Lecture 02 – Structural Operational Semantics (SOS)

PROGRAM ANALYSIS & SYNTHESIS

Eran Yahav

Previously…

- static analysis
  - over-approximation of program behavior
- abstract interpretation
  - abstraction, transformers, fixed-point computation
- examples of program analysis
  - parity abstraction
  - heap abstraction (shape analysis)
  - numerical abstractions
- examples of program synthesis
  - optimized search in program space
  - program repair as a game
  - SKETCH
  - SMARTEdit

Today

- What is the meaning of a program?
- semantics
- structural operational semantics
- intro to abstract interpretation

What is the "meaning" of a program?

```
int foo(int a ) {
  if( 0 < a < 5) c = 42 else c = 73;
  return c;
}
```

```
int a() { printf("a"); return 1; }
int b() { printf("b"); return 2; }
int c() { printf("c"); return 3; }
int sum(int x, int y, int z) { return x+y+z; }
void bar() {
  printf("%d, sum(a(),b(),c());
}
```

Semantics

"mathematical models of and methods for describing and reasoning about the behavior of programs"

Why Formal Semantics?

- implementation-independent definition of a programming language
- automatically generating interpreters (and some day maybe full fledged compilers)
- verification and debugging
  - if you don't know what it does, how do you know its incorrect?

Different Approaches

- Denotational Semantics
  - define an input/output relation that assigns meaning to each construct (denotation)
- Structural Operational Semantics
  - define a transition system, transition relation describes evaluation steps of a program
- Axiomatic Semantics
  - define the effect of each construct on logical statements about program state (assertions)

Denotational Semantics

```
int double1(int x) {
  int t = 0;
  t = t + x;
  t = t + x;
  return t;
}
```

$\lambda x.2*x$

```
int double2(int x) {
  int t = 2*x;
  return t;
}
```

$\lambda x.2*x$

Operational Semantics

```
int double1(int x) {
  int t = 0;
  t = t + x;
  t = t + x;
  return t;
}
```

$x \mapsto 2$

$[t \mapsto 0, x \mapsto 2]$

$[t \mapsto 2, x \mapsto 2]$

$[t \mapsto 4, x \mapsto 2]$

$[t \mapsto 4, x \mapsto 2]$

```
int double2(int x) {
  int t = 2*x;
  return t;
}
```

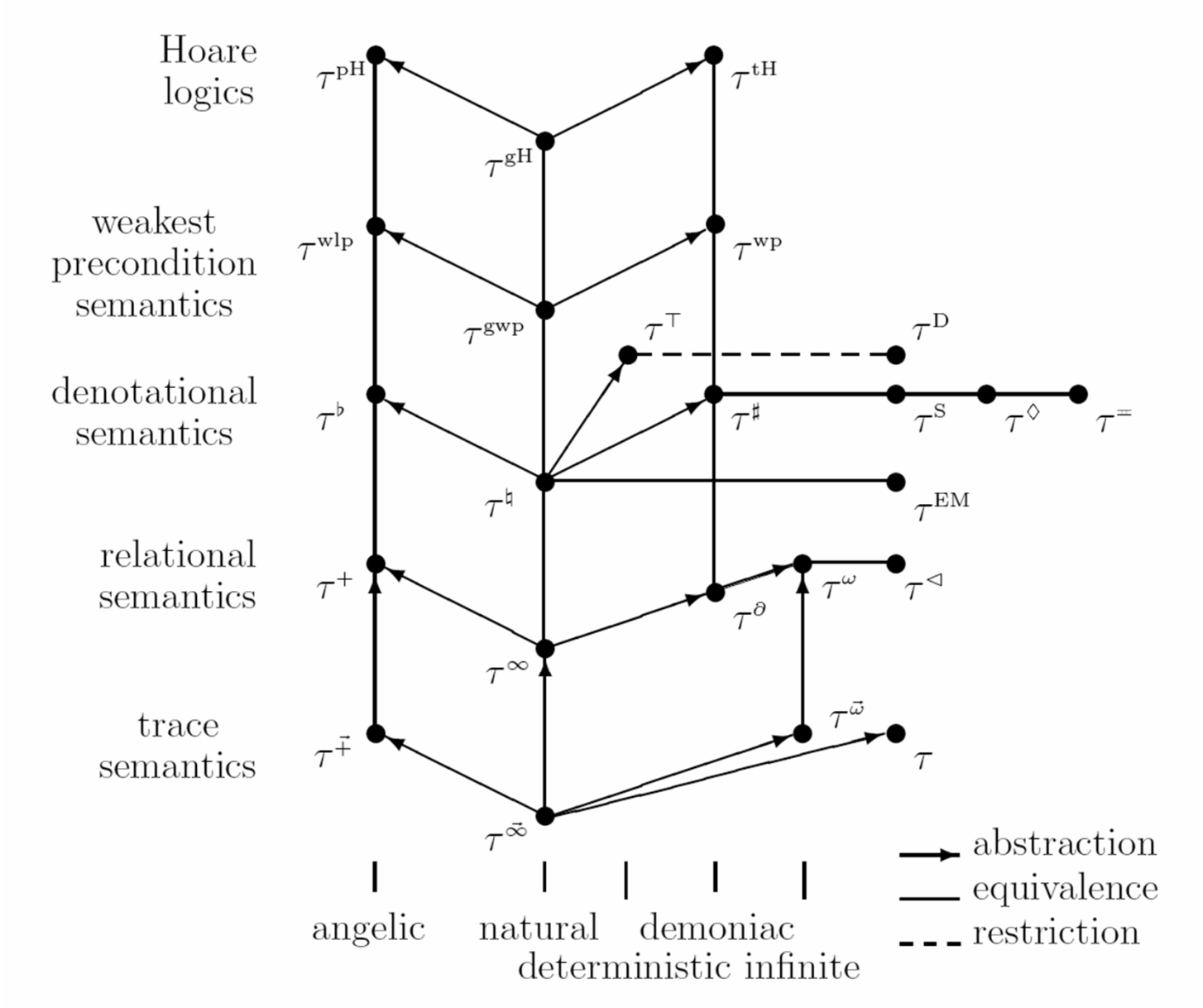
$[t \mapsto 4, x \mapsto 2]$

Axiomatic Semantics

```
int double1(int x) {
 { x = x_0 }
 int t = 0;
 { x = x_0 ∧ t = 0 }
 t = t + x;
{ x = x_0 ∧ t = x_0 }
 t = t + x;
{ x = x_0 ∧ t = 2*x_0 }
 return t;
}
```

```
int double2(int x) {
 { x = x_0 }
 int t = 2*x;
  { x = x_0 ∧ t = 2*x_0 }
 return t;
}
```

Relating Semantics



What is the “meaning” of this program?

```
[y := x]1;
[z := 1]2;
while [y > 0]3 (
  [z := z * y]4;
  [y := y – 1]5;
 )
[y := 0]6
```

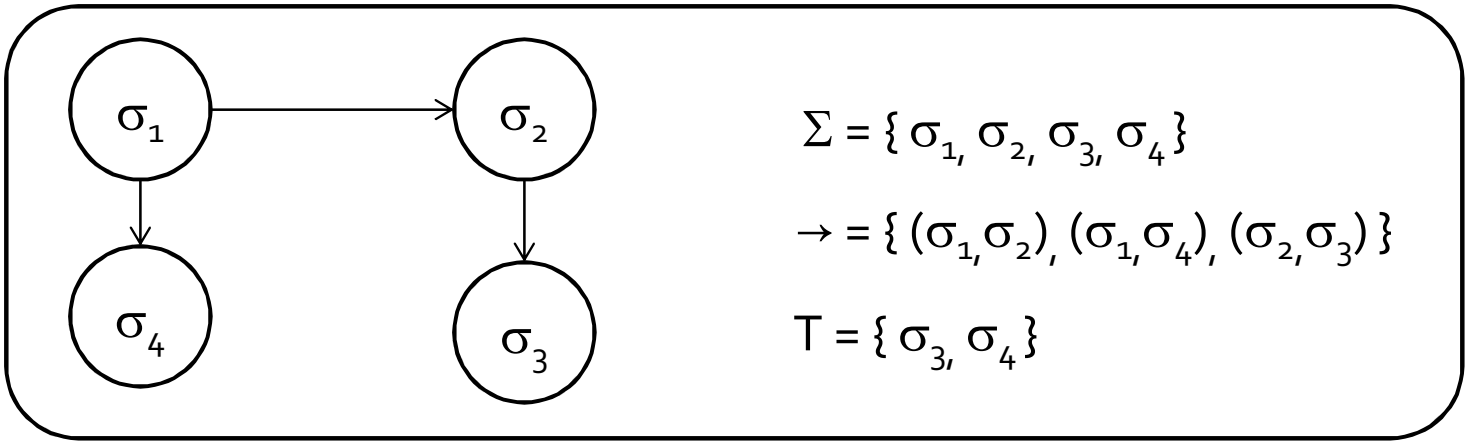
what is the "meaning" of an arithmetic expression?

- z * y
- y – 1

- First: syntax of simple arithmetic expressions
- For now, assume no variables
- a ::= n
  | a1 + a2
  | a1 – a2
  | a1 * a2
  | (a1)

Structural Operational Semantics

- Defines a transition system $(\Sigma, \rightarrow, T)$
  - configurations Σ: snapshots of current state of the program
  - transitions $\rightarrow \subseteq \Sigma \times \Sigma$: steps between configurations
  - final configurations $T \subseteq \Sigma$



Structural Operational Semantics

Useful Notations

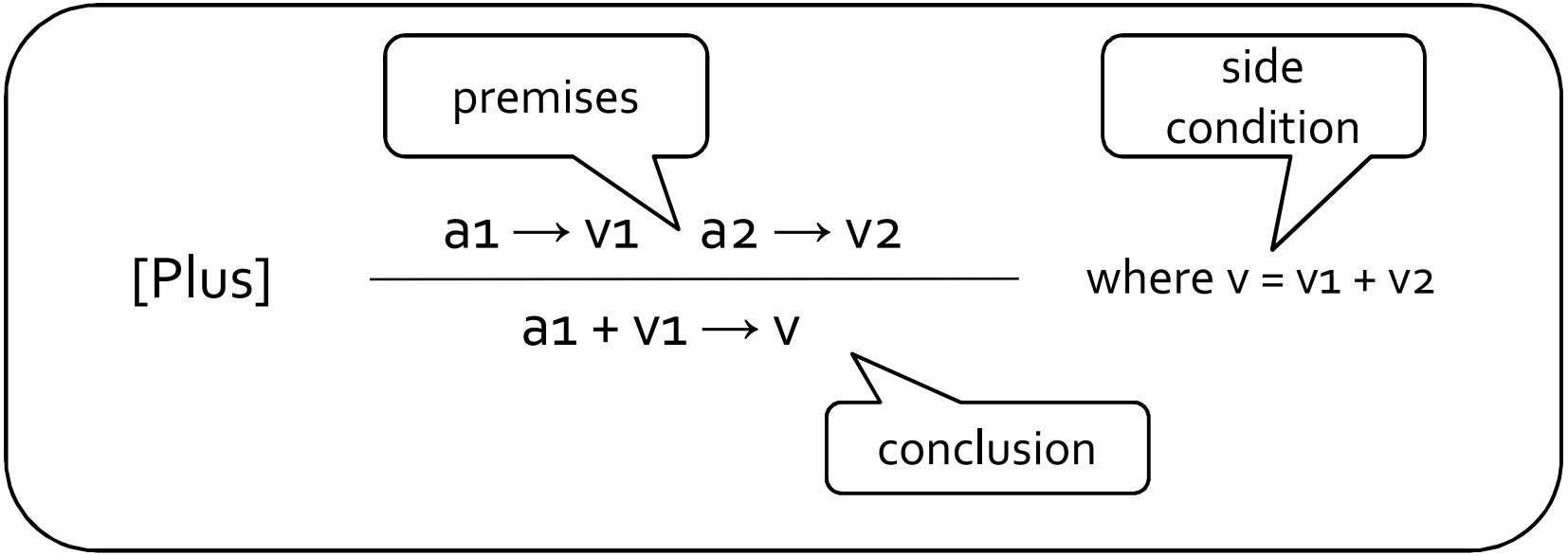
- We write $\sigma \to \sigma'$ when $(\sigma, \sigma') \in \to$

- $\to^*$ denotes the reflexive transitive closure of the relation $\to$
  - $\sigma \to^* \sigma'$ when there is a sequence $\sigma = \sigma_0 \to \sigma_1 \to \ldots \sigma_n = \sigma'$ for some $n \geq 0$

Big-step vs. Small-step

- Big-step
  - $\sigma \rightarrow \sigma'$ describes the entire computation
  - σ' is always a terminal configuration
- Small-step
  - $\sigma \Rightarrow \sigma'$ describes a single step of a larger computation
  - σ' need not be a terminal configuration
- pros/cons to each
- big-step hard in the presence of concurrency

Simple Arithmetic Expressions
(big step semantics)



a ⟶ v means "expression **a** evaluates to the value **v**"

a ∈ AExp , v ∈ Z

Simple Arithmetic Expressions
(big step semantics)

$$\text{[Plus]}\quad \frac{a1 \longrightarrow v1 \quad a2 \longrightarrow v2}{a1 + v1 \longrightarrow v} \quad \text{where } v = v1 + v2$$

$$\text{[Minus]}\quad \frac{a1 \longrightarrow v1 \quad a2 \longrightarrow v2}{a1 - v1 \longrightarrow v} \quad \text{where } v = v1 - v2$$

$$\text{[Mult]}\quad \frac{a1 \longrightarrow v1 \quad a2 \longrightarrow v2}{a1 * v1 \longrightarrow v} \quad \text{where } v = v1 * v2$$

$$\text{[Paren]}\quad \frac{a1 \longrightarrow v1}{(a1) \longrightarrow v}$$

$$\text{[Num]}\quad n \longrightarrow v \text{ if } N[\![n]\!] = v$$

Simple Arithmetic Expressions
(big step semantics)

- Transition system $(\Sigma, \rightarrow, T)$
  - configurations $\Sigma = AExp \cup Z$
  - transitions $\rightarrow \subseteq \Sigma \times \Sigma$: defined by the rules on the previous slide
  - final configurations $T = Z$

- Transitions are syntax directed

Derivation Tree

- show that (2+4)*(4+3) ⟶ 42

$$2 \longrightarrow 2 \quad 4 \longrightarrow 4 \qquad\qquad 4 \longrightarrow 4 \quad 3 \longrightarrow 3$$

$$\frac{2 \longrightarrow 2 \quad 4 \longrightarrow 4}{2 + 4 \longrightarrow 6} \qquad\qquad \frac{4 \longrightarrow 4 \quad 3 \longrightarrow 3}{4 + 3 \longrightarrow 7}$$

$$\frac{2 + 4 \longrightarrow 6}{(2 + 4) \longrightarrow 6} \qquad\qquad \frac{4 + 3 \longrightarrow 7}{(4 + 3) \longrightarrow 7}$$

$$\frac{(2+4) \longrightarrow 6 \quad (4 + 3) \longrightarrow 7}{(2+4)*(4 + 3) \longrightarrow 42}$$

Derivation Tree

$$\dfrac{\dfrac{\dfrac{2 \longrightarrow 2 \quad 4 \longrightarrow 4}{2 + 4 \longrightarrow 6}}{(2 + 4) \longrightarrow 6} \quad \dfrac{\dfrac{4 \longrightarrow 4 \quad 3 \longrightarrow 3}{4 + 3 \longrightarrow 7}}{(4 + 3) \longrightarrow 7}}{(2+4)*(4 + 3) \longrightarrow 42}$$

Simple Arithmetic Expressions
(small step semantics)

$$\text{[Plus-1]} \quad \frac{a1 \Rightarrow a1'}{a1 + a2 \Rightarrow a1' + a2}$$

$$\text{[Plus-2]} \quad \frac{a2 \Rightarrow a2'}{a1 + a2 \Rightarrow a1 + a2'}$$

$$\text{[Plus-3]} \quad v1 + v2 \Rightarrow v \text{ where } v = v1 + v2$$

- intermediate values
- intermediate configurations

Determinacy

- We would like the big-step semantics of arithmetic expressions to be deterministic
  - $a \longrightarrow v1$ and $a \longrightarrow v2$ then $v1 = v2$

- induction on the height of the derivation tree ("transition induction")
  - show that rules for roots are deterministic
  - show that transition rules are deterministic

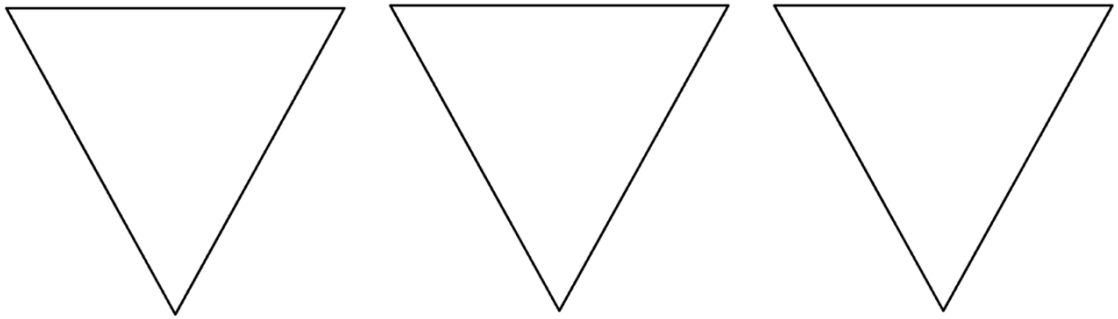
Determinacy

- Is the small-step semantics of arithmetic expressions deterministic?
- we want
  - if a $\Rightarrow$ v1 and a $\Rightarrow$ v2 then v1 = v2

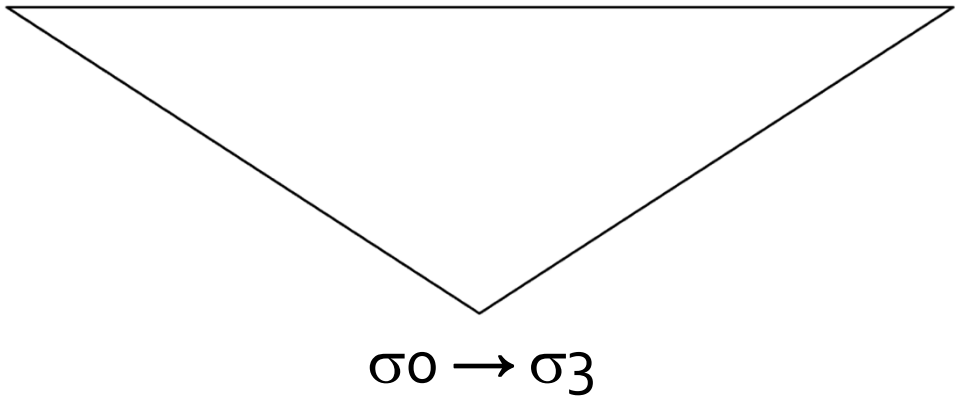
- but we have, for example
  - 2 +3 $\Rightarrow$ 2 + **3**
  - 2 + 3 $\Rightarrow$ **2** + 3

Small Step and Big Step

small step



big step



The WHILE Language: Syntax

Var	set of variables	$A \in AExp$	arithmetic expressions
Lab	set of labels	$B \in BExp$	boolean expressions
Op_a	arithmetic operators	$S \in Stmt$	statements
Op_b	boolean operators		
Op_r	relational operators		

$a ::= x \mid n \mid a1\ op_a\ a2$

$b ::= true \mid false \mid not\ b \mid b1\ op_b\ b2 \mid a1\ op_r\ a2$

$S ::= [x := a]^{lab}$
$\mid [skip]^{lab}$
$\mid S1;S2$
$\mid if\ [b]^{lab}\ then\ S1\ else\ S2$
$\mid while\ [b]^{lab}\ do\ S$

(We are going to abuse syntax later for readability)

The WHILE Language: Structural Operational Semantics

- $\sigma \in \text{State} = \text{Var} \rightharpoonup Z$
- Configuration:
  - $\langle S, \sigma \rangle$
  - σ for terminal configuration
- Transitions:
  - $\langle S, \sigma \rangle \Rightarrow \langle S', \sigma' \rangle$
  - $\langle S, \sigma \rangle \Rightarrow \sigma'$

Both the statement that remains to be executed, and the state, can change

The WHILE Language: Structural Operational Semantics

- Transition system $(\Sigma, \rightarrow, T)$
  - configurations
    $\Sigma = (\text{Stmt} \times \text{State}) \cup \text{State}$
  - transitions $\rightarrow \subseteq \Sigma \times \Sigma$
  - final configurations $T = \text{State}$

Arithmetic Expressions

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

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

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

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

Boolean Expressions

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

$B[\![not\ b]\!]\sigma = \neg B[\![b]\!]\sigma$

$B[\![b_1\ op_b\ b_2]\!]\sigma = B[\![b_1]\!]\sigma\ op_b\ B[\![b_2]\!]\sigma$

$B[\![a_1\ op_r\ a_2]\!]\sigma = A[\![a_1]\!]\sigma\ op_r\ A[\![a_2]\!]\sigma$

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] $$\frac{\langle S_1, \sigma \rangle \Rightarrow \sigma'}{\langle S_1; S_2, \sigma \rangle \Rightarrow \langle S_2, \sigma' \rangle}$$

(Table 2.6 from PPA)

The WHILE Language: Structural Operational Semantics

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

(Table 2.6 from PPA2)

Derivation Sequences

- Finite derivation sequence
  - A sequence $<S_o, \sigma_o> \ldots \sigma_n$
  - $<S_i, \sigma_i> \Rightarrow <S_{i+1}, \sigma_{i+1}>$
  - σ_n terminal configuration

- Infinite derivation sequence
  - A sequence $<S_o, \sigma_o> \ldots$
  - $<S_i, \sigma_i> \Rightarrow <S_{i+1}, \sigma_{i+1}>$

Termination in small-step semantics

while $[o = o]^1$ (
$[skip]^2$;
)

$< \text{while } [o = o]^1 ([skip]^2), \sigma > \Rightarrow$

$< [skip]^2; \text{while } [o = o]^1 ([skip]^2), \sigma > \Rightarrow$

$< \text{while } [o = o]^1 ([skip]^2), \sigma > \Rightarrow$

$< [skip]^2; \text{while } [o = o]^1 ([skip]^2), \sigma > \Rightarrow \ldots$

Termination in small-step semantics

- We say that S terminates from a start state σ when there exists a state σ' such that $\langle S,\sigma\rangle \Rightarrow^* \sigma'$

Termination in big-step semantics

while [o = o]^1 (
[skip]^2;
)

- what would be the transition in the big-step semantics for this example?

Semantic Equivalence

- formal semantics enables us to reason about programs and their equivalence
- S1 and S2 are semantically equivalent when
  - for all σ and σ' $\langle S1,\sigma\rangle \Rightarrow^* \sigma'$ iff $\langle S2,\sigma\rangle \Rightarrow^* \sigma'$
- We write S1 ~ S2 when S1 and S2 are semantically equivalent

Abnormal Termination

- add a statement **abort** for aborting execution
- in the big-step semantics
  - while (o=o) skip; ~ abort
- big-step semantics does not distinguish between abnormal termination and infinite-loops
- in the small-step semantics
  - while (o=o) skip; ~ abort
- but we can distinguish the cases if we look at the transitions
  - $\langle abort,\sigma\rangle \Rightarrow_o \langle abort,\sigma\rangle$
  - infinite trace of skips

Nondeterminism

big-step semantics

- new language construct s1 OR s2

[OR1-BSS] $$\frac{<S_1, \sigma> \rightarrow \sigma'}{<S_1 \text{ OR } S_2, \sigma> \rightarrow \sigma'}$$

[OR2-BSS] $$\frac{<S_2, \sigma> \rightarrow \sigma'}{<S_1 \text{ OR } S_2, \sigma> \rightarrow \sigma'}$$

Nondeterminism

small-step semantics

[OR1-SSS] $\langle S_1 \text{ OR } S_2, \sigma \rangle \Rightarrow \langle S1, \sigma \rangle$

[OR1-SSS] $\langle S_1 \text{ OR } S_2, \sigma \rangle \Rightarrow \langle S2, \sigma \rangle$

Nondeterminism

- (x = 1) OR while(0=0) skip;

- big-step semantics suppresses infinite loops
- small step semantics has the infinite sequence created by picking the while

<(x = 1) OR while(0=0) skip;,σ> $\Rightarrow$ <while(0=0) skip;,σ> $\Rightarrow$ …

What is the "meaning" of this program?

```
[y := x]1;
[z := 1]2;
while [y > 0]3 (
 [z := z * y]4;
 [y := y – 1]5;
)
[y := 0]6
```

now we can answer this question using derivation sequences

Example of Derivation Sequence

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

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

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

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

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

Traces

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

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

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

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

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

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

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

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

Traces

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

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

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

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

$< 1, \{ x \mapsto 42, y \mapsto 0, z \mapsto 0 \} > \Rightarrow$ $[y := x]^1$

$< 2, \{ x \mapsto 42, y \mapsto 42, z \mapsto 0 \} > \Rightarrow$ $[z := 1]^2$

$< 3, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \Rightarrow$ $[y > 0]^3$

$< 4, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \ldots$

Traces

$$\langle 1, \{ x \mapsto 42, y \mapsto 0, z \mapsto 0 \} \rangle \xRightarrow{[y := x]^1}$$

$$\langle 2, \{ x \mapsto 42, y \mapsto 42, z \mapsto 0 \} \rangle \xRightarrow{[z := 1]^2}$$

$$\langle 3, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} \rangle \xRightarrow{[y > 0]^3}$$

$$\langle 4, \{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} \rangle \ldots$$

Trace Semantics

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

$[y := x]^1$ $[z := 1]^2$

$< 1,\{ x \mapsto 42, y \mapsto 0, z \mapsto 0 \} > \Rightarrow < 2,\{ x \mapsto 42, y \mapsto 42, z \mapsto 0 \} > \Rightarrow$

$[y > 0]^3$

$< 3,\{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \Rightarrow < 4,\{ x \mapsto 42, y \mapsto 42, z \mapsto 1 \} > \ldots$

$[y := x]^1$ $[z := 1]^2$

$< 1,\{ x \mapsto 73, y \mapsto 0, z \mapsto 0 \} > \Rightarrow < 2,\{ x \mapsto 73, y \mapsto 73, z \mapsto 0 \} > \Rightarrow$

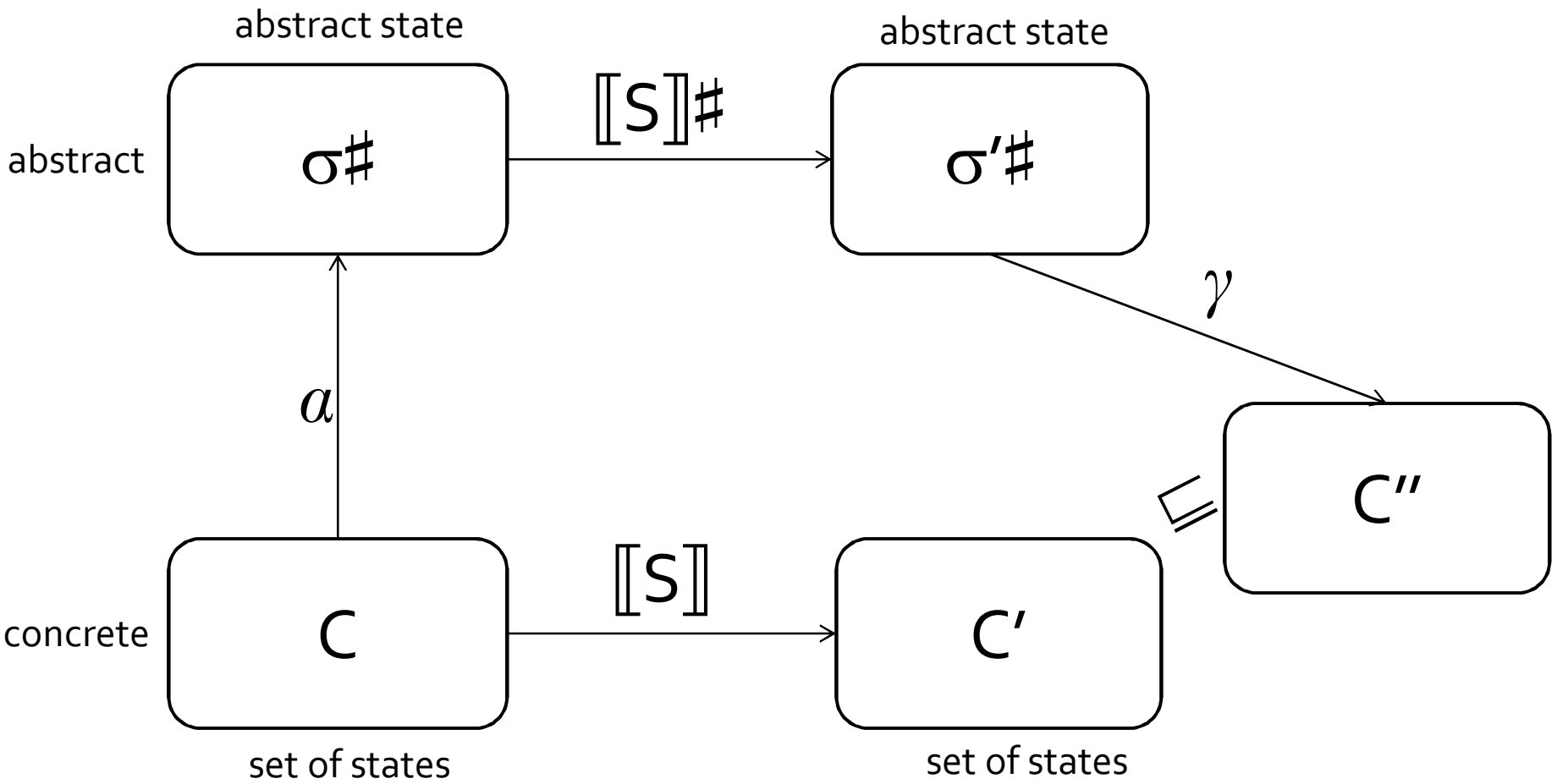
$[y > 0]^3$

$< 3,\{ x \mapsto 73, y \mapsto 73, z \mapsto 1 \} > \Rightarrow < 4,\{ x \mapsto 73, y \mapsto 73, z \mapsto 1 \} > \ldots$

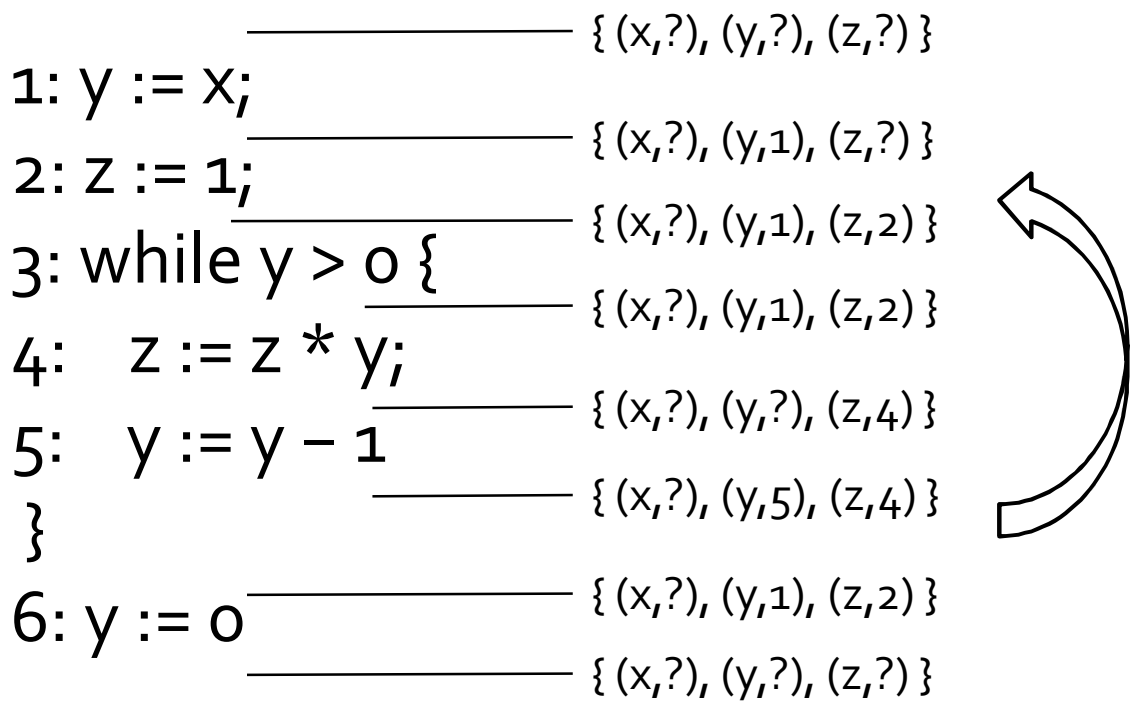
⋮

- In the beginning, there was the trace semantics…
- note that input (x) can be anything
- clearly, the trace semantics is not computable

Abstract Interpretation



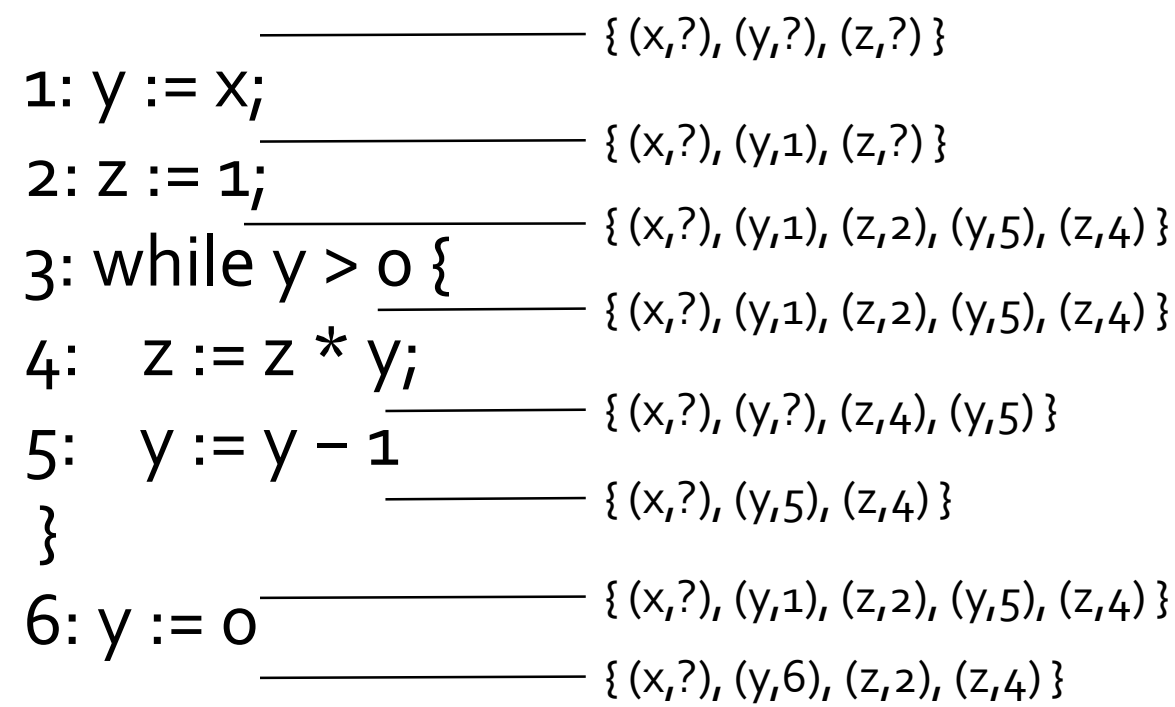
Dataflow Analysis



- 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**

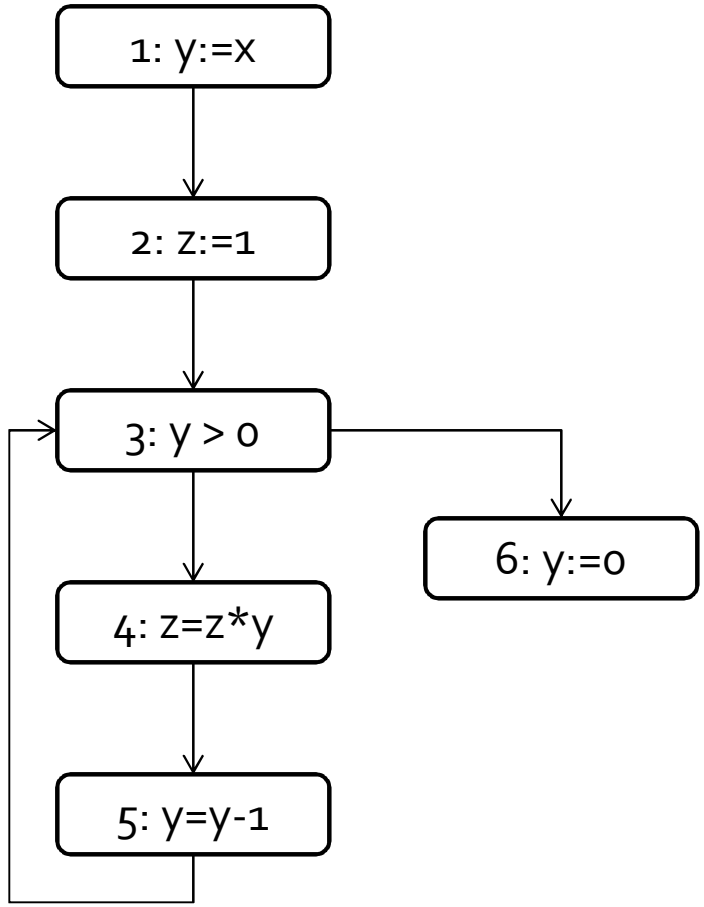
(adapted from Nielson, Nielson & Hankin)

Dataflow Analysis

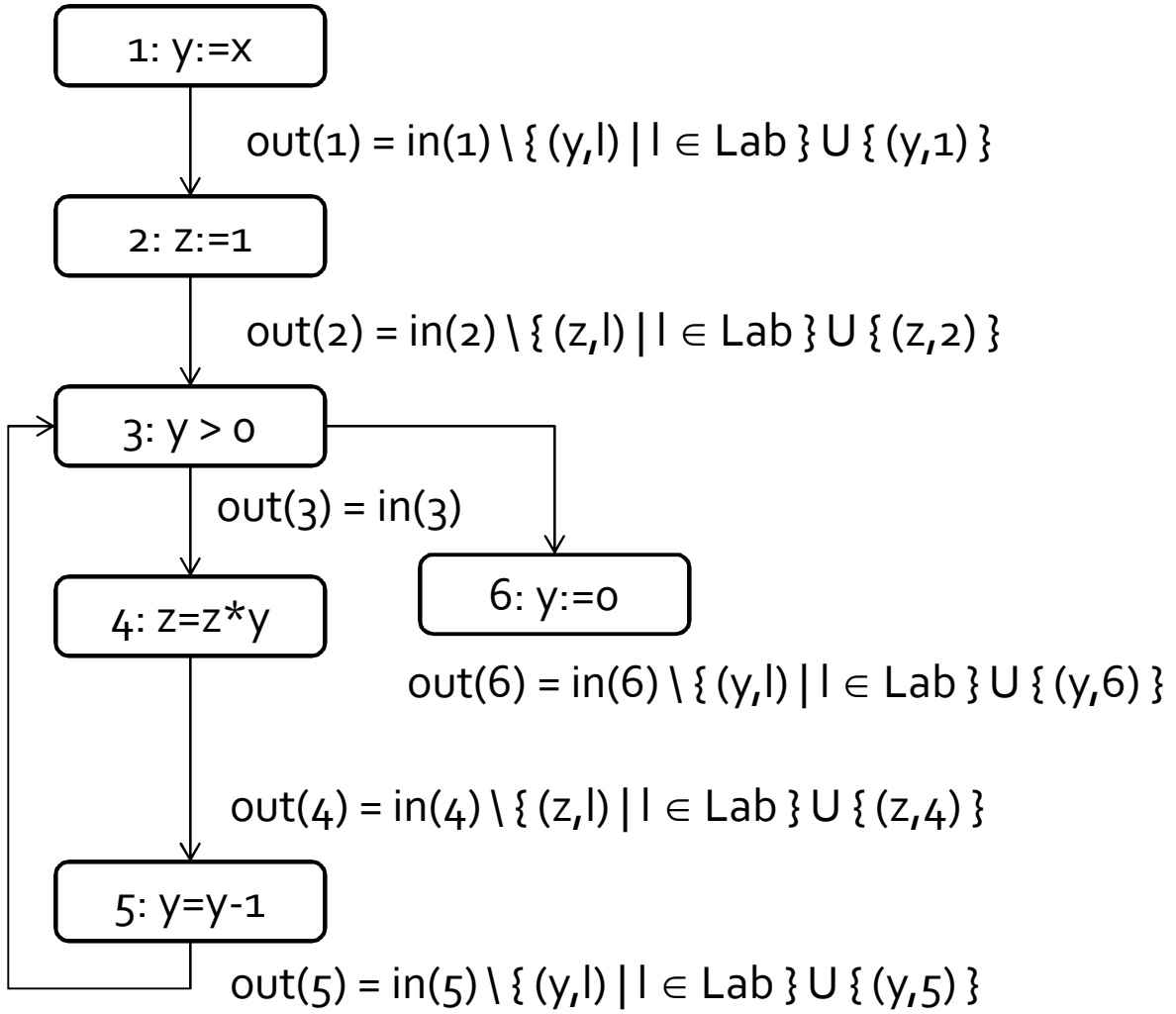
- Build control-flow graph
- Assign transfer functions
- Compute fixed point

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

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

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

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}

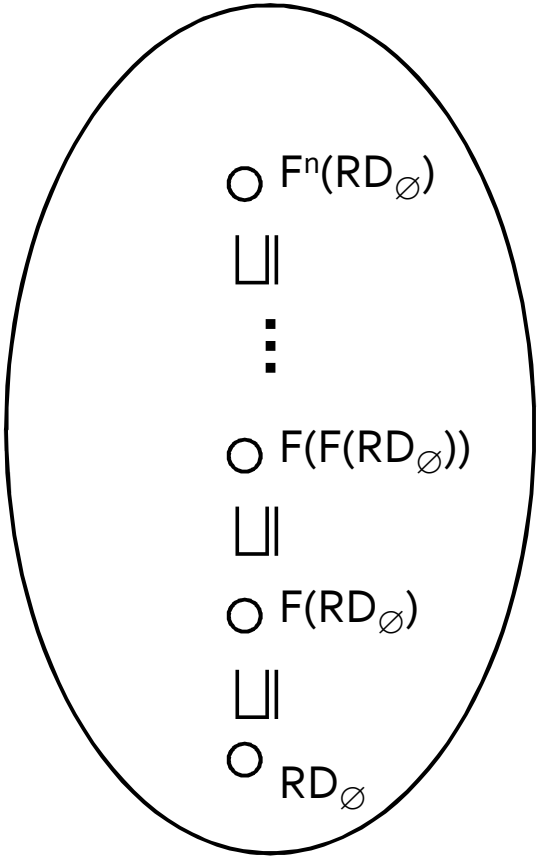
RD $\sqsubseteq$ RD′ when $\forall$i: RD(i) $\subseteq$ RD′(i)

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?

References

- "Transitions and Trees" / Huttel
- "Principles of Program Analysis" / Nielson, Nielson, and Hankin