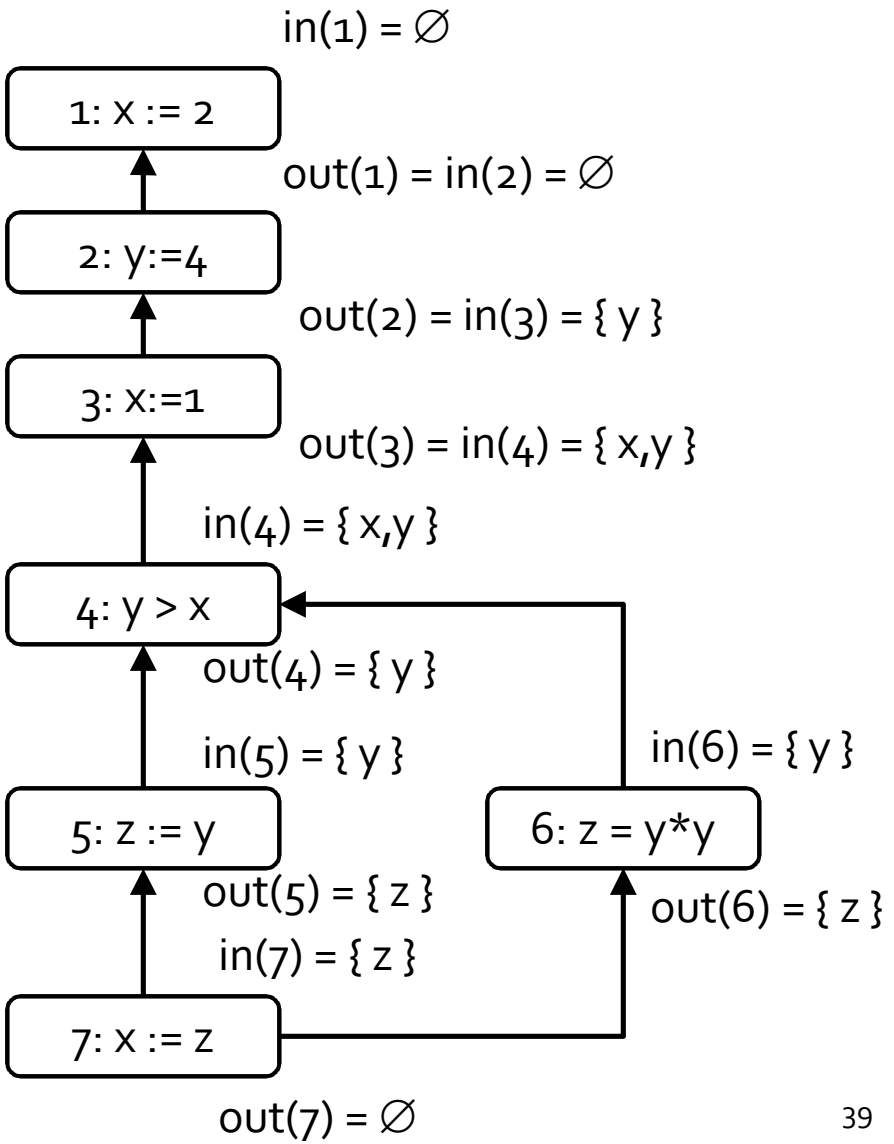
Block	kill	gen
$[x := a]^{lab}$	{ x }	{ FV(a) }
$[skip]^{lab}$	Ø	Ø
$[b]^{lab}$	Ø	FV(b)



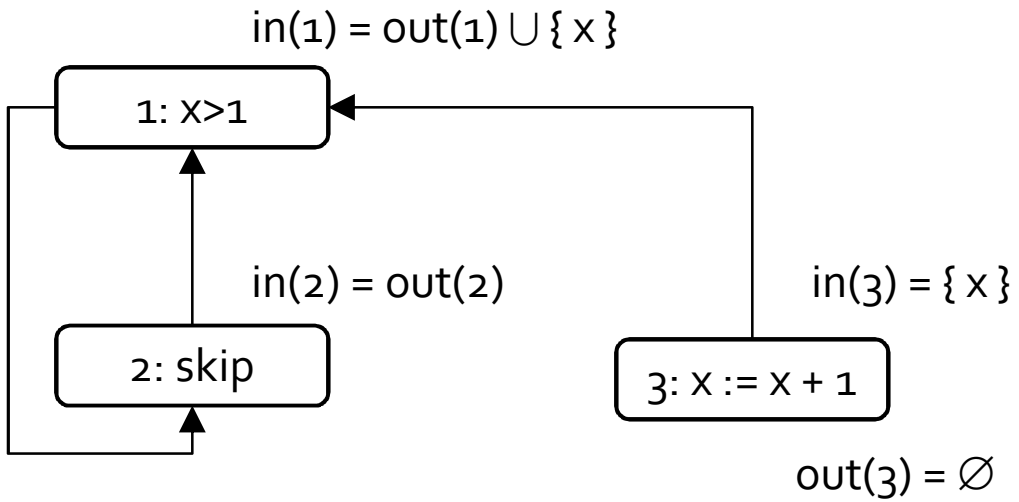
Live Variables: solution

$[x := 2]^1$;
$[y := 4]^2$;
$[x := 1]^3$;
(if $[y > x]^4$ then $[z := y]^5$
else $[z := y * y]^6$);
$[x := z]^7$

Block	kill	gen
$[x := a]^{lab}$	{ x }	{ FV(a) }
$[skip]^{lab}$	$\emptyset$	$\emptyset$
$[b]^{lab}$	$\emptyset$	FV(b)



Why solution with smallest set?



out(1) = in(2) **U** in(3)
out(2) = in(1)
out(3) = ∅

---

while [x>1]^1 (
  [skip]^2;
)
[x := x+1]^3;

After simplification: in(1) = in(1) U { x }

Many solutions: any superset of { x }

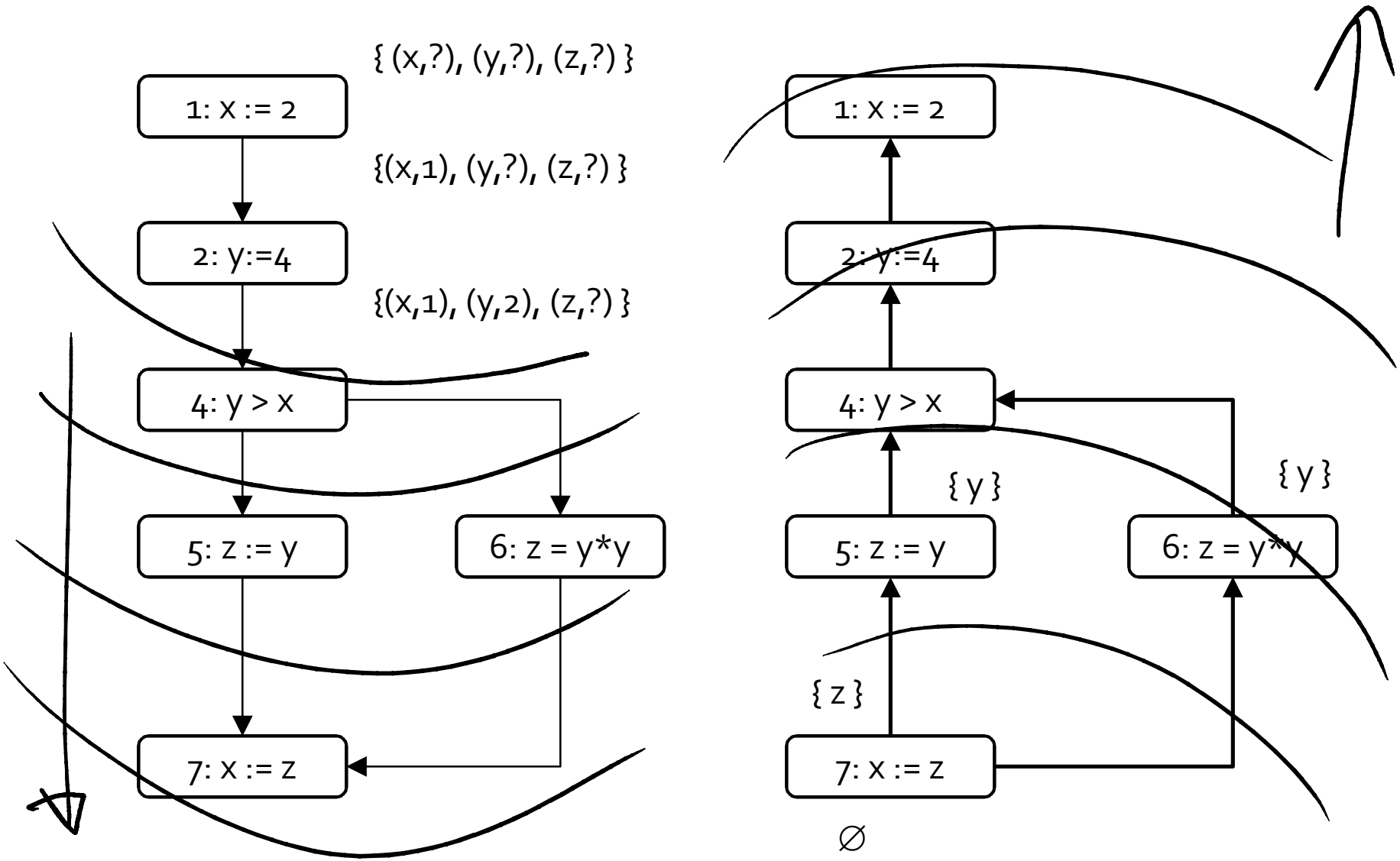
Monotone Frameworks

$$In(lab) = \begin{cases} Initial & \text{when } lab \in \text{Entry labels} \\ \bigsqcup \{ out(lab') \mid (lab', lab) \in \text{CFG edges} \} & \text{otherwise} \end{cases}$$

$$out(lab) = f_{lab}(in(lab))$$

- $\bigsqcup$ is $\bigcup$ or $\bigcap$
- CFG edges go either forward or backwards
- Entry labels are either initial program labels or final program labels (when going backwards)
- Initial is an initial state (or final when going backwards)
- f_{lab} is the transfer function associated with the blocks B^{lab}

Forward vs. Backward Analyses



Must vs. May Analyses

- When $\sqcup$ is $\cap$ - must analysis
  - Want largest sets the solves the equation system
  - Properties hold on all paths reaching a label (exiting a label, for backwards)

- When $\sqcup$ is $\cup$ - may analysis
  - Want smallest sets that solve the equation system
  - Properties hold at least on one path reaching a label (existing a label, for backwards)

Example: Reaching Definition

- L = $\wp$(Var×Lab) is partially ordered by $\subseteq$
- $\sqcup$ is $\cup$
- L satisfies the Ascending Chain Condition because Var × Lab is finite (for a given program)

Example: Available Expressions

- L = $\wp$(AExp) is partially ordered by $\supseteq$
- $\sqcup$ is $\cap$
- L satisfies the Ascending Chain Condition because AExp is finite (for a given program)

Analyses Summary

	Reaching Definitions	**Available Expressions**	**Live Variables**
L	$\wp$(Var x Lab)	$\wp$(AExp)	$\wp$(Var)
$\sqsubseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cup$	$\cap$	$\cup$
$\bot$	$\varnothing$	AExp	$\varnothing$
Initial	$\{ (x,?) \mid x \in Var \}$	$\varnothing$	$\varnothing$
Entry labels	{ init }	{ init }	final
Direction	Forward	Forward	Backward
F	$\{ f: L \rightarrow L \mid \exists k,g : f(val) = (val \setminus k) \cup g \}$		
f_{lab}	$f_{lab}(val) = (val \setminus kill) \cup gen$		

Analyses as Monotone Frameworks

- Property space
  - Powerset
  - Clearly a complete lattice
- Transformers
  - Kill/gen form
  - Monotone functions (let's show it)

Monotonicity of Kill/Gen transformers

- Have to show that $x \sqsubseteq x'$ implies $f(x) \sqsubseteq f(x')$
- Assume $x \sqsubseteq x'$, then for kill set k and gen set g $(x \setminus k) \cup g \sqsubseteq (x' \setminus k) \cup g$

- Technically, since we want to show it for all functions in F, we also have to show that the set is closed under function composition

Distributivity of Kill/Gen transformers

- Have to show that $f(x \sqcup y) \sqsubseteq f(x) \sqcup f(y)$
- $f(x \sqcup y) = ((x \sqcup y) \setminus k) \cup g$
  $= ((x \setminus k) \sqcup (y \setminus k)) \cup g$
  $= (((x \setminus k) \cup g) \sqcup ((y \setminus k) \cup g))$
  $= f(x) \sqcup f(y)$

- Used distributivity of $\sqcup$ and $\cup$
  - Works regardless of whether $\sqcup$ is $\cup$ or $\cap$