Lecture 04 – Numerical Abstractions

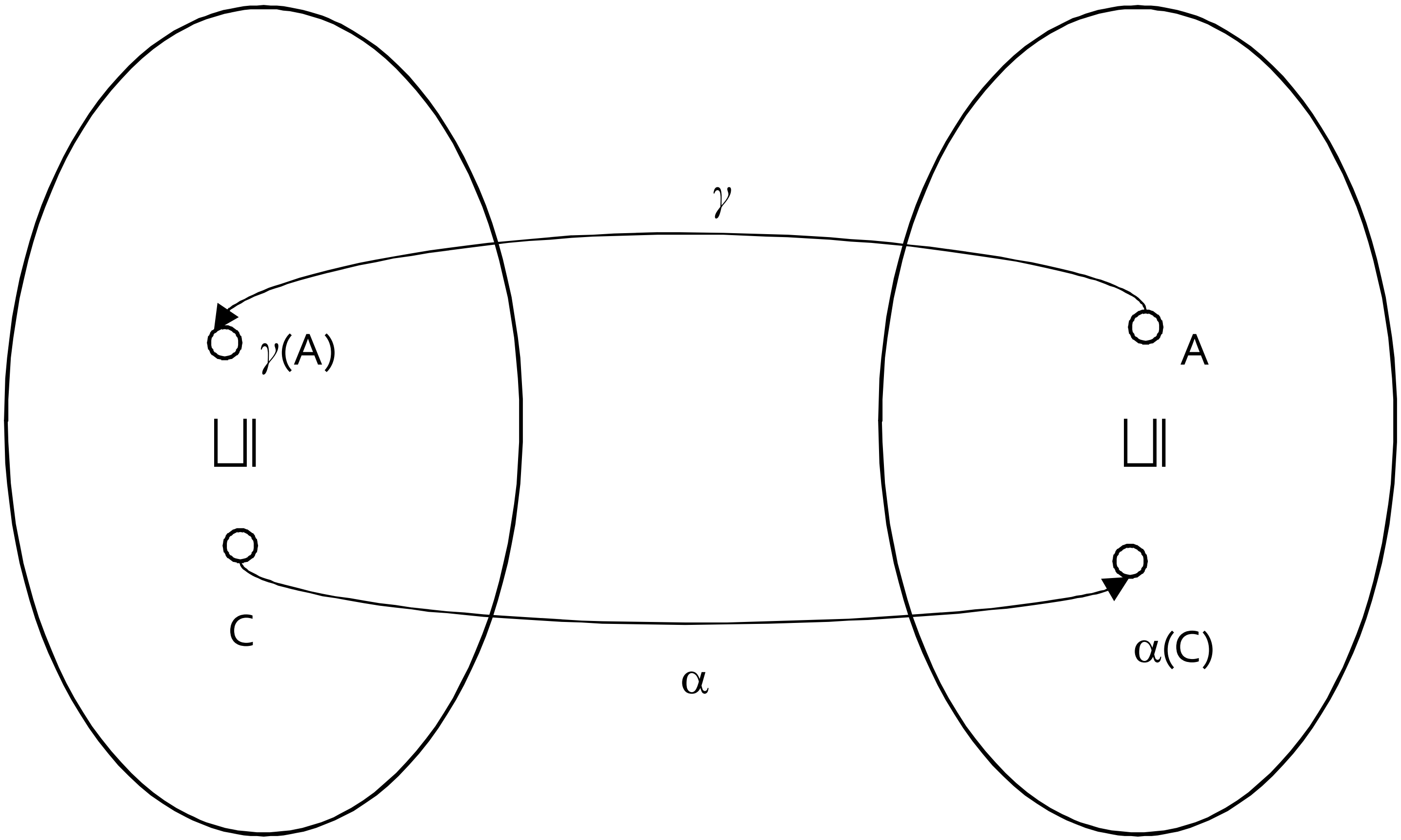
PROGRAM ANALYSIS & SYNTHESIS

Eran Yahav

Previously…

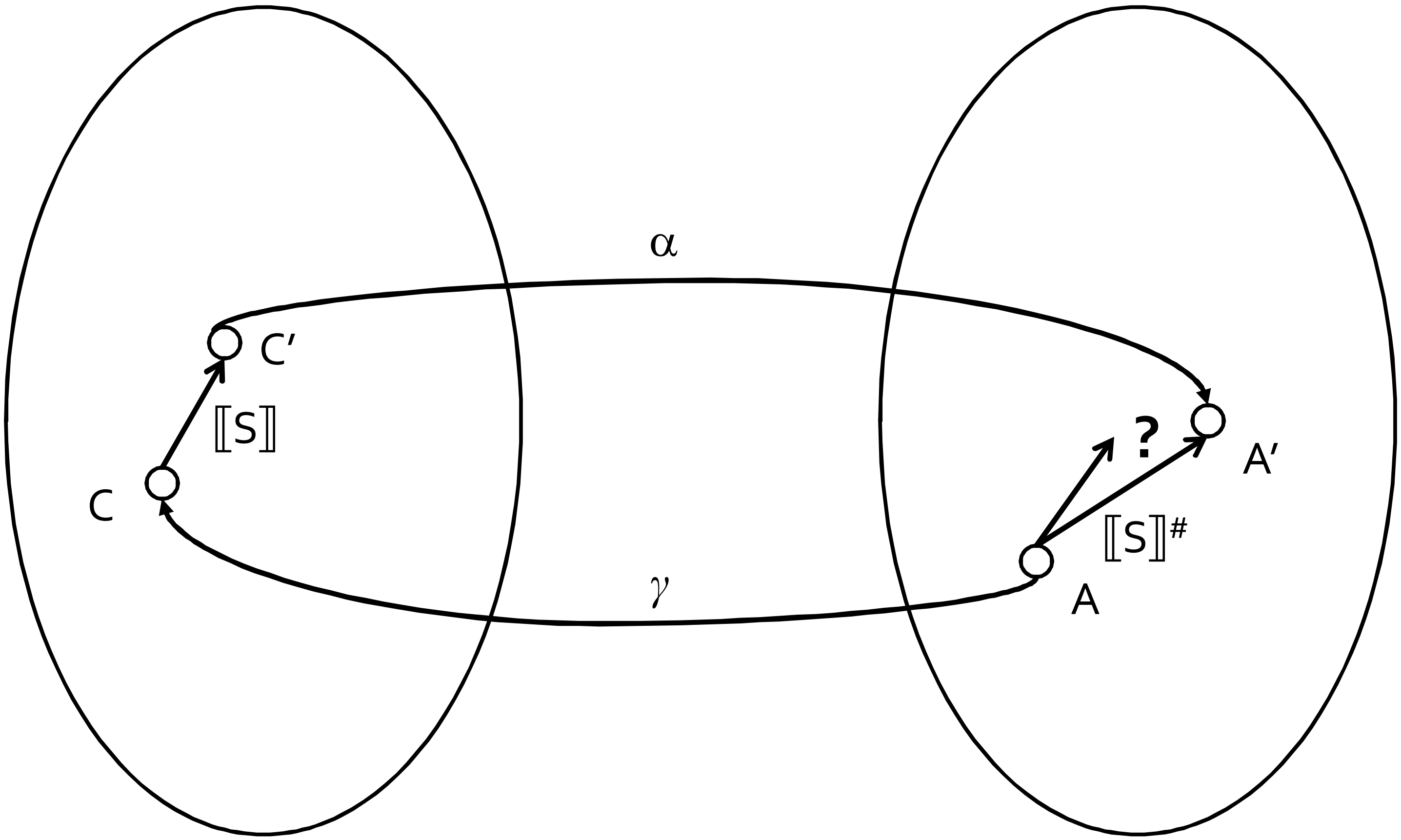
- Trace semantics
- Collecting semantics
- Lattices
- Galois connection
- Least fixed point

Galois Connection



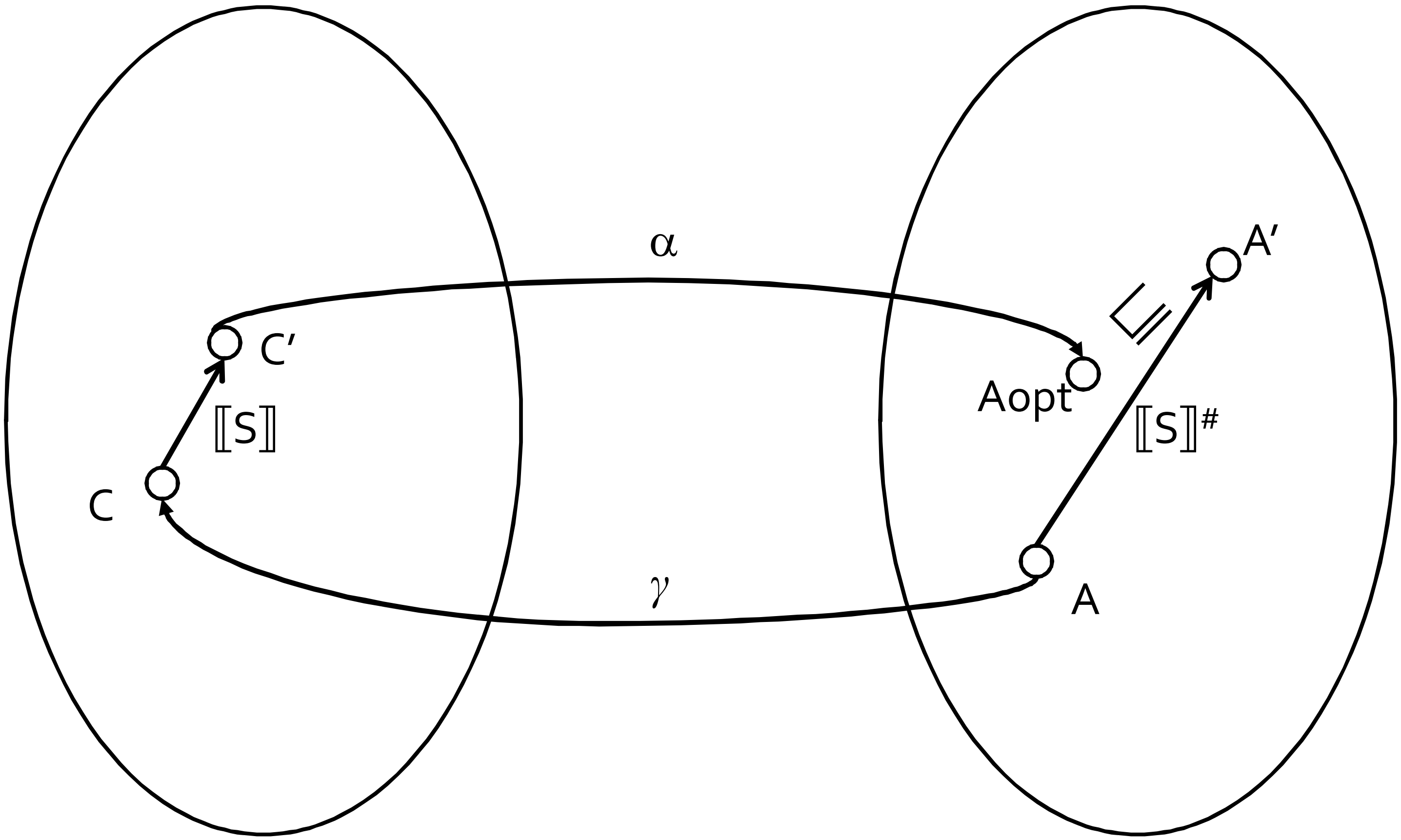
$$\alpha(C) \sqsubseteq A \Leftrightarrow C \sqsubseteq \gamma(A)$$

Best (Induced) Transformer



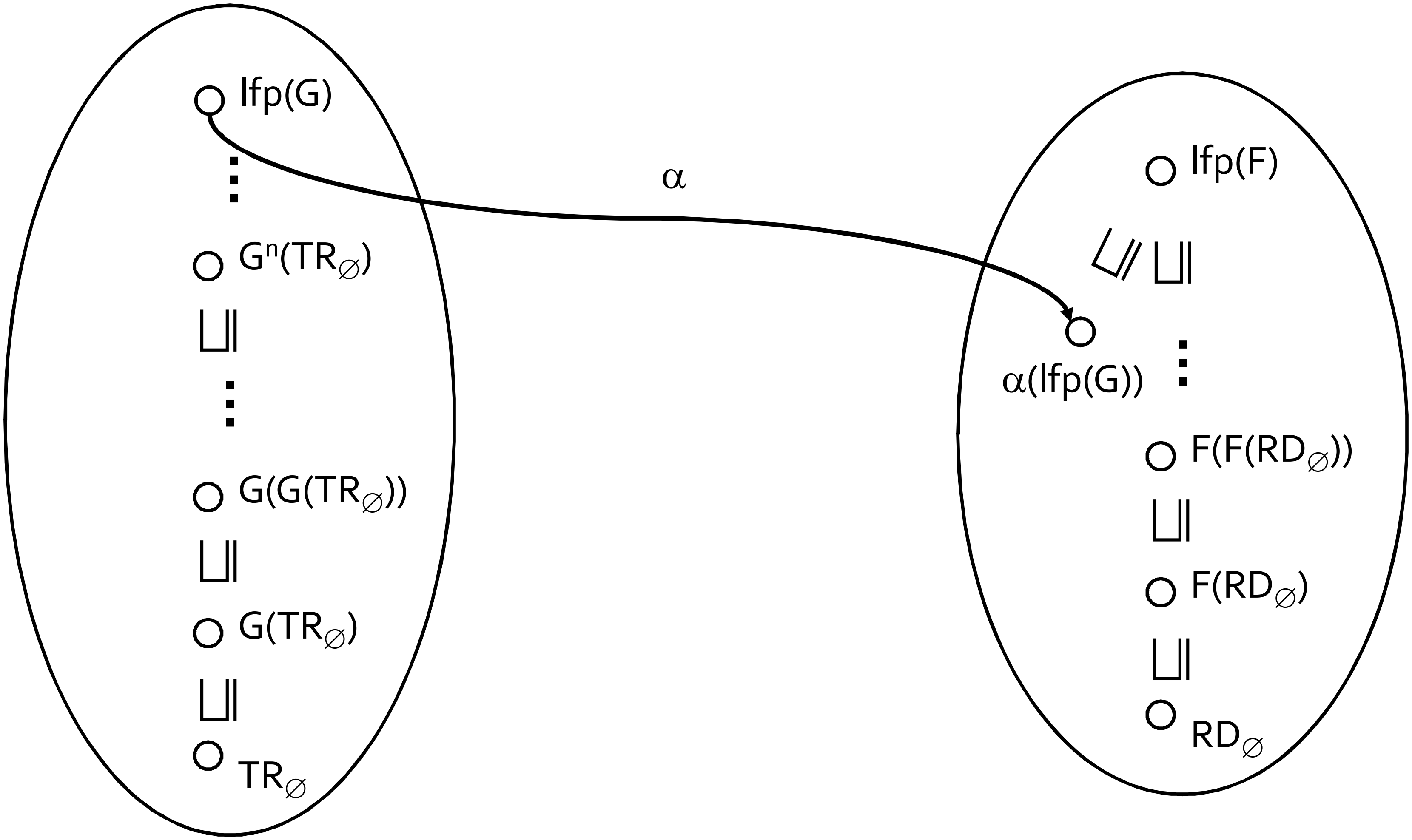
How do we figure out the abstract effect $[\![S]\!]^{\#}$ of a statement S ?

Sound Abstract Transformer



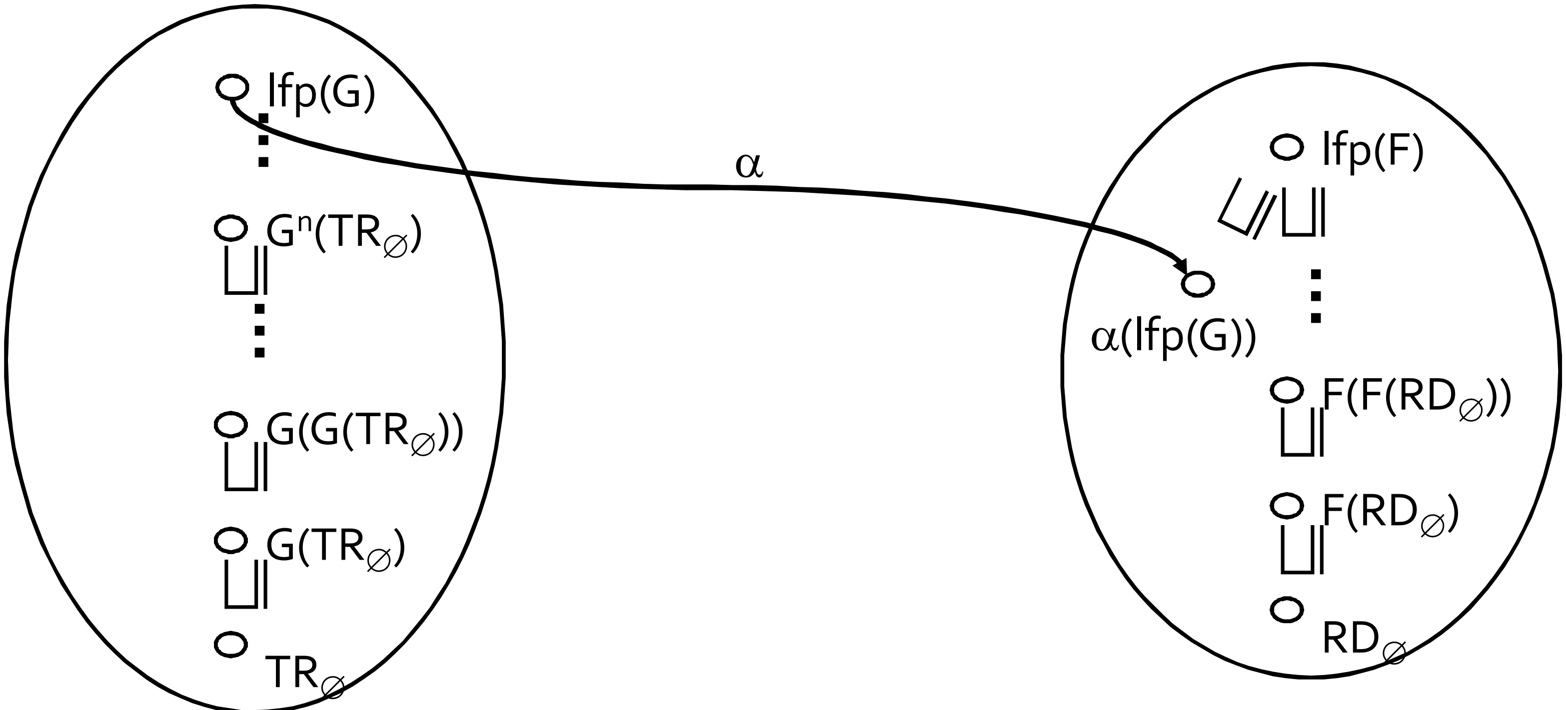
$$\alpha \circ [\![S]\!] \circ \gamma(A) \sqsubseteq [\![S]\!]^{\#}(A)$$

Soundness of Induced Analysis



$$\alpha(\mathrm{lfp}(G)) \sqsubseteq \mathrm{lfp}(\alpha \circ G \circ \gamma) \sqsubseteq \mathrm{lfp}(F)$$

Trivial Example



$$\alpha(\mathrm{lfp}(G)) \sqsubseteq \mathrm{lfp}(\alpha \circ G \circ \gamma) \sqsubseteq \mathrm{lfp}(F)$$

x = 42;
while(?) x++;

x = 42; // $x \mapsto \{+\}$
while(?) x++; // $x \mapsto \{+\}$

is x = 17 possible in lfp(G)? is it in γ(lfp(F))?

Today

- A few more words about Lattices, Galois Connections, and friends
- Numerical Abstractions
- parity
- signs
- constant
- interval
- octagon
- polyhedra

Complete Lattices

- A complete lattice $(L, \sqsubseteq)$ is a poset such that all subsets have least upper bounds as well as greatest lower bounds
- In particular
  - $\perp = \bigsqcup\emptyset = \bigsqcap L$ is the least element (bottom)
  - $\top = \bigsqcup L = \bigsqcap\emptyset$ is the greatest element (top)

- Why do we care? (roughly speaking)
  - use a lattice to represent properties of a program
  - join operation $\sqcup$ handle information reaching a program point from multiple sources
  - meet operation $\sqcap$ handle restrictions (e.g., conditions)

Galois Connection

- Connect two lattices
  - $(C, \sqsubseteq_c)$ representing "concrete" information
  - $(A, \sqsubseteq_a)$ representing abstract information
- Using two functions
  - $\alpha: C \to A$ abstraction function
  - $\gamma: A \to C$ concretization function
- such that
  - $\alpha(C) \sqsubseteq_a A \Leftrightarrow C \sqsubseteq_c \gamma(A)$
- Alternatively
  - α and γ are order-preserving (monotone)
  - $\forall a \in A\ \alpha(\gamma(a)) \sqsubseteq_a a$
  - $\forall c \in C\ c \sqsubseteq_c \gamma(\alpha(c))$

Galois Connection

- Why do we care? (roughly)
  - captures intuition: values in one lattice used to represent values in another lattice in a conservative manner
  - In a Galois Connection α and γ determine one another, enough to define one, and can compute the other
    - $\alpha(c) = \sqcap\{ a \mid c \sqsubseteq \gamma(a) \}$
    - $\gamma(a) = \sqcup\{ c \mid \alpha(c) \sqsubseteq a \}$
  - global soundness theorem allows us to extend "local soundness" of individual operations to soundness of LFP computation

"Analysis Algorithm"

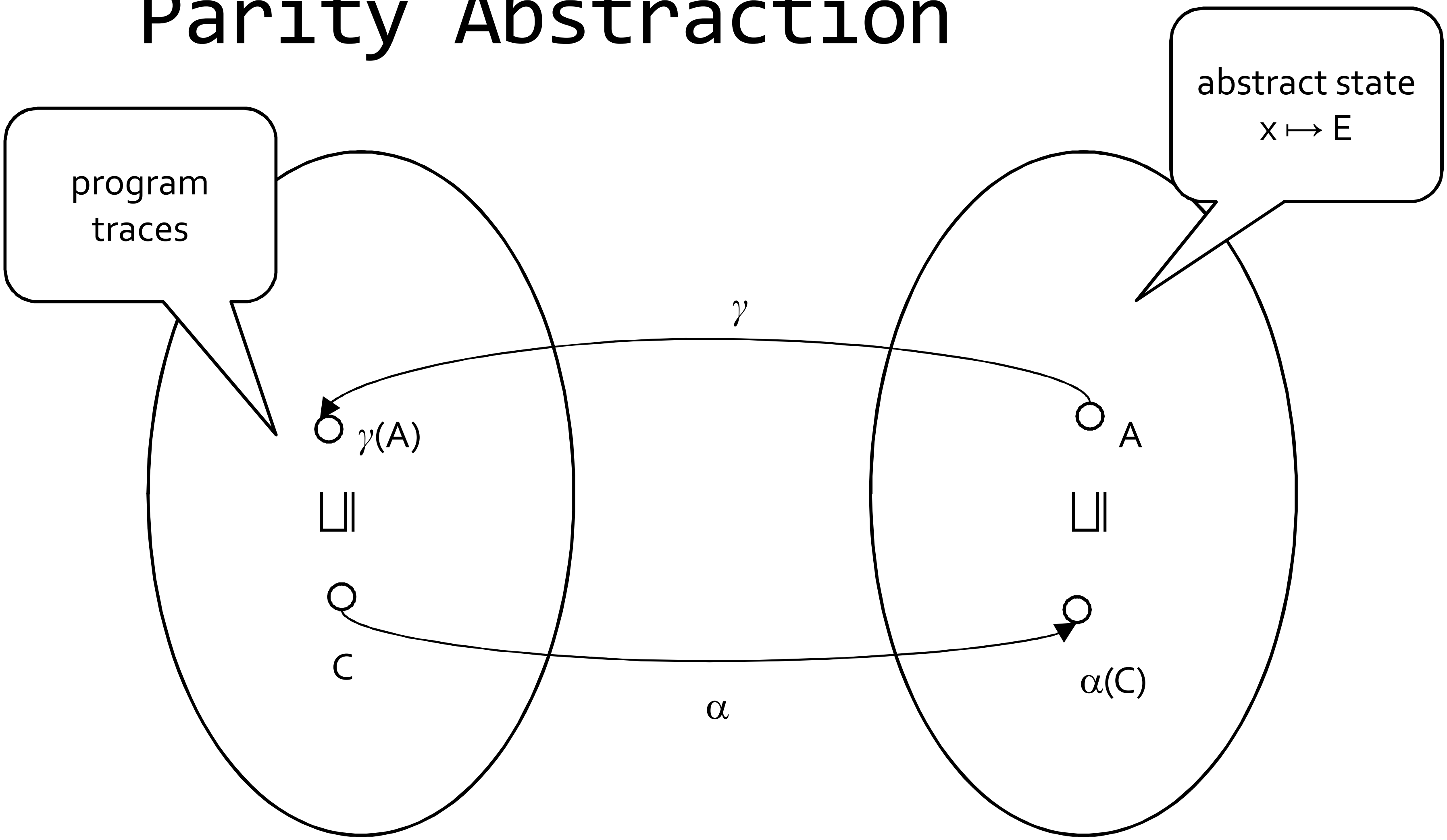
$a = \bot$
while $a \sqsubseteq f(a)$ do $a = f(a)$

- Complete lattice $(A, \sqsubseteq_a, \bot, \top, \sqcup, \sqcap)$
- $[\![S]\!]^{\#}(a)$ the abstract transformer for each statement
- $a_1 \sqcup a_2$ join operation
- $a_1 \sqsubseteq a_2$ check for convergence (fixed point)

Parity Abstraction

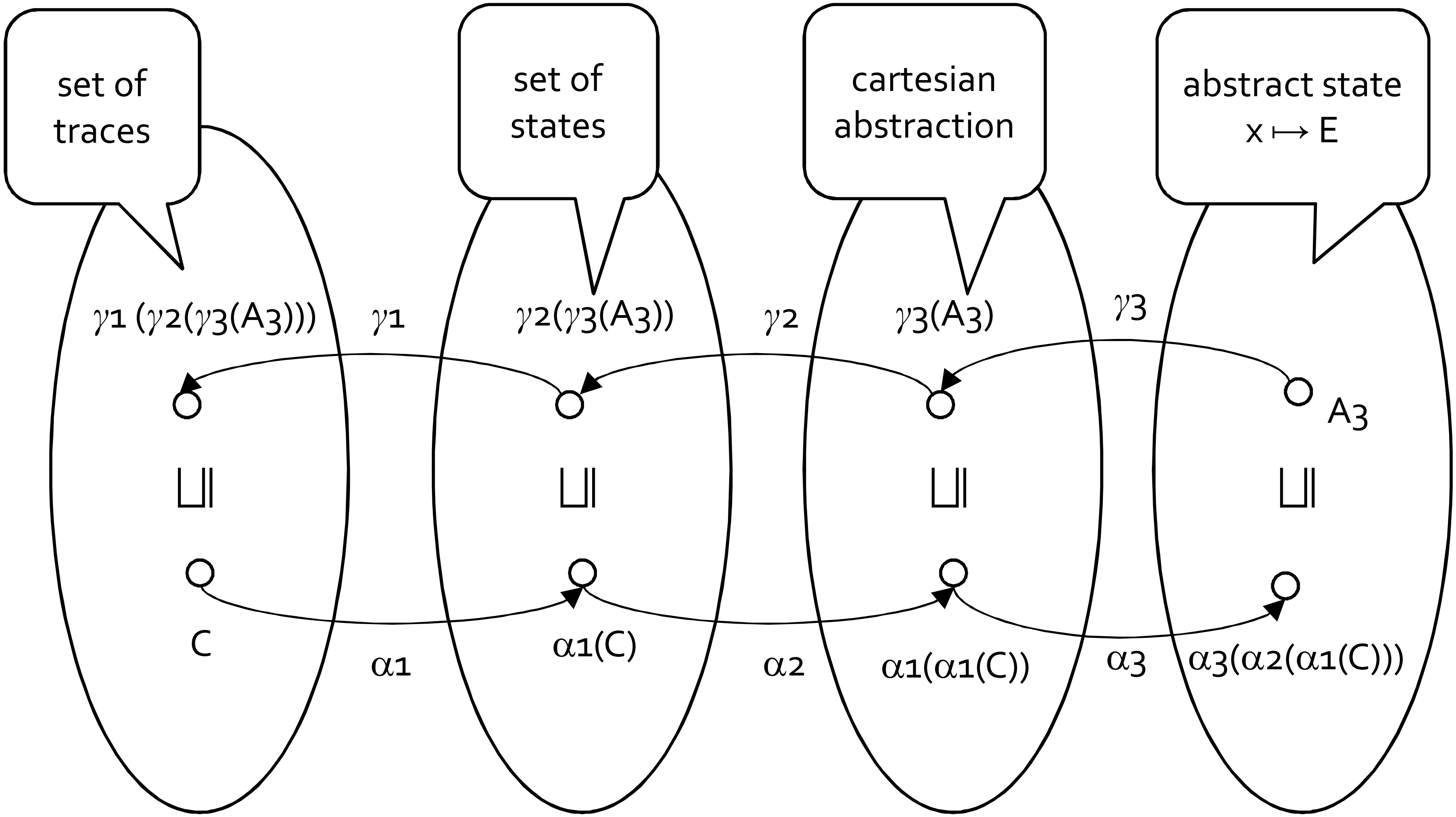
```
while (x !=1 ) do {
 if (x % 2) == 0 {
     x := x / 2;
 } else {
     x := x * 3 + 1;
     assert (x %2 ==0);
 }
}
```

Parity Abstraction



$$\alpha(C) \sqsubseteq A \Leftrightarrow C \sqsubseteq \gamma(A)$$

Parity Abstraction



$$\alpha(C) \sqsubseteq A \Leftrightarrow C \sqsubseteq \gamma(A)$$

Some traces of the example program

<u>x = 3</u>

1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (10)-> 6:assert 10 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (5)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (16)-> 6:assert 16 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (8)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (4)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (2)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (1)

<u>x = 5</u>

1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (16)-> 6:assert 16 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (8)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (4)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (2)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (1)

<u>x = 7</u>

1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (22)-> 6:assert 22 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (11)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (34)-> 6:assert 34 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (17)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (52)-> 6:assert 52 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (26)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (13)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (40)-> 6:assert 40 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (20)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (10)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (5)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (16)-> 6:assert 16 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (8)-> …

Some traces of the example program

x = 9

```
1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (28)-> 6:assert 28 mod2==0 -
>1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (14)-> 1:x!=1 -> 2:xmod2==0 ->
3:x=x/2 (7)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (22)-> 6:assert 22
mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (11)-> 1:x!=1 ->
2:xmod2!=0 -> 5:x=x*3+1 (34)-> 6:assert 34 mod2==0 ->1:x!=1 ->
2:xmod2==0 -> 3:x=x/2 (17)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (52)-
> 6:assert 52 mod2==0 ->1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (26)->
1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (13)-> 1:x!=1 -> 2:xmod2!=0 ->
5:x=x*3+1 (40)-> 6:assert 40 mod2==0 ->1:x!=1 -> 2:xmod2==0 ->
3:x=x/2 (20)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (10)-> 1:x!=1 ->
2:xmod2==0 -> 3:x=x/2 (5)-> 1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (16)->
6:assert 16 mod2==0 ->…
```

x = 11

```
1:x!=1 -> 2:xmod2!=0 -> 5:x=x*3+1 (34)-> 6:assert 34 mod2==0 -
>1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (17)-> 1:x!=1 -> 2:xmod2!=0 ->
5:x=x*3+1 (52)-> 6:assert 52 mod2==0 ->1:x!=1 -> 2:xmod2==0 ->
3:x=x/2 (26)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (13)-> 1:x!=1 ->
2:xmod2!=0 -> 5:x=x*3+1 (40)-> 6:assert 40 mod2==0 ->1:x!=1 ->
2:xmod2==0 -> 3:x=x/2 (20)-> 1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (10)->
1:x!=1 -> 2:xmod2==0 -> 3:x=x/2 (5)-> 1:x!=1 -> 2:xmod2!=0 ->
5:x=x*3+1 (16)-> 6:assert 16 mod2==0 ->…
```

Collecting Semantics (label 6)

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(10)->6:assert 10 mod2==0
```

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(10)->6:assert 10 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(5)->1:x!=1->2:xmod2!=0->5:x=x*3+1(16)->
6:assert 16 mod2==0
```

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(16)->6:assert 16 mod2==0
```

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0
```

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(11)->1:x!=1->2:xmod2!=0->5:x=x*3+1(34)->
6:assert 34 mod2==0
```

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(11)->1:x!=1->2:xmod2!=0->5:x=x*3+1(34)->6:assert
34 mod2==0->1:x!=1->2:xmod2==0->3:x=x/2(17)->1:x!=1->2:xmod2!=0->
5:x=x*3+1(52)->6:assert 52 mod2==0
```

…

From Set of Traces to Set of States

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(10)->6:assert 10 mod2==0
```

$6: x \mapsto 10$

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(10)->6:assert 10 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(5)->1:x!=1->2:xmod2!=0->5:x=x*3+1(16)->
6:assert 16 mod2==0
```

$6: x \mapsto 16$

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(16)->6:assert 16 mod2==0
```

$6: x \mapsto 16$

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0
```

$6: x \mapsto 22$

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(11)->1:x!=1->2:xmod2!=0->5:x=x*3+1(34)->
6:assert 34 mod2==0
```

$6: x \mapsto 34$

```
1:x!=1->2:xmod2!=0->5:x=x*3+1(22)->6:assert 22 mod2==0->1:x!=1-
>2:xmod2==0->3:x=x/2(11)->1:x!=1->2:xmod2!=0->5:x=x*3+1(34)->6:assert
34 mod2==0->1:x!=1->2:xmod2==0->3:x=x/2(17)->1:x!=1->2:xmod2!=0->
5:x=x*3+1(52)->6:assert 52 mod2==0
```

$6: x \mapsto 52$

…

Set of States

- still unbounded
- can abstract it using parity
- cannot compute it through the concrete semantics
- need to compute directly in the abstract

$6 : x \mapsto 10$

$6 : x \mapsto 16$

$6 : x \mapsto 16$

$6 : x \mapsto 22$

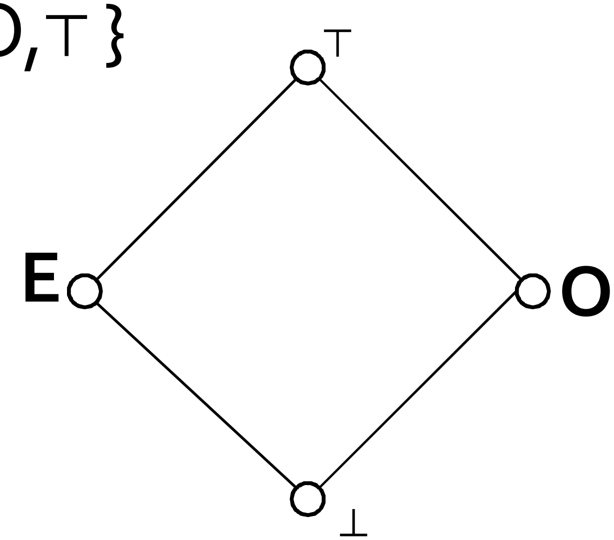
$6 : x \mapsto 34$

$6 : x \mapsto 52$

$\vdots$

Parity Abstraction

- concrete state: Var$\rightarrow$Z
- abstract state: Var $\rightarrow$ { $\perp$,E,O,$\top$ }
  - even / odd
  - $\perp$ non-initialized (bottom)
  - $\top$ either even or odd (top)
- Transformers:
  - $[\![x = x / 2]\!]^{\#}(\sigma)$ = ?
  - $[\![x = x*3 + 1]\!]^{\#}(\sigma)$ =
    - x is even, then $\sigma[x \mapsto O]$
    - x is odd, then $\sigma[x \mapsto E]$



Parity Abstraction

```
1: while (x !=1 ) do {        // x↦{E, O}
2:  if (x % 2) == 0 {
                               // x↦{E}
3:      x := x / 2;            // x↦{E,O}
4:  } else {
                              // x↦{O}
5:      x := x * 3 + 1;        // x↦{E}
6:      assert (x %2 ==0);     // x↦{E}
7:  }
8: }
```

Where does the Galois Connection help me?

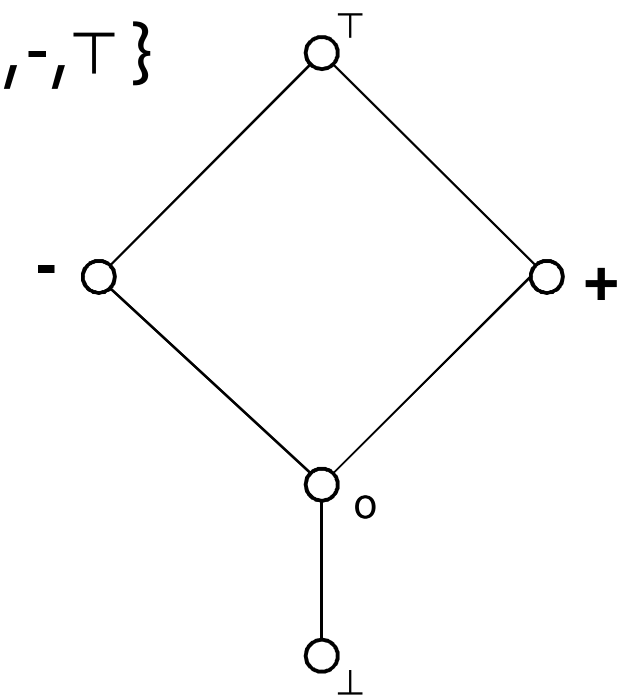
- Establish Galois connection
- Show each individual transformer is sound
- Show each individual transformer is monotonic

- soundness of the analysis is guaranteed

- global soundness theorem

Sign Abstraction

- concrete state: Var$\rightarrow$Z
- abstract state: Var $\longrightarrow$ { $\perp$,o,+,-,$\top$ }
  - zero, positive, negative
  - $\perp$ non-initialized (bottom)
  - $\top$ (top)



Example

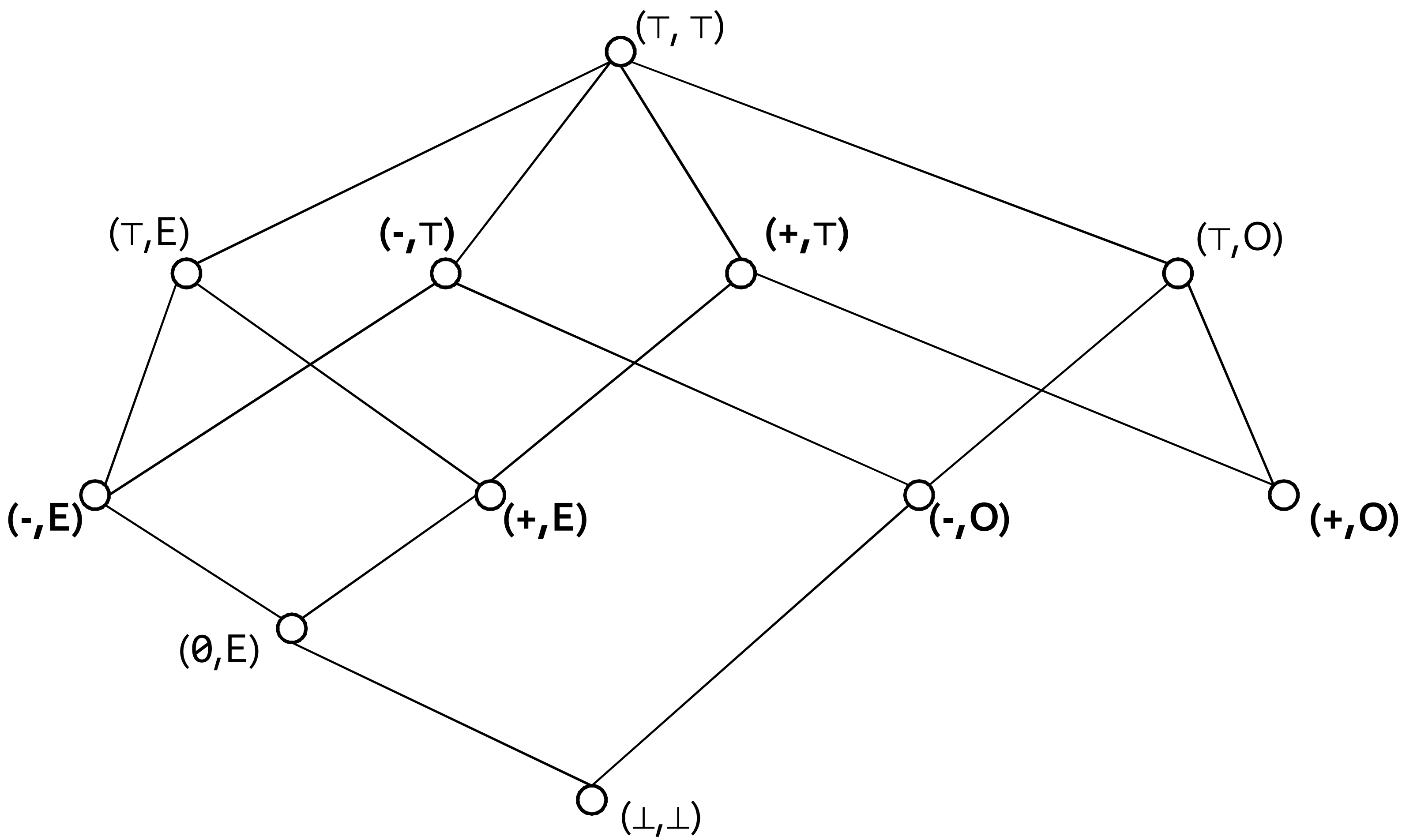
```
main(int i) {
  int x=3,y=1;

  do {
    y = y + 1;
  } while(--i > 0)
  assert 0 < x + y
}
```

x	y	i
⊥	⊥	⊤
+	+	⊤
+	+	⊤
+	+	⊤
+	+	⊤

x	y	i
+	+	⊤
+	+	⊤
+	+	⊤

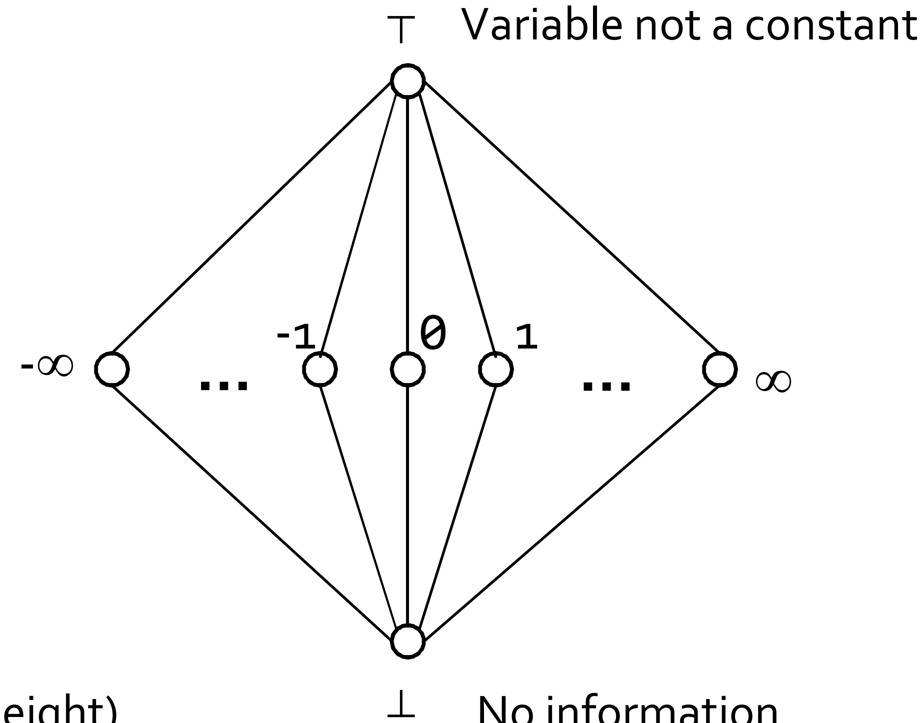
✓

Sign and Parity



(slide from Patrick Cousot)

Constant Abstraction



(infinite lattice, finite height)

$$\sigma \in \text{State} = (\text{Var} \longrightarrow \text{Z}^{\top})_{\perp}$$

Constant Abstraction

- $L = ((Var \longrightarrow Z^{\top})_{\perp}, \sqsubseteq)$
- $\sigma 1 \sqsubseteq \sigma 2$ iff $\forall v: \sigma 1(v) \sqsubseteq \sigma 2(v)$
- $\sqsubseteq$ ordering in the $Z^{\top}{}_{\perp}$ lattice

- Examples:
- $[x \mapsto \perp, y \mapsto 42, z \mapsto \perp] \sqsubseteq [x \mapsto \perp, y \mapsto 42, z \mapsto 73]$
- $[x \mapsto \perp, y \mapsto 42, z \mapsto 73] \sqsubseteq [x \mapsto \perp, y \mapsto 42, z \mapsto \top]$

Constant Abstraction

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

$$A[\![x]\!]\sigma = \begin{cases} \perp \text{ If } \sigma = \perp \\ \sigma(x) \text{ otherwise} \end{cases} \qquad A[\![n]\!]\sigma = \begin{cases} \perp \text{ If } \sigma = \perp \\ n \text{ otherwise} \end{cases}$$

$$A[\![a1 \text{ op } a2]\!]\sigma = A[\![a1]\!]\sigma \text{ op } A[\![a2]\!]\sigma$$

Stmt	**Meaning**
$[x := a]^{lab}$	$\perp$ If $\sigma = \perp$ $\sigma[x \mapsto A[\![a]\!]\sigma]$ otherwise
$[skip]^{lab}$	σ
$[b]^{lab}$	σ

Example

$[X \mapsto \bot, y \mapsto \bot, z \mapsto \bot, w \mapsto \bot]$

$[x := 42]^1;$

$[X \mapsto 42, y \mapsto \bot, z \mapsto \bot, w \mapsto \bot]$

$[y := 73]^2;$

$[X \mapsto 42, y \mapsto 73, z \mapsto \bot, w \mapsto \bot]$

$($if $[?]^3$ then

$[X \mapsto 42, y \mapsto 73, z \mapsto \bot, w \mapsto \bot]$

  $[z := x+y]^4$

$[X \mapsto 42, y \mapsto 73, z \mapsto 115, w \mapsto \bot]$

else

  $[z := 12]^5$

$[X \mapsto 42, y \mapsto 73, z \mapsto 12, w \mapsto \bot]$

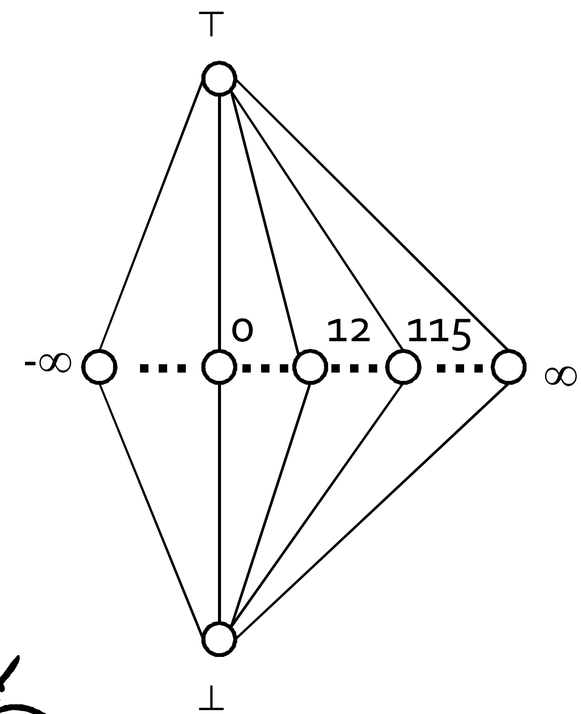
  $[w := z]^6$

$\sqcup$

$[X \mapsto 42, y \mapsto 73, z \mapsto 12, w \mapsto 12]$

$);$

$[X \mapsto 42, y \mapsto 73, z \mapsto \top, w \mapsto 12]$



Constant Propagation is Non Distributive

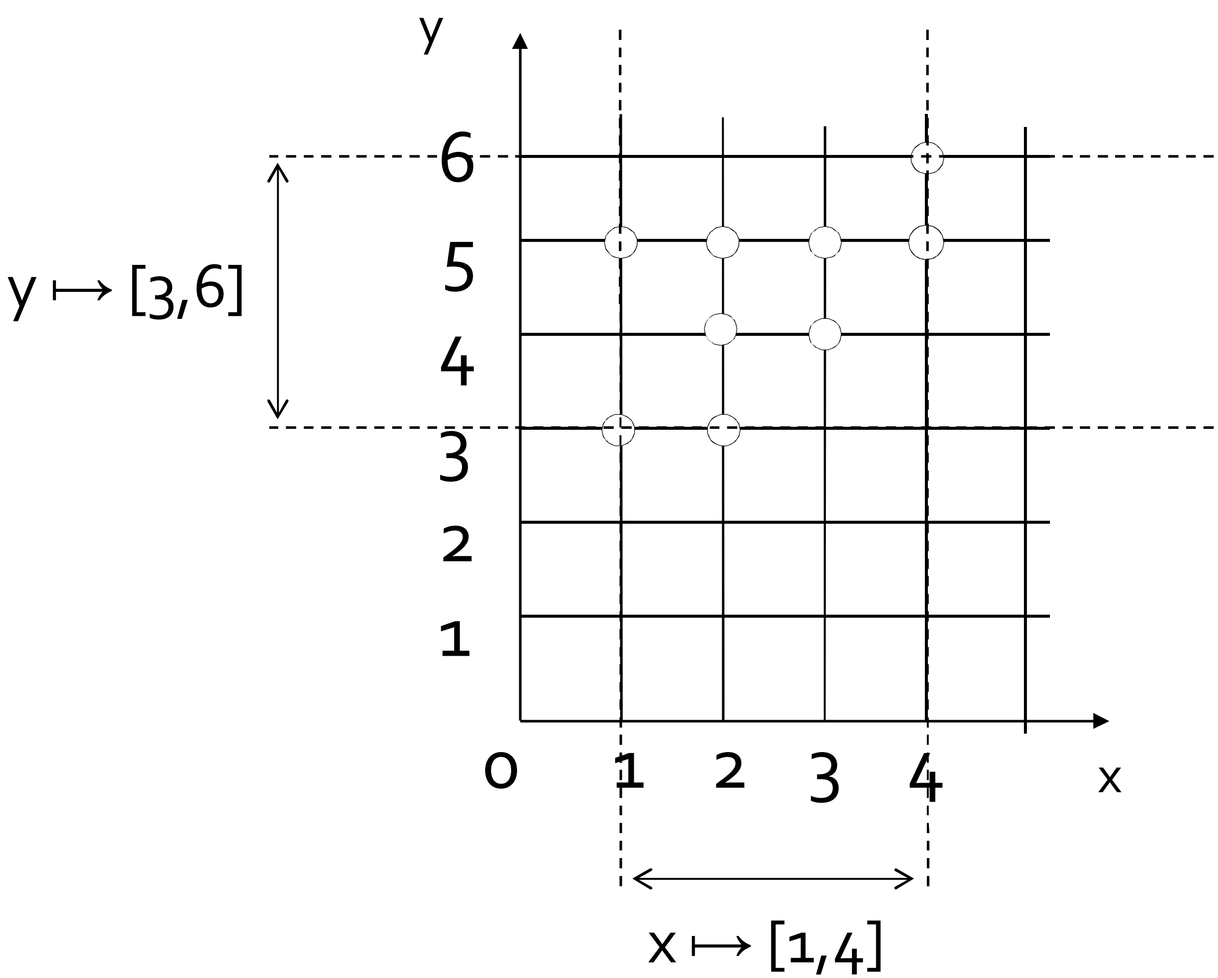
- Consider the transformer $f = [\![[y=x*x]]\!]^\#$
- Consider two states $\sigma 1$, $\sigma 2$
  - $\sigma 1(x) = 1$
  - $\sigma 2(x) = -1$

$(\sigma 1 \sqcup \sigma 2)(x) = \top$
$f(\sigma 1 \sqcup \sigma 2)$ maps y to $\top$

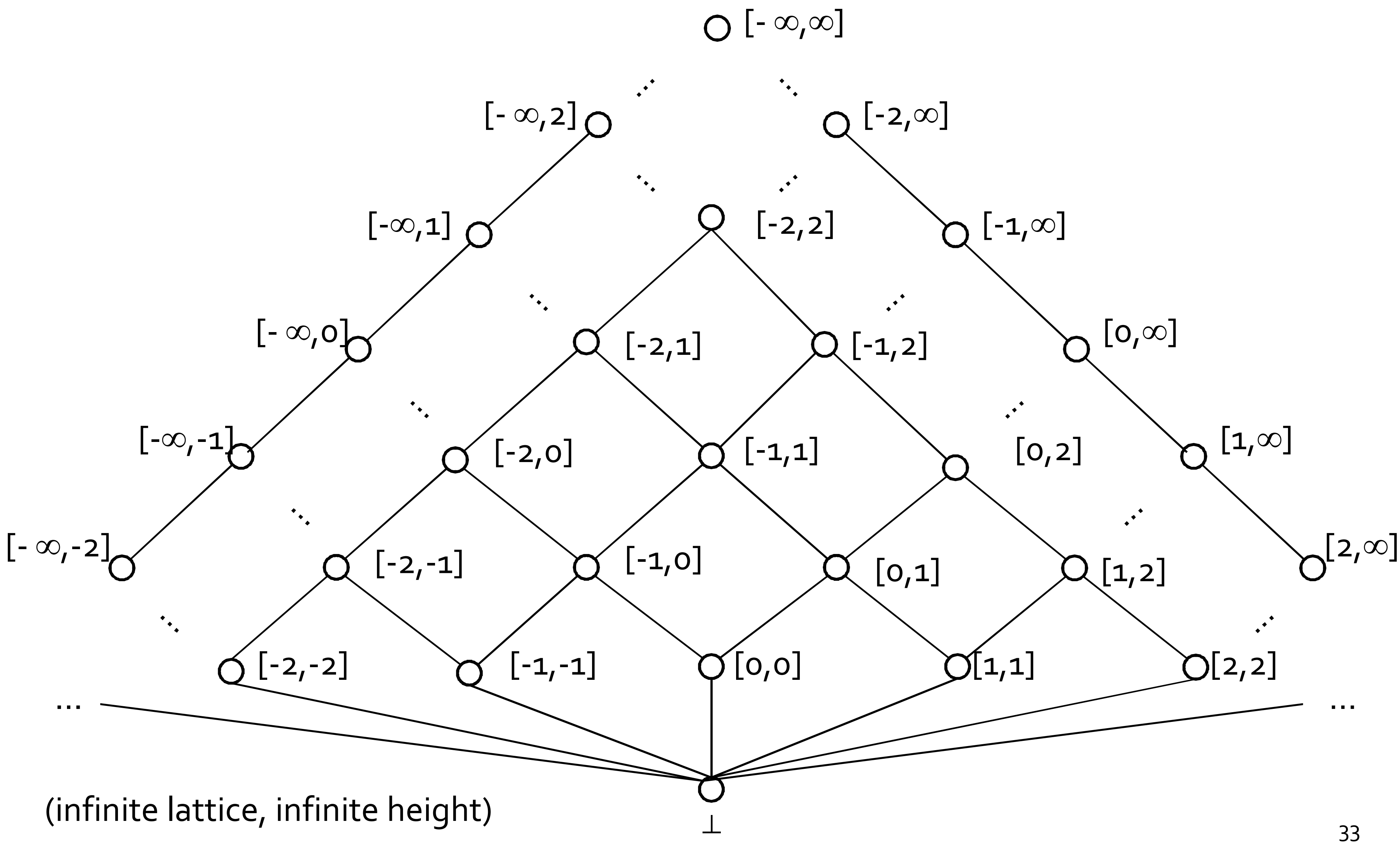
$f(\sigma 1)$ maps y to 1
$f(\sigma 2)$ maps y to 1
$f(\sigma 1) \sqcup f(\sigma 2)$ maps y to 1

$$f(\sigma 1 \sqcup \sigma 2) \neq f(\sigma 1) \sqcup f(\sigma 2)$$

Intervals Abstraction



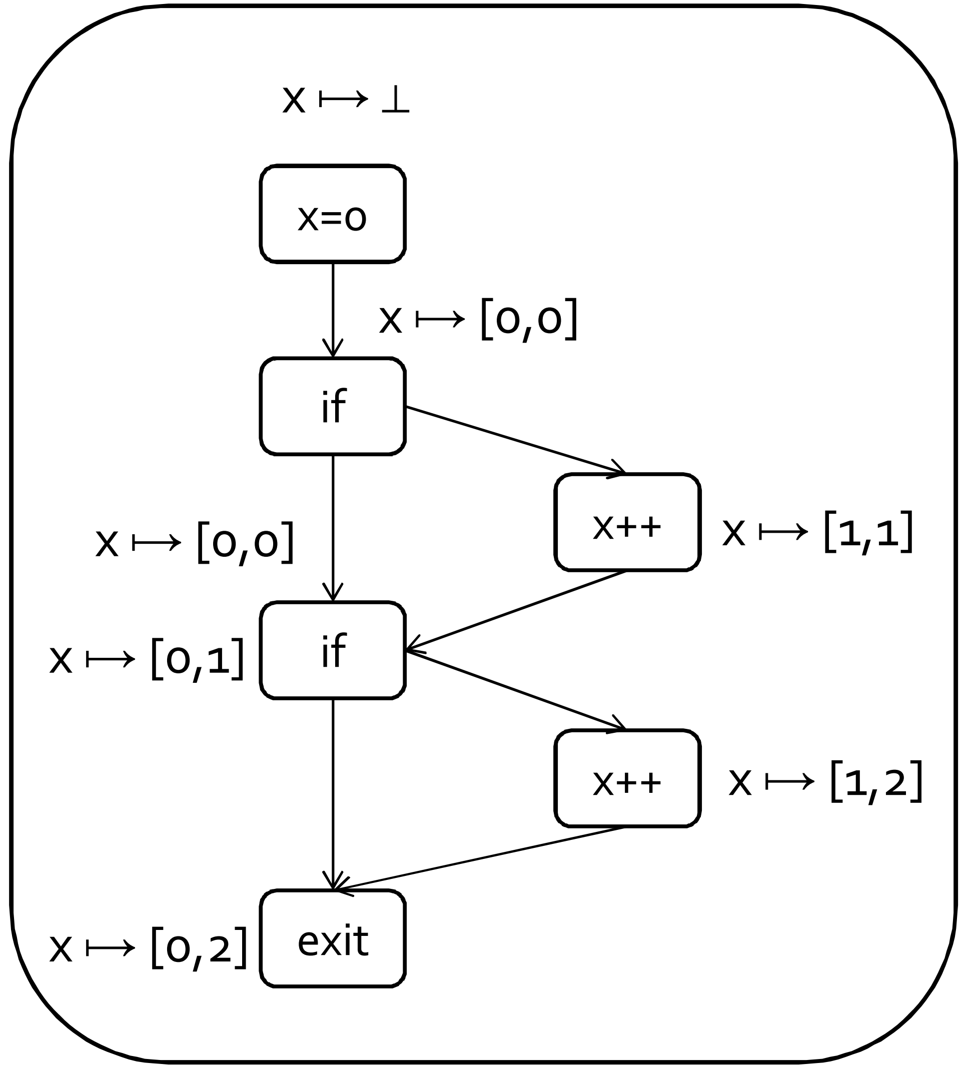
Interval Lattice



(infinite lattice, infinite height)

Example

```
int x = 0;
if (?) x++;
if (?) x++;
```



$[a1,a2] \sqcup [b1,b2] = [\min(a1,b1), \max(a2,b2)]$

Example

```
int x = 0;
while(?) x++;
```

[0,0]

[0,1] [0,2] [0,3] [0,4] [0,5] …

What now?

Widening

- Idea: replace join operator with a more conservative operator that will guarantee convergence
- a.k.a. acceleration

- Complete lattice $(A, \sqsubseteq)$
- A function $\nabla: A \times A \rightarrow A$ is a widening operator iff
  - for every two elements $a1,a2 \in A$, $a1 \sqcup a2 \sqsubseteq a1 \nabla a2$
  - and for every increasing chain $x0,x1,\ldots \in A$ the increasing chain $y0=x0$, $y_{n+1} = y_n \nabla x_{n+1}$ is finite.

An Algorithm for Computing Over-Approximation of lfp

above the lfp

use widening instead of join

$$\mathrm{lfp}(f) \sqsubseteq \nabla_{n \in N} f^n(\bot)$$

$l = \bot$

while $f(l) \neq l$ do $l = f(l)$

- guarantee convergence even in infinite-height lattices with widening

Widening

- useful also in the finite case

```
int x = 0;
while(x<10000) x++;
```

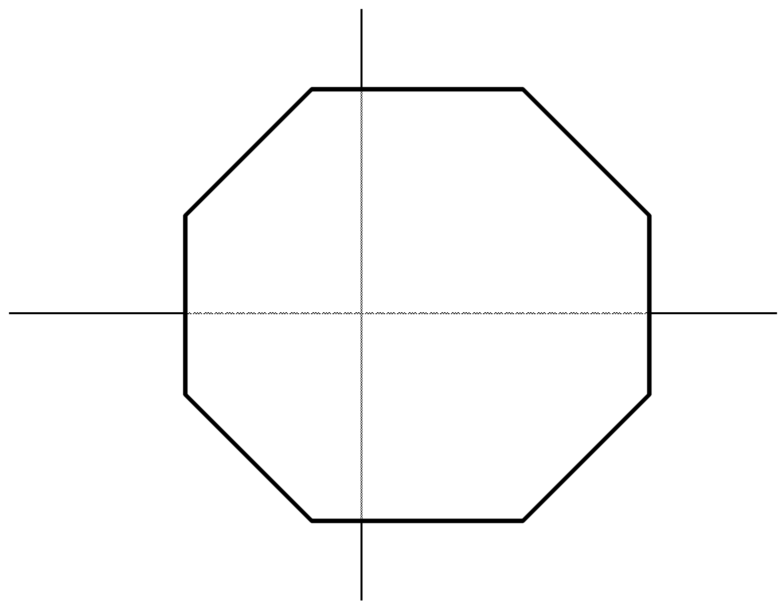
Cartesian vs. Relational Abstractions

- cartesian (also called independent-attribute) abstraction abstracts each variable separately
  - set of points abstracted by a point of sets
    { (1,2), (3,4), (5,6) } => ({1,3,5},(2,4,6)}
    losing relationship between variables
  - e.g., intervals, constants, signs, parity

- relational abstraction tracks relationships between variables

Octagon Abstraction

- abstract state is an intersection of linear inequalities of the form $\pm x \pm y \leq c$



- captures relationships common in programs (array access)

Example

```
proc incr (x:int) returns (y:int)
begin
  y = x+1;
end

var i:int;
begin
  i = 0;
  while (i<=10) do
    i = incr(i);
  done;
end
```

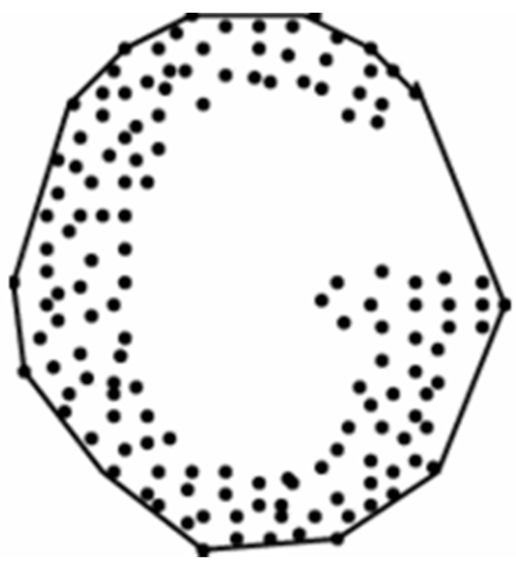
Result with Octagon

```
proc incr (x : int) returns (y : int)   {
/*  [|x>=0; -x+10>=0|] */
  y = x + 1;
/* [|x>=0; -x+10>=0; -x+y-1>=0; x+y-1>=0; y-1>=0; -x-y+21>=0;x-y+1>=0;
-y+11>=0|] */
}
begin
  /* top */
  i = 0; /* [|i>=0; -i+11>=0|] */
  while i <= 10 do
    /* [|i>=0; -i+10>=0|] */
    i = incr(i); /* [|i-1>=0; -i+11>=0|] */
  done; /* [|i-11>=0; -i+11>=0|] */
end
```

Polyhedral Abstraction

- abstract state is an intersection of linear inequalities of the form $a_1x_2+a_2x_2+\ldots a_nx_n \leq c$

- represent a set of points by their convex hull



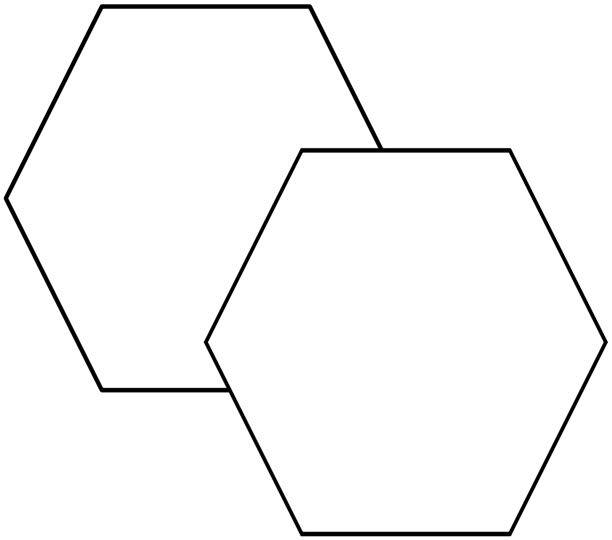
(image from http://www.cs.sunysb.edu/~algorith/files/convex-hull.shtml)

McCarthy 91 function

```
proc MC (n : int) returns (r : int) var t1 : int, t2 : int;
begin
                   /* top */
  if n > 100 then
                   /* [|n-101>=0|] */
     r = n - 10;   /* [|-n+r+10=0; n-101>=0|] */
  else
                   /* [|-n+100>=0|] */
    t1 = n + 11;   /* [|-n+t1-11=0; -n+100>=0|] */
    t2 = MC(t1);   /* [|-n+t1-11=0; -n+100>=0;
                     -n+t2-1>=0; t2-91>=0|] */
    r = MC(t2);    /* [|-n+t1-11=0; -n+100>=0;
                         -n+t2-1>=0; t2-91>=0; r-t2+10>=0;
                         r-91>=0|] */
  endif;           /* [|-n+r+10>=0; r-91>=0|] */
end

var a : int, b : int;
begin /* top */
  b = MC(a);       /* [|-a+b+10>=0; b-91>=0|] */
end
```

if (n>=101) then n-10 else 91

Operations on Polyhedra



Recap

- Cartesian
  - parity – finite
  - signs – finite
  - constant – infinite lattice, finite height
  - interval – infinite height
- Relational
  - octagon – infinite
  - polyhedra – infinite

Back to a bit of dataflow analysis…

Recap

- Represent properties of a program using a lattice $(L,\sqsubseteq,\sqcup,\sqcap,\perp,\top)$
- A continuous function $f: L \longrightarrow L$
  - Monotone function when L satisfies ACC implies continuous
- Kleene's fixedpoint theorem
  - $\mathrm{lfp}(f) = \bigsqcup_{n \in N} f^n(\perp)$
- A constructive method for computing the lfp

Some required notation

blocks : Stmt $\longrightarrow$ P(Blocks)
blocks($[x := a]^{lab}$) = $\{[x := a]^{lab}\}$
blocks($[skip]^{lab}$) = $\{[skip]^{lab}\}$
blocks(S1; S2) = blocks(S1) $\cup$ blocks(S2)
blocks(if $[b]^{lab}$ then S1 else S2) = $\{[b]^{lab}\}$ $\cup$ blocks(S1) $\cup$ blocks(S2)
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 $\longrightarrow \wp$(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

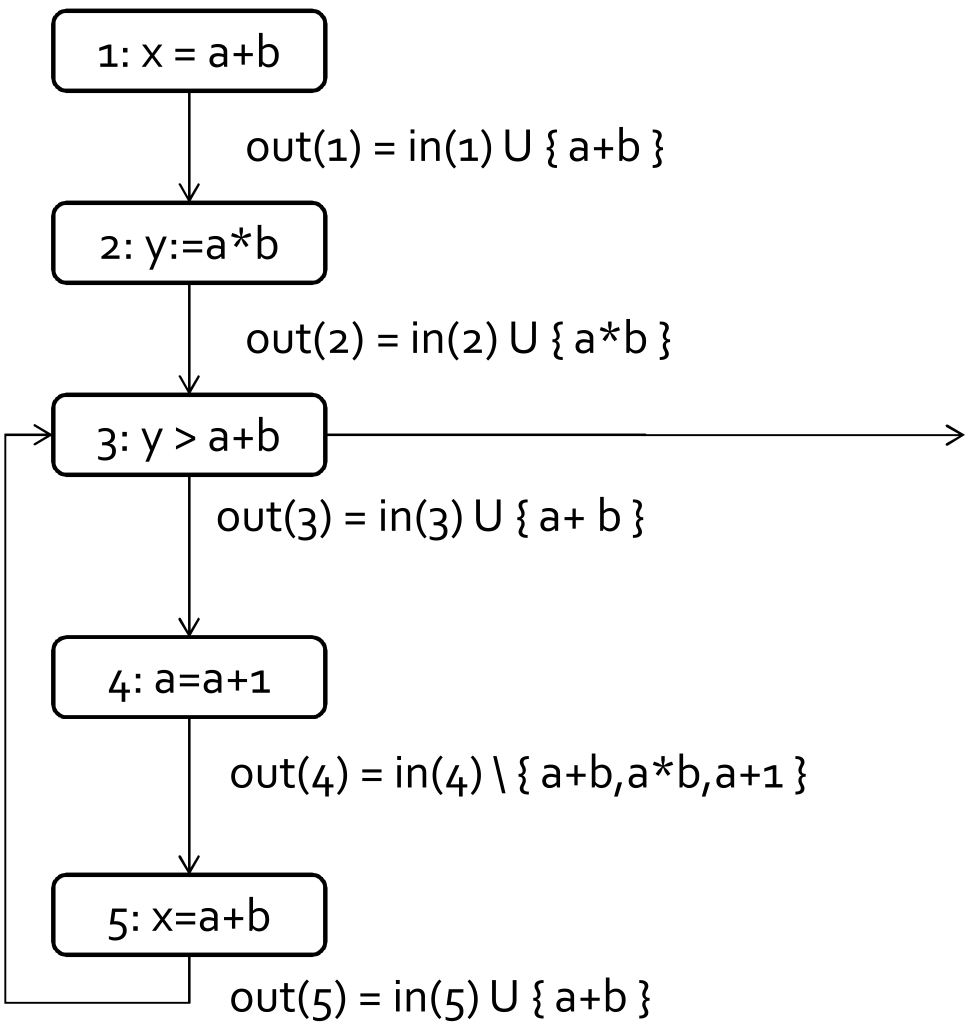
Available Expressions Analysis

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

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

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

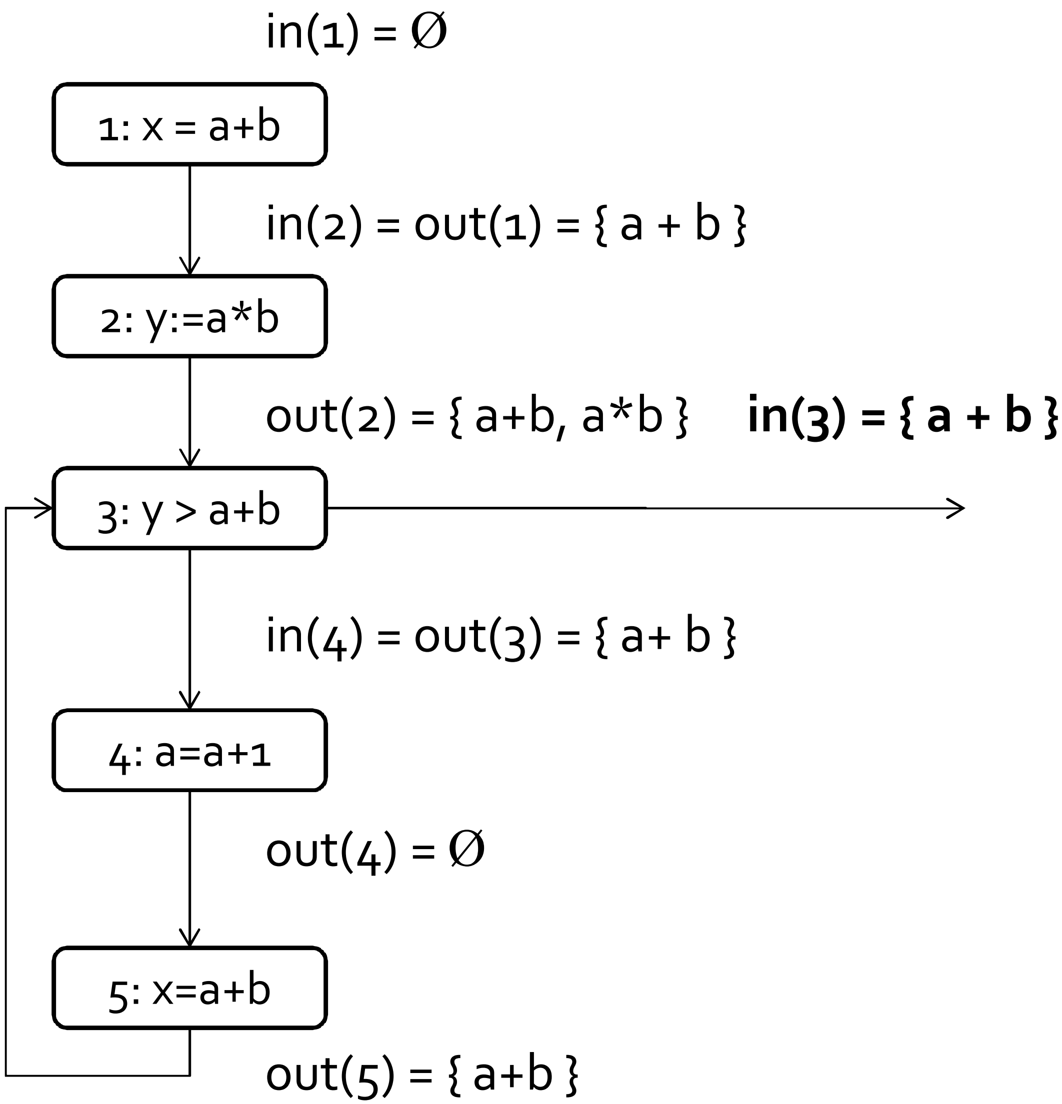
Transfer Functions



in(1) = ∅
in(2) = out(1)
in(3) = out(2) ⋂ 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

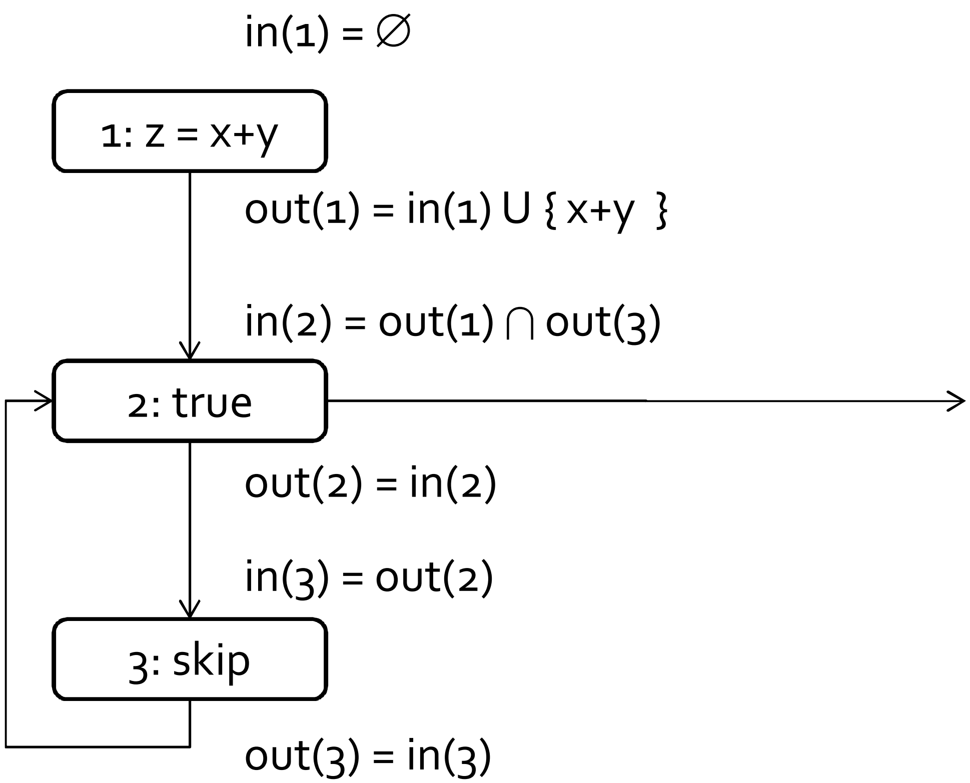
Block	**out (lab)**
$[x := a]^{lab}$	$in(lab) \setminus \{ a' \in AExp \mid x \in FV(a') \} \cup \{ a' \in AExp(a) \mid x \notin FV(a') \}$
$[skip]^{lab}$	$in(lab)$
$[b]^{lab}$	$in(lab) \cup AExp(b)$

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) = ∅
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 ∅

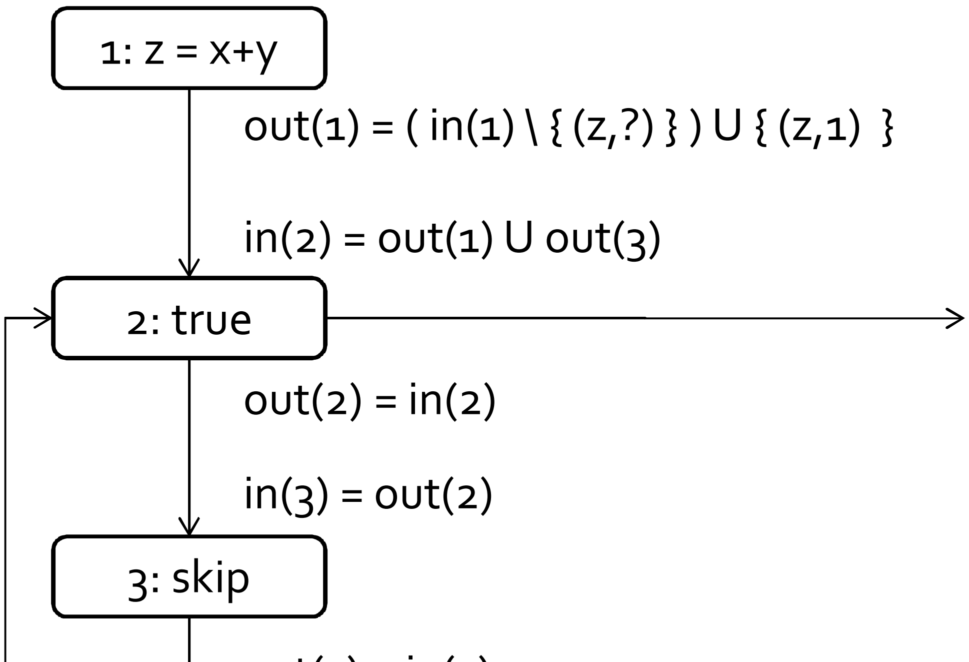
Reaching Definitions Revisited

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 <u>may</u> have been made and not overwritten, when program execution reaches this point along <u>some path</u>.

Why solution with smallest sets?



in(1) = { (x,?),(y,?),(z,?) }

out(1) = (in(1) \ { (z,?) }) U { (z,1) }

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

out(2) = in(2)

in(3) = out(2)

out(3) = in(3)

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) }

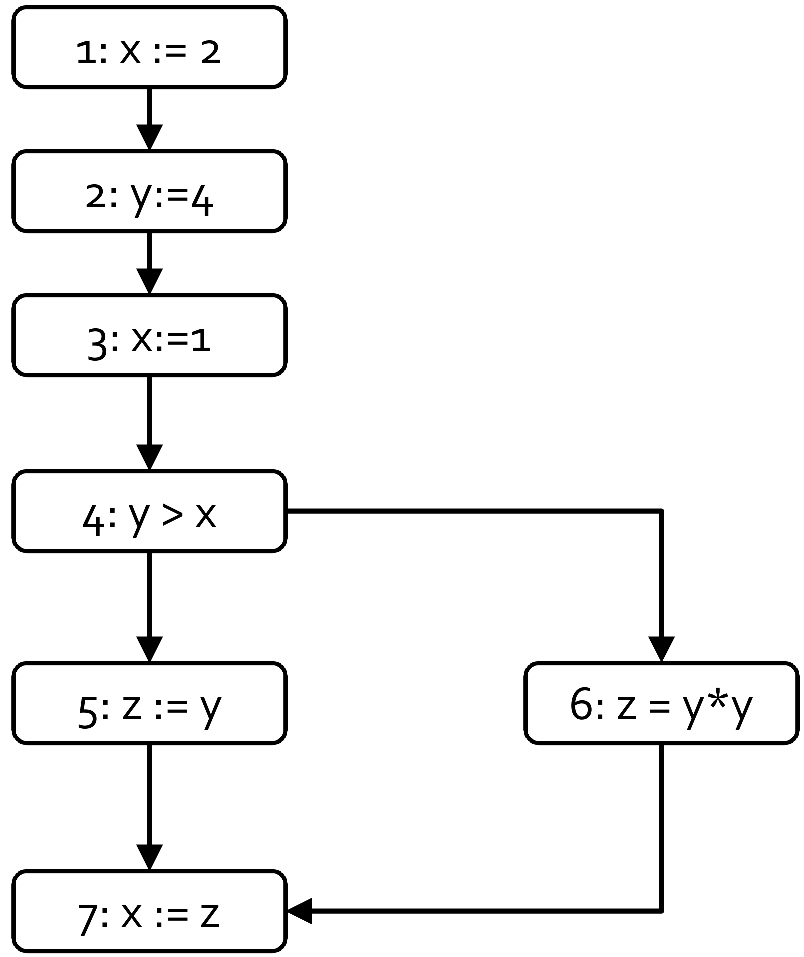
Live Variables

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

For each program point, which variables <u>may</u> be live at the exit from the point.

Live Variables

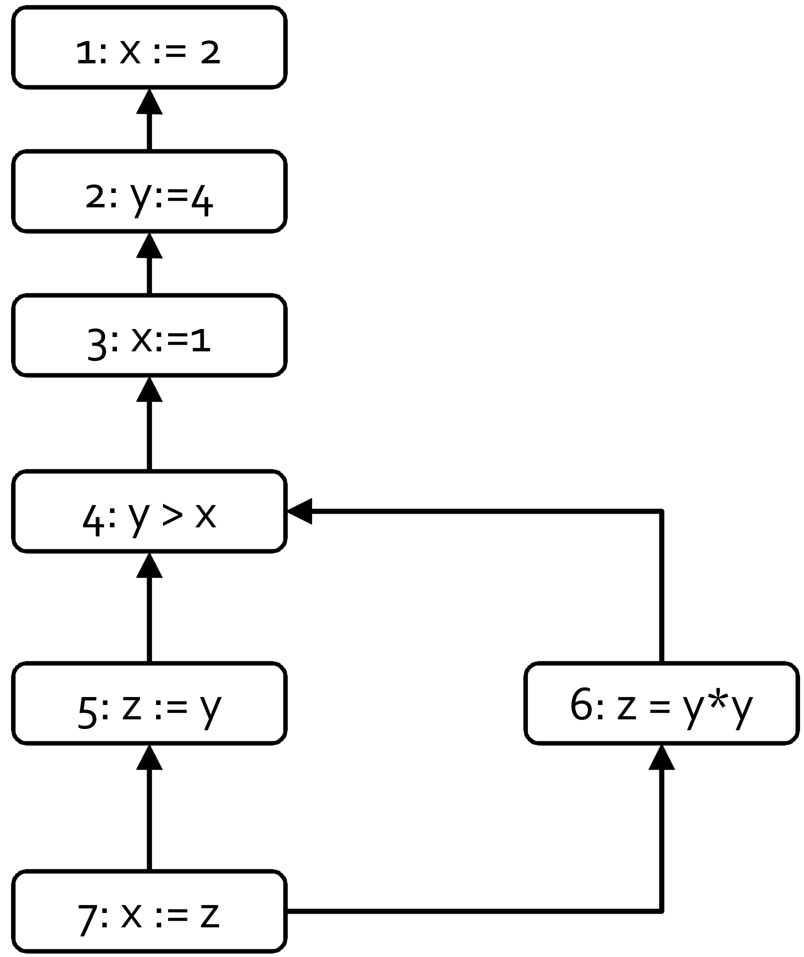
[x :=2]^1;
[y:=4]^2;
[x:=1]^3;
(if [y>x]^4 then [z:=y]^5
else [z:=y*y]^6);
[x:=z]^7



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

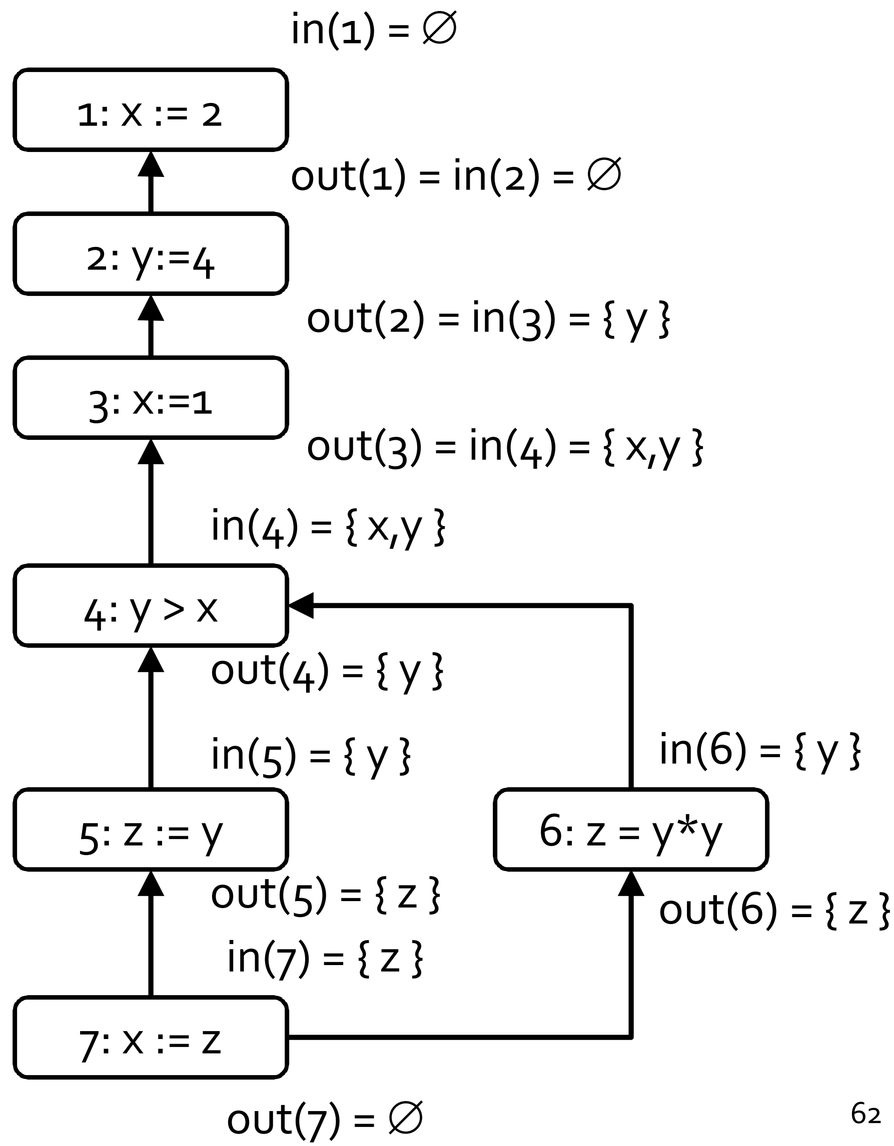
Block	kill	gen
$[x := a]^{lab}$	{ x }	{ FV(a) }
$[skip]^{lab}$	Ø	Ø
$[b]^{lab}$	Ø	FV(b)



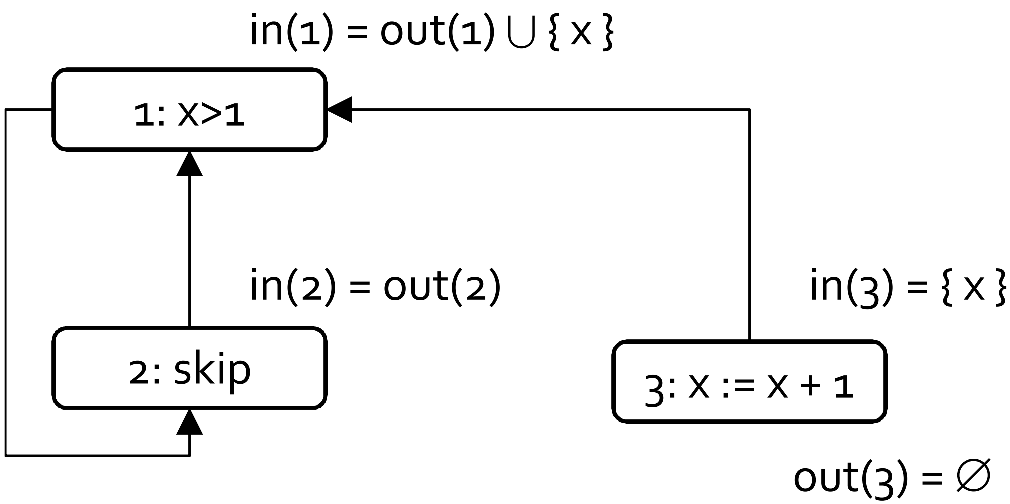
Live Variables: solution

[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}	Ø	Ø
[b]^{lab}	Ø	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 }

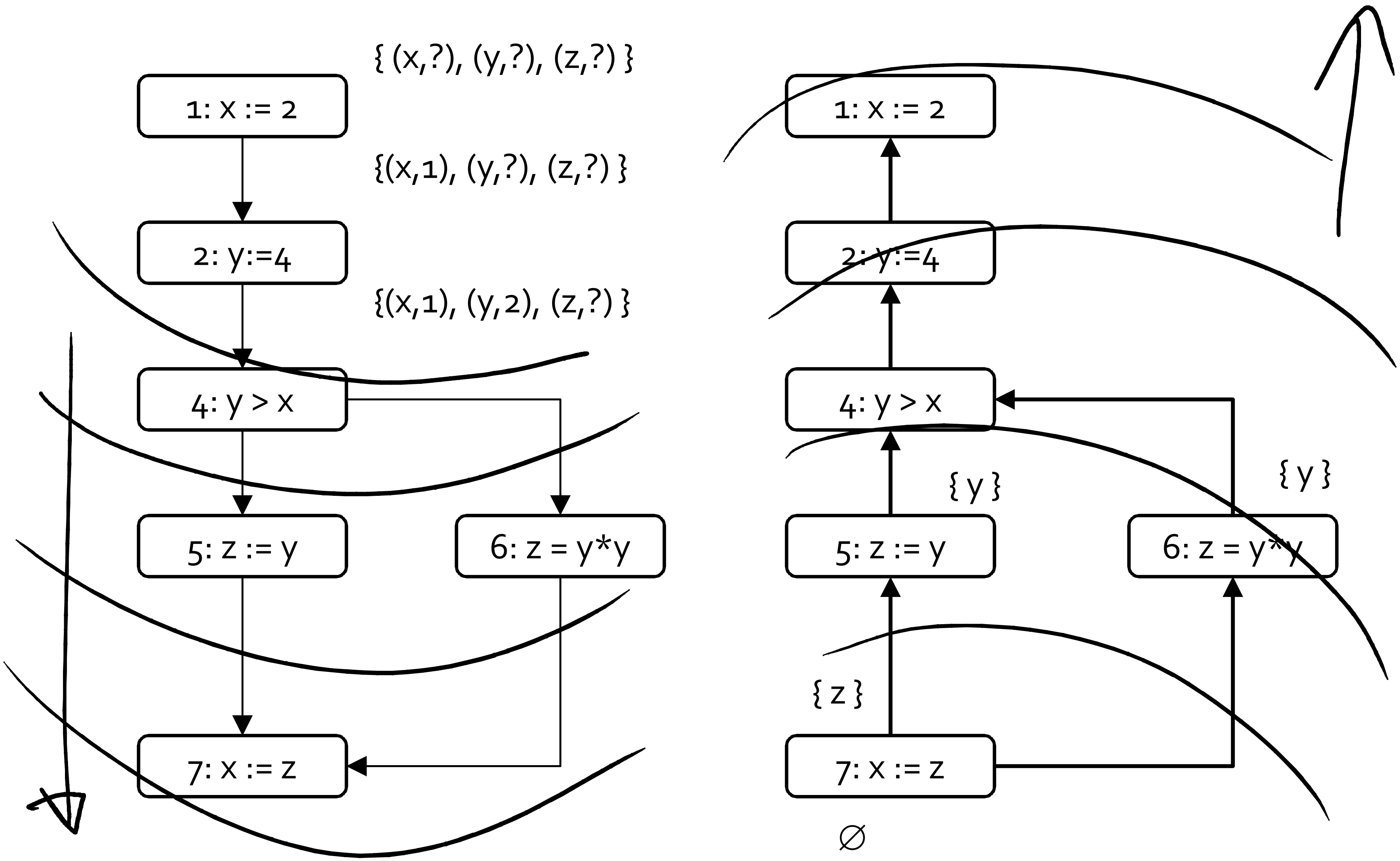
Monotone Frameworks

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

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

- $\bigsqcup$ is $\cup$ or $\cap$
- 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)
$\sqsubseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cup$	$\cap$	$\cup$
$\perp$	$\varnothing$	AExp	$\varnothing$
Initial	{ (x,?) | x $\in$ Var}	$\varnothing$	$\varnothing$
Entry labels	{ init }	{ init }	final
Direction	Forward	Forward	Backward
F	{ f: L $\rightarrow$ L | $\exists$k,g : f(val) = (val \ k) U g }		
f_{lab}	f_{lab}(val) = (val \ 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 $\cap$