Program Analysis and Verification

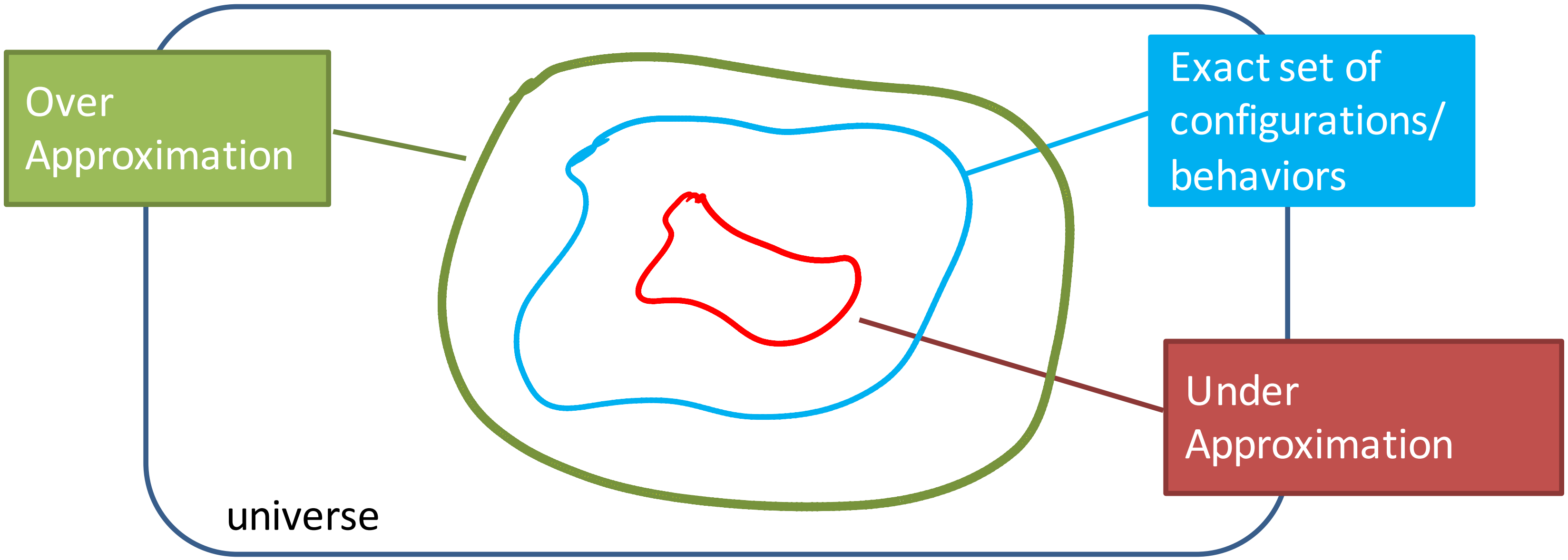
0368-4479

Noam Rinetzky

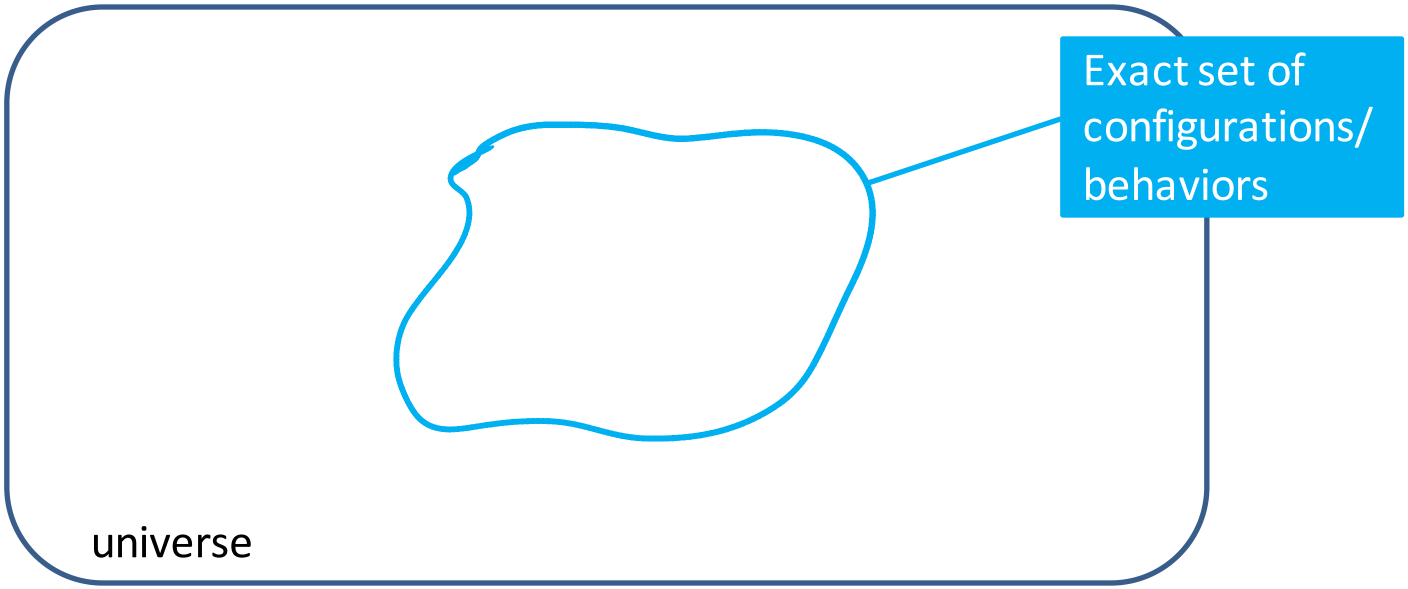
Lecture 2: Operational Semantics

Slides credit: Tom Ball, Dawson Engler, Roman Manevich, Erik Poll, Mooly Sagiv, Jean Souyris, Eran Tromer, Avishai Wool, Eran Yahav

Verification by over-approximation



Program semantics



Program analysis & verification

```
y = ?; x = ?;
x = y * 2
if (x % 2 == 0) {
   y = 42;
}  else {
   y  = 73;
   foo();
}
assert (y == 42);
```

?

What does P do?

```
y = ?; x = ?;
x = y * 2
if (x % 2 == 0) {
   y = 42;
}  else {
   y  = 73;
   foo();
}
assert (y == 42);
```

?

What does P mean?

```
y = ?; x = ?;
x = y * 2
if (x % 2 == 0) {
  y = 42;
} else {
  y  = 73;
  foo();
}
assert (y == 42);
```

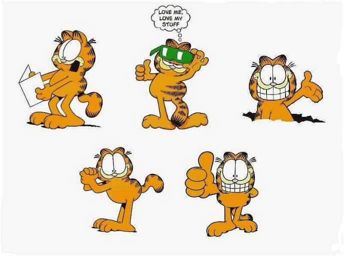
= ...

syntax

semantics

Program semantics

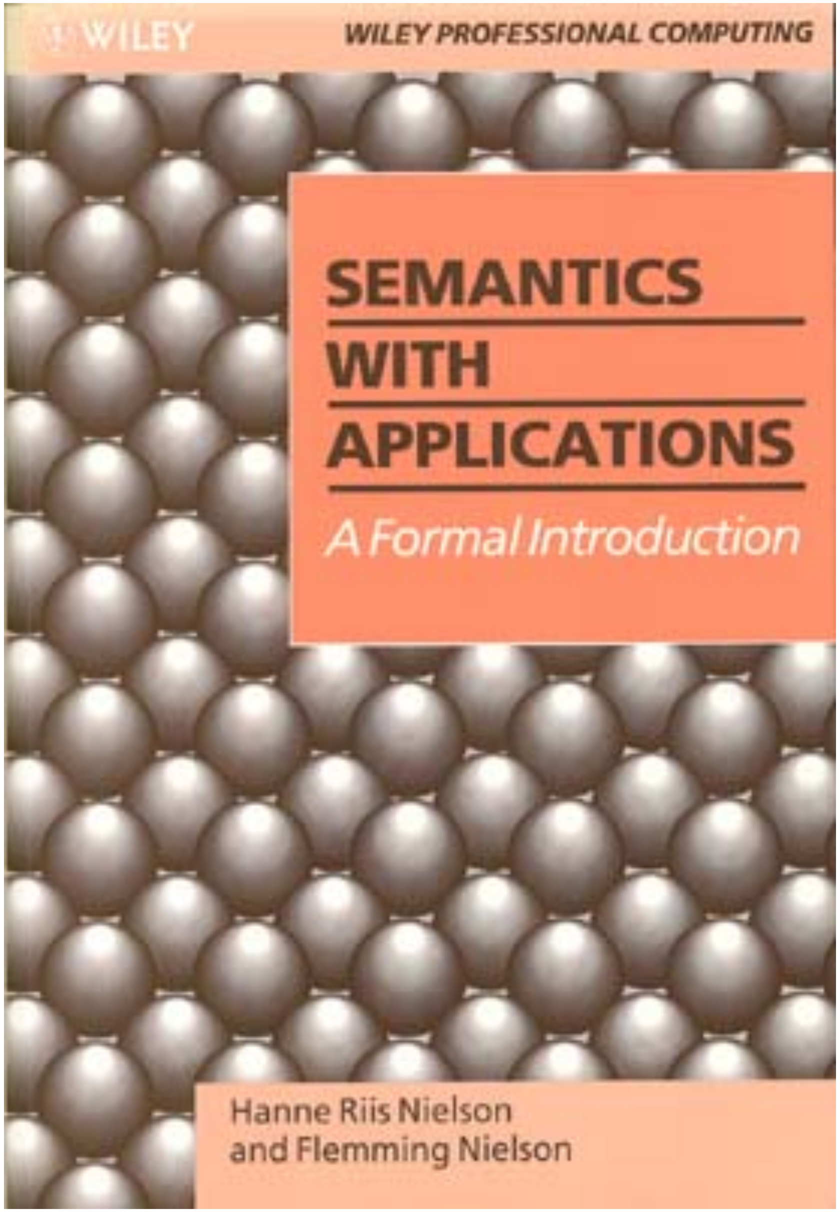
- State-transformer
  - Set-of-states transformer
  - Trace transformer
- Predicate-transformer
- Functions
- Cat-transformer



Program semantics & verification

Operational Semantics

http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html



A simple imperative language: **While**

Abstract syntax:

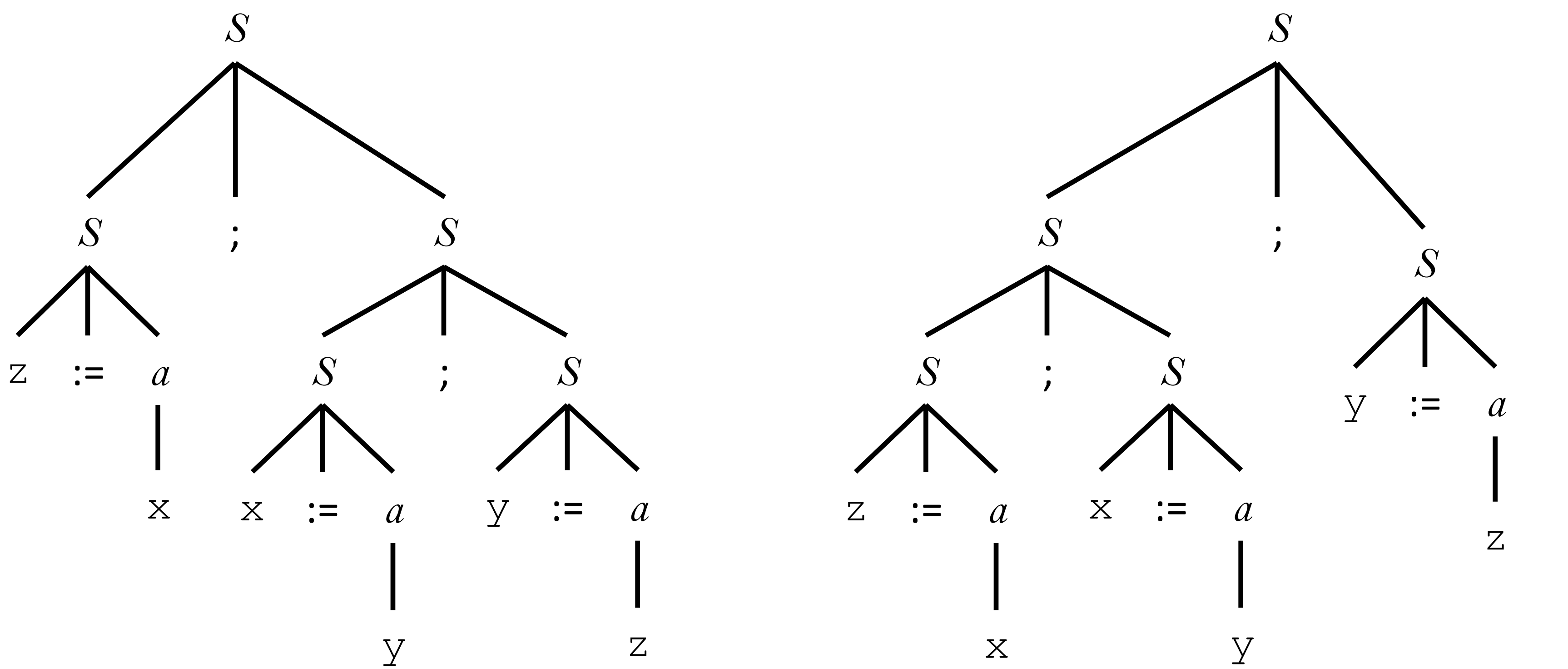
$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

$$b ::= \texttt{true} \mid \texttt{false}$$
$$\mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$

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

Concrete syntax vs. abstract syntax

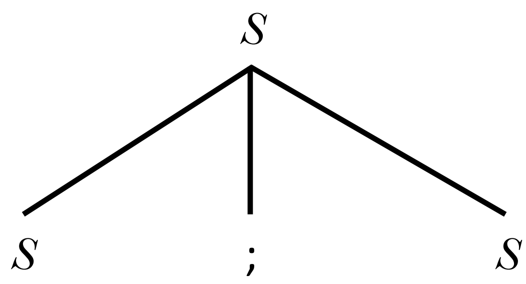
`z:=x; x:=y; y:=z`



`z:=x; (x:=y; y:=z)`   **`(z:=x; x:=y); y:=z`**

Exercise: draw an AST

y:=1; while ¬(x=1) do (y:=y*x; x:=x-1)



Syntactic categories

$n \in$ **Num**	numerals
$x \in$ **Var**	program variables
$a \in$ **Aexp**	arithmetic expressions
$b \in$ **Bexp**	boolean expressions
$S \in$ **Stm**	statements

Semantic categories

Z	Integers {0, 1, -1, 2, -2, …}
T	Truth values {**ff**, **tt**}
State	**Var** → **Z**

Example state:	s=[x↦5, y↦7, z↦0]
Lookup:	s x = 5
Update:	s[x↦6] = [x↦6, y↦7, z↦0]

Example state manipulations

- $[x \mapsto 1, y \mapsto 7, z \mapsto 16]\ y$ =
- $[x \mapsto 1, y \mapsto 7, z \mapsto 16]\ t$ =
- $[x \mapsto 1, y \mapsto 7, z \mapsto 16][x \mapsto 5]$ =
- $[x \mapsto 1, y \mapsto 7, z \mapsto 16][x \mapsto 5]\ x$ =
- $[x \mapsto 1, y \mapsto 7, z \mapsto 16][x \mapsto 5]\ y$ =

Semantics of arithmetic expressions

- Arithmetic expressions are side-effect free
- Semantic function $\mathcal{A}[\![\,\textbf{Aexp}\,]\!] : \textbf{State} \rightarrow \textbf{Z}$
- Defined by induction on the syntax tree

$\mathcal{A}[\![\, n \,]\!]\, s = n$

$\mathcal{A}[\![\, x \,]\!]\, s = s\, x$

$\mathcal{A}[\![\, a_1 + a_2 \,]\!]\, s = \mathcal{A}[\![\, a_1 \,]\!]\, s + \mathcal{A}[\![\, a_2 \,]\!]\, s$

$\mathcal{A}[\![\, a_1 - a_2 \,]\!]\, s = \mathcal{A}[\![\, a_1 \,]\!]\, s - \mathcal{A}[\![\, a_2 \,]\!]\, s$

$\mathcal{A}[\![\, a_1 * a_2 \,]\!]\, s = \mathcal{A}[\![\, a_1 \,]\!]\, s \times \mathcal{A}[\![\, a_2 \,]\!]\, s$

$\mathcal{A}[\![\, (a_1) \,]\!]\, s = \mathcal{A}[\![\, a_1 \,]\!]\, s$ --- not needed

$\mathcal{A}[\![\, - a \,]\!]\, s = 0 - \mathcal{A}[\![\, a_1 \,]\!]\, s$

- Compositional
- Properties can be proved by structural induction

Arithmetic expression exercise

Suppose s x = 3

Evaluate $\mathcal{A}\,[\![\mathbf{x+1}]\!]$ s

Semantics of boolean expressions

- Boolean expressions are side-effect free
- Semantic function $\mathcal{B}[\![\textbf{Bexp}]\!] : \textbf{State} \to \textbf{T}$
- Defined by induction on the syntax tree

$\mathcal{B}[\![\text{true}]\!] s = tt$

$\mathcal{B}[\![\text{false}]\!] s = ff$

$\mathcal{B}[\![a_1 = a_2]\!] s =$

$\mathcal{B}[\![a_1 \leq a_2]\!] s =$

$\mathcal{B}[\![b_1 \wedge b_2]\!] s =$

$\mathcal{B}[\![\neg b]\!] s =$

Operational semantics

- Concerned with **how** to execute programs
  - How statements modify **state**
  - Define transition relation between configurations
- Two flavors
  - Natural semantics: describes how the **overall** results of executions are obtained
    - So-called “big-step” semantics
  - Structural operational semantics: describes how the **individual steps** of a computations take place
    - So-called “small-step” semantics

Natural operating semantics (NS)

Natural operating semantics (NS)

- aka "Large-step semantics"

$$\langle S, s \rangle \longrightarrow s'$$

all steps

Natural operating semantics

- Developed by Gilles Kahn [STACS 1987]
- Configurations

$\langle S, s\rangle$	Statement *S* is about to execute on state *s*
s	Terminal (final) state

- Transitions

$\langle S, s\rangle \rightarrow s'$	Execution of *S* from *s* will terminate with the result state *s*'

  – Ignores non-terminating computations

Natural operating semantics

- $\longrightarrow$ defined by rules of the form

premise

side condition

$$\frac{\langle S_1, s_1\rangle \longrightarrow s_1', \ \ldots\ , \langle S_n, s_n\rangle \longrightarrow s_n'}{\langle S, s\rangle \longrightarrow s'} \quad \text{if}\ldots$$

conclusion

- The meaning of compound statements is defined using the meaning immediate constituent statements

Natural semantics for **While**

[ass$_{ns}$] $\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$

[skip$_{ns}$] $\langle \texttt{skip}, s\rangle \to s$

axioms

[comp$_{ns}$] $$\frac{\langle S_1, s\rangle \to s', \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''}$$

[if$^{tt}_{ns}$] $$\frac{\langle S_1, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to s'}$$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$

[if$^{ff}_{ns}$] $$\frac{\langle S_2, s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to s'}$$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$

Natural semantics for **While**

[while$^{ff}_{ns}$] $\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \longrightarrow s$ if $\mathcal{B}[\![b]\!]\,s = \mathbf{ff}$

Non-compositional

[while$^{tt}_{ns}$] $$\frac{\langle S, s\rangle \rightarrow s',\ \langle \texttt{while}\ b\ \texttt{do}\ S, s'\rangle \longrightarrow s''}{\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \longrightarrow s''}$$ if $\mathcal{B}[\![b]\!]\,s = \mathbf{tt}$

Example

- Let s_0 be the state which assigns zero to all program variables

$$\langle \texttt{x:=x+1}, s_0 \rangle \longrightarrow s_0[\mathsf{x} \mapsto 1]$$

$$\langle \texttt{skip}, s_0 \rangle \longrightarrow s_0$$

$$\frac{\langle \texttt{skip}, s_0 \rangle \longrightarrow s_0, \ \langle \texttt{x:=x+1}, s_0 \rangle \longrightarrow s_0[\mathsf{x} \mapsto 1]}{\langle \texttt{skip; x:=x+1}, s_0 \rangle \longrightarrow s_0[\mathsf{x} \mapsto 1]}$$

$$\frac{\langle \texttt{x:=x+1}, s_0 \rangle \longrightarrow s_0[\mathsf{x} \mapsto 1]}{\langle \texttt{if x=0 then x:=x+1 else skip}, s_0 \rangle \longrightarrow s_0[\mathsf{x} \mapsto 1]}$$

Derivation trees

- Using axioms and rules to derive a transition $\langle S, s \rangle \rightarrow s'$ gives a derivation tree
  - Root: $\langle S, s \rangle \rightarrow s'$
  - Leaves: axioms
  - Internal nodes: conclusions of rules
    - Immediate children: matching rule premises

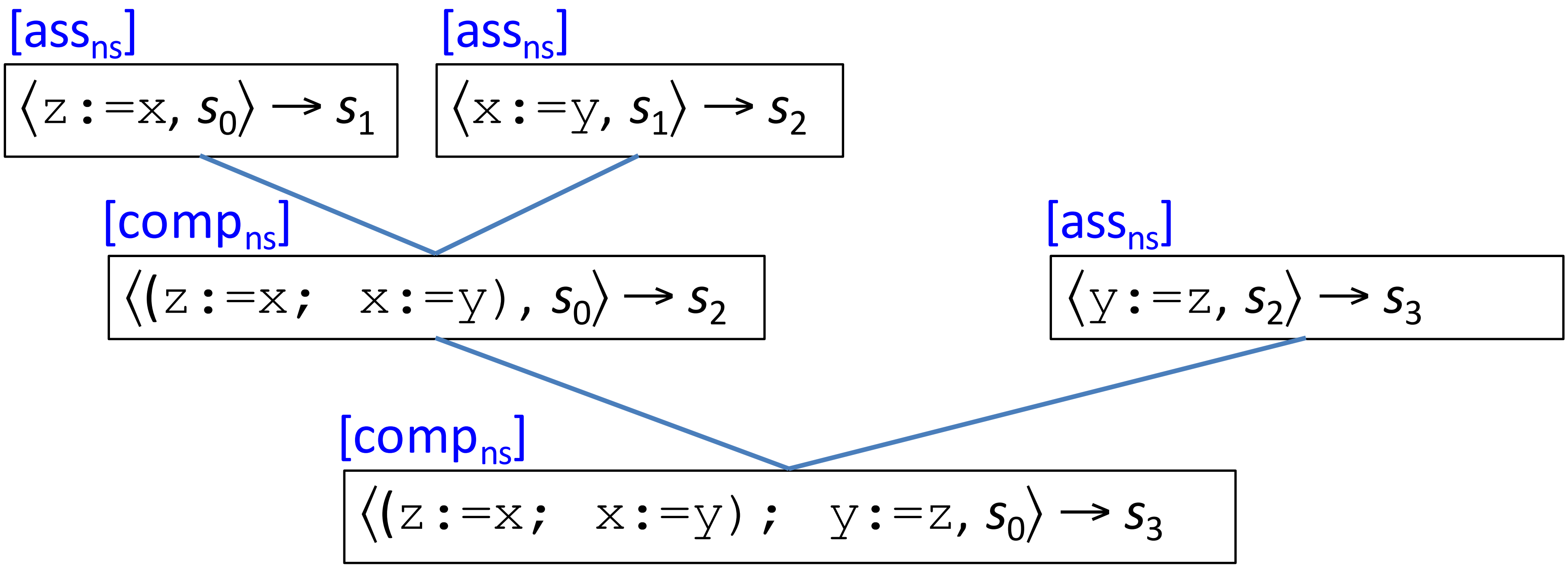
Derivation tree example 1

- Assume $s_0=[x\mapsto 5,\ y\mapsto 7,\ z\mapsto 0]$
  $s_1=[x\mapsto 5,\ y\mapsto 7,\ z\mapsto 5]$
  $s_2=[x\mapsto 7,\ y\mapsto 7,\ z\mapsto 5]$
  $s_3=[x\mapsto 7,\ y\mapsto 5,\ z\mapsto 5]$

$$\dfrac{\dfrac{\overset{[\text{ass}_{\text{ns}}]}{\langle \texttt{z:=x},\ s_0\rangle \to s_1} \qquad \overset{[\text{ass}_{\text{ns}}]}{\langle \texttt{x:=y},\ s_1\rangle \to s_2}}{\overset{[\text{comp}_{\text{ns}}]}{\langle (\texttt{z:=x;}\ \ \texttt{x:=y}),\ s_0\rangle \to s_2}} \qquad \overset{[\text{ass}_{\text{ns}}]}{\langle \texttt{y:=z},\ s_2\rangle \to s_3}}{\overset{[\text{comp}_{\text{ns}}]}{\langle (\texttt{z:=x;}\ \ \texttt{x:=y});\ \ \texttt{y:=z},\ s_0\rangle \to s_3}}$$

Derivation tree example 1

- Assume $s_0=[x\mapsto5, y\mapsto7, z\mapsto0]$
  $s_1=[x\mapsto5, y\mapsto7, z\mapsto5]$
  $s_2=[x\mapsto7, y\mapsto7, z\mapsto5]$
  $s_3=[x\mapsto7, y\mapsto5, z\mapsto5]$



Top-down evaluation via derivation trees

- Given a statement S and an input state s find an output state s' such that $\langle S, s\rangle \to s'$
- Start with the root and repeatedly apply rules until the axioms are reached
  - Inspect different alternatives in order
- In While s' and the derivation tree is unique

Top-down evaluation example

- Factorial program with s `x` = 2
- Shorthand: W=`while ¬(x=1) do (y:=y*x; x:=x-1)`

[ass$_{ns}$]

$\langle$`y:=y*x`$, s[y \mapsto 1]\rangle \rightarrow s[y \mapsto 2]$

[ass$_{ns}$]

$\langle$`x:=x-1`$, s[y \mapsto 2]\rangle \rightarrow s[y \mapsto 2][x \mapsto 1]$

[comp$_{ns}$]

$\langle$`y:=y*x; x:=x-1`$, s[y \mapsto 1]\rangle \rightarrow s[y \mapsto 2][x \mapsto 1]$

[while$^{ff}{}_{ns}$]

$\langle W, s[y \mapsto 2][x \mapsto 1]\rangle \rightarrow s[y \mapsto 2, x \mapsto 1]$

[ass$_{ns}$]

$\langle$`y:=1`$, s\rangle \rightarrow s[y \mapsto 1]$

[while$^{tt}{}_{ns}$]

$\langle W, s[y \mapsto 1]\rangle \rightarrow s[y \mapsto 2, x \mapsto 1]$

[comp$_{ns}$]

$\langle$`y:=1; while ¬(x=1) do (y:=y*x; x:=x-1)`$, s\rangle \rightarrow s[y \mapsto 2][x \mapsto 1]$

Program termination

- Given a statement *S* and input *s*
  - *S* terminates on s if there exists a state *s*' such that $\langle S, s \rangle \longrightarrow$ s'
  - S loops on s if there is no state *s*' such that $\langle S, s \rangle \longrightarrow$ s'
- Given a statement *S*
  - *S* always terminates if for every input state *s*, *S* terminates on *s*
  - *S* always loops if for every input state *s*, *S* loops on *s*

Semantic equivalence

- S_1 and S_2 are semantically equivalent if for all s and s'
  $\langle S_1, s\rangle \rightarrow s'$ if and only if $\langle S_2, s\rangle \rightarrow s'$
- Simple example
  `while` ***b*** `do` ***S***
  is semantically equivalent to:
  `if` ***b*** `then` (***S***; `while` ***b*** `do` ***S***) `else skip`
  – Read proof in pages 26-27

Properties of natural semantics

- Equivalence of program constructs
  - **`skip; skip`** is semantically equivalent to **`skip`**
  - $((S_1; S_2); S_3)$ is semantically equivalent to $(S_1; (S_2; S_3))$
  - **`(x:=5; y:=x*8)`** is semantically equivalent to **`(x:=5; y:=40)`**

Equivalence of $(S_1; S_2); S_3$ and $S_1; (S_2; S_3)$

Equivalence of $(S_1; S_2); S_3$ and $S_1; (S_2; S_3)$

Assume $\langle (S_1; S_2); S_3, s\rangle \longrightarrow s'$ then the following unique derivation tree exists:

$$\dfrac{\dfrac{\langle S_1, s\rangle \longrightarrow s_1, \ \langle S_2, s_1\rangle \longrightarrow s_{12}}{\langle (S_1; S_2), s\rangle \longrightarrow s_{12},} \qquad \langle S_3, s_{12}\rangle \longrightarrow s'}{\langle (S_1; S_2); S_3, s\rangle \longrightarrow s'}$$

Using the rule applications above, we can construct the following derivation tree:

$$\dfrac{\langle S_1, s\rangle \longrightarrow s_1, \qquad \dfrac{\langle S_2, s_1\rangle \longrightarrow s_{12}, \ \langle S_3, s_{12}\rangle \longrightarrow s'}{\langle (S_2; S_3), s_{12}\rangle \longrightarrow s'}}{\langle S_1; (S_2; S_3, s\rangle \longrightarrow s'}$$

And vice versa.

Deterministic semantics for **While**

- **Theorem:** for all statements S and states s_1, s_2
  if $\langle S, s\rangle \rightarrow s_1$ and $\langle S, s\rangle \rightarrow s_2$ then s_1=s_2
- The proof uses induction on the shape of derivation trees (pages 29-30)
  - Prove that the property holds for all simple derivation trees by showing it holds for axioms
  - Prove that the property holds for all composite trees:
    - For each rule assume that the property holds for its premises (induction hypothesis) and prove it holds for the conclusion of the rule

single node

#nodes>1

The semantic function S_{ns}

- The meaning of a statement S is defined as a partial function from **State** to **State**

  S_{ns}: **Stm** $\rightarrow$ (**State** $\hookrightarrow$ **State**)

$$S_{ns}[\![S]\!]\,s = \begin{cases} s' & \text{if } \langle S, s\rangle \rightarrow s' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- Examples:

  $S_{ns}[\![\texttt{skip}]\!]s = s$

  $S_{ns}[\![\texttt{x:=1}]\!]s = s\,[x \mapsto 1]$

  $S_{ns}[\![\texttt{while true do skip}]\!]s = \text{undefined}$

Structural operating semantics (SOS)

Structural operating semantics (SOS)

- aka “Small-step semantics”

$$\langle S, s \rangle \Rightarrow \langle S', s' \rangle$$

first step

Structural operational semantics

- Developed by Gordon Plotkin
- Configurations: γ has one of two forms:

$\langle S, s\rangle$	Statement S is about to execute on state s
s	Terminal (final) state

first step

- Transitions $\langle S, s\rangle \Rightarrow \gamma$

$\gamma = \langle S', s'\rangle$ Execution of S from s is **not** completed and remaining computation proceeds from intermediate configuration γ

$\gamma = s'$ Execution of S from s has **terminated** and the final state is s'

- $\langle S, s\rangle$ is stuck if there is no γ such that $\langle S, s\rangle \Rightarrow \gamma$

Structural semantics for **While**

[ass$_{sos}$] $\langle x{:=}a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$

[skip$_{sos}$] $\langle \texttt{skip},\ s\rangle \Rightarrow s$

[comp$^1_{sos}$] $$\frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;\ S_2,\ s\rangle \Rightarrow \langle S_1';\ S_2,\ s'\rangle}$$

[comp$^2_{sos}$] $$\frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$$

When does this happen?

[if$^{tt}_{sos}$] $\langle \texttt{if}\ \ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle$ if $\mathcal{B}[\![b]\!]\,s = \mathbf{tt}$

[if$^{ff}_{sos}$] $\langle \texttt{if}\ \ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle$ if $\mathcal{B}[\![b]\!]\,s = \mathbf{ff}$

Structural semantics for **While**

[while$_{sos}$]

$$\langle \texttt{while } b \texttt{ do } S, s\rangle \Rightarrow \langle \texttt{if } b \texttt{ then } S; \texttt{while } b \texttt{ do } S) \texttt{ else } \texttt{skip}, s\rangle$$

Derivation sequences

- A derivation sequence of a statement S starting in state s is either
- A **finite** sequence $\gamma_0, \gamma_1, \gamma_2 \ldots, \gamma_k$ such that
  1. $\gamma_0 = \langle S, s \rangle$
  2. $\gamma_i \Rightarrow \gamma_{i+1}$
  3. γ_k is either stuck configuration or a final state
- An **infinite** sequence $\gamma_0, \gamma_1, \gamma_2, \ldots$ such that
  1. $\gamma_0 = \langle S, s \rangle$
  2. $\gamma_i \Rightarrow \gamma_{i+1}$
- Notations:
  - $\gamma_0 \Rightarrow^k \gamma_k$ γ_0 derives γ_k in k steps
  - $\gamma_0 \Rightarrow^* \gamma$ γ_0 derives γ in a finite number of steps
- For **each** step there is a corresponding derivation tree

Derivation sequence example

- Assume s_0=[x↦5, y↦7, z↦0]

$$\langle \texttt{(z:=x; x:=y); y:=z}, s_0 \rangle$$
$$\Rightarrow \langle \texttt{x:=y; y:=z}, s_0[z\mapsto 5] \rangle$$
$$\Rightarrow \langle \texttt{y:=z}, (s_0[z\mapsto 5])[x\mapsto 7] \rangle$$
$$\Rightarrow ((s_0[z\mapsto 5])[x\mapsto 7])[y\mapsto 5]$$

- Derivation tree for first step:

$$\dfrac{\dfrac{\langle \texttt{z:=x}, s_0 \rangle \Rightarrow s_0[z\mapsto 5]}{\langle \texttt{z:=x; x:=y}, s_0 \rangle \Rightarrow \langle \texttt{x:=y}, s_0[z\mapsto 5] \rangle}}{\langle \texttt{(z:=x; x:=y); y:=z}, s_0 \rangle \Rightarrow \langle \texttt{x:=y; y:=z}, s_0[z\mapsto 5] \rangle}$$

Evaluation via derivation sequences

- For any **While** statement S and state s it is always possible to find at least one derivation sequence from $\langle S, s\rangle$
  – Apply axioms and rules forever or until a terminal or stuck configuration is reached
- **Proposition:** there are no stuck configurations in **While**

Factorial (*n!)* example

- Input state s such that s x = 3

```
y := 1; while ¬(x=1) do (y := y * x; x := x - 1)
```

$\langle$y :=1 ; W, s$\rangle$

$\Rightarrow \langle$W, s[y$\mapsto$1]$\rangle$

$\Rightarrow \langle$if ¬(x =1) then ((y := y * x; x := x – 1); W else skip), s[y$\mapsto$1]$\rangle$

$\Rightarrow \langle$((y := y * x; x := x – 1); W), s[y$\mapsto$1]$\rangle$

$\Rightarrow \langle$(x := x – 1; W), s[y$\mapsto$3]$\rangle$

$\Rightarrow \langle$W , s[y$\mapsto$3][x$\mapsto$2]$\rangle$

$\Rightarrow \langle$if ¬(x =1) then ((y := y * x; x := x – 1); W else skip), s[y$\mapsto$3][x$\mapsto$2]$\rangle$

$\Rightarrow \langle$((y := y * x; x := x – 1); W), s[y$\mapsto$3] [x$\mapsto$2]$\rangle$

$\Rightarrow \langle$(x := x – 1; W) , s[y$\mapsto$6] [x$\mapsto$2]$\rangle$

$\Rightarrow \langle$W, s[y$\mapsto$6][x$\mapsto$1]$\rangle$

$\Rightarrow \langle$if ¬(x =1) then ((y := y * x; x := x – 1); W else skip, s[y$\mapsto$6][x$\mapsto$1]$\rangle$

$\Rightarrow \langle$skip, s[y$\mapsto$6][x$\mapsto$1]$\rangle$

$\Rightarrow$ s[y$\mapsto$6][x$\mapsto$1]

Program termination

- Given a statement S and input s
  - S terminates on s if there exists a finite derivation sequence starting at $\langle S, s\rangle$
  - S terminates successfully on s if there exists a finite derivation sequence starting at $\langle S, s\rangle$ leading to a final state
  - S loops on s if there exists an infinite derivation sequence starting at $\langle S, s\rangle$

Properties of structural operational semantics

- S_1 and S_2 are semantically equivalent if:
  - for all s and γ which is either final or stuck, $\langle S_1, s\rangle \Rightarrow^* \gamma$ if and only if $\langle S_2, s\rangle \Rightarrow^* \gamma$
  - for all s, there is an infinite derivation sequence starting at $\langle S_1, s\rangle$ if and only if there is an infinite derivation sequence starting at $\langle S_2, s\rangle$
- **Theorem: While** is deterministic:
  - If $\langle S, s\rangle \Rightarrow^* s_1$ and $\langle S, s\rangle \Rightarrow^* s_2$ then $s_1 = s_2$

Sequential composition

- **Lemma:** If $\langle S_1; S_2, s\rangle \Rightarrow^k s''$ then there exists s' and k=m+n such that $\langle S_1, s\rangle \Rightarrow^m s'$ and $\langle S_2, s'\rangle \Rightarrow^n s''$
- The proof (pages 37-38) uses induction on the length of derivation sequences
  - Prove that the property holds for all derivation sequences of length 0
  - Prove that the property holds for all other derivation sequences:
    - Show that the property holds for sequences of length k+1 using the fact it holds on all sequences of length k (induction hypothesis)

The semantic function S_{sos}

- The meaning of a statement S is defined as a partial function from **State** to **State**

$$S_{sos}\colon \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$S_{sos}\,[\![S]\!]\,s = \begin{cases} s' & \text{if } \langle S, s\rangle \Rightarrow^* s' \\ \text{undefined} & \text{else} \end{cases}$$

- Examples:

$S_{sos}\,[\![\texttt{skip}]\!]\,s = s$

$S_{sos}\,[\![\texttt{x:=1}]\!]\,s = s\,[x \mapsto 1]$

$S_{sos}\,[\![\texttt{while true do skip}]\!]\,s = \text{undefined}$

An equivalence result

- For every statement in **While**

$$S_{ns} [\![S]\!] = S_{sos} [\![S]\!]$$

- Proof in pages 40-43

Language Extensions

- `abort` statement (like C's exit w/o return value)
- Non-determinism
- Parallelism
- Local Variables
- Procedures
  - Static Scope
  - Dynamic scope

While + `abort`

- Abstract syntax

$$S ::= x \texttt{ := } a \mid \texttt{skip} \mid S_1 ;\ S_2$$
$$\mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2$$
$$\mid \texttt{while } b \texttt{ do } S$$
$$\mid \texttt{abort}$$

- Abort terminates the execution
  - In "**`skip;`** *S*" the statement *S* executes
  - In "**`abort;`** *S*" the statement *S* should never execute
- Natural semantics rules: …?
- Structural semantics rules: …?

Comparing semantics

Statement	Natural semantics	Structural semantics
`abort`		
`abort;` *S*		
`skip;` *S*		
`while true do skip`		
`if x = 0 then abort else y := y + x`		

Conclusions

- The natural semantics cannot distinguish between looping and abnormal termination
  - Unless we add a special error state
- In the structural operational semantics looping is reflected by infinite derivations and abnormal termination is reflected by stuck configuration

While + non-determinism

- Abstract syntax

$$S ::= x \ \texttt{:=}\ a \mid \texttt{skip} \mid S_1; S_2$$
$$\mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2$$
$$\mid \texttt{while}\ b\ \texttt{do}\ S$$
$$\mid S_1\ \texttt{or}\ S_2$$

- Either S_1 is executed or S_2 is executed
- Example: `x:=1 or (x:=2; x:=x+2)`
  - Possible outcomes for `x`: 1 and 4

While + non-determinism: natural semantics

[or$^1_{ns}$] $$\frac{\langle S_1, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'}$$

[or$^2_{ns}$] $$\frac{\langle S_2, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'}$$

While + non-determinism: structural semantics

[or^1_{sos}] ?

[or^2_{sos}] ?

While + non-determinism

- What about the definitions of the semantic functions?
  - $S_{\text{ns}} [\![S_1 \texttt{ or } S_2]\!] s$
  - $S_{\text{sos}} [\![S_1 \texttt{ or } S_2]\!] s$

Comparing semantics

Statement	Natural semantics	Structural semantics
`x:=1 or (x:=2; x:=x+2)`		
`(while true do skip) or (x:=2; x:=x+2)`		

Conclusions

- In the natural semantics non-determinism will suppress non-termination (looping) if possible
- In the structural operational semantics non-determinism does not suppress non-terminating statements

While + parallelism

Abstract syntax

$$S ::= x := a \mid \textbf{skip} \mid S_1; S_2$$
$$\mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2$$
$$\mid \textbf{while}\ b\ \textbf{do}\ S$$
$$\mid S_1 \parallel S_2$$

- All the interleaving of S_1 and S_2 are executed
- Example: `x:=1 ‖ (x:=2; x:=x+2)`
  – Possible outcomes for `x`: 1, 3, 4

While + parallelism: structural semantics

[par$^{1}_{sos}$] $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 \| S_2, s\rangle \Rightarrow \langle S_1' \| S_2, s'\rangle}$$

[par$^{2}_{sos}$] $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \| S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

[par$^{3}_{sos}$] $$\frac{\langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle}{\langle S_1 \| S_2, s\rangle \Rightarrow \langle S_1 \| S_2', s'\rangle}$$

[par$^{4}_{sos}$] $$\frac{\langle S_2, s\rangle \Rightarrow s'}{\langle S_1 \| S_2, s\rangle \Rightarrow \langle S_1, s'\rangle}$$

While + parallelism: natural semantics

Challenge problem:
Give a formal proof that this is in fact impossible.

Idea: try to prove on a restricted version of **While** without loops/conditions

Example: derivation sequences of a parallel statement

$\langle \texttt{x:=1} \parallel \texttt{(x:=2; x:=x+2)}, s\rangle \Rightarrow$

Conclusion

- In the structural operational semantics we concentrate on small steps so interleaving of computations can be easily expressed

- In the natural semantics immediate constituent is an atomic entity so we cannot express interleaving of computations

While + memory

Abstract syntax

$$S ::= x := a \mid \texttt{skip} \mid S_1 ; S_2$$
$$\mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2$$
$$\mid \texttt{while}\ b\ \texttt{do}\ S$$
$$\mid x := \texttt{malloc}(a)$$
$$\mid x := [y]$$
$$\mid [x] := y$$

~~**State : Var** → Z~~

State : Stack × **Heap**

Stack : Var → Z

Heap : Z → Z

Integers as memory addresses

From states to traces

Trace semantics

- Low-level (conceptual) semantics
- Add program counter (pc) with states
  - Σ = **State** + pc
- The meaning of a program is a relation

$$\tau \subseteq \Sigma \times \mathbf{Stm} \times \Sigma$$

- Execution is a finite/infinite sequence of states
- A useful concept in defining static analysis as we will see later

Example

```
1: y := 1;
 while 2: ¬(x=1) do (
     3: y := y * x;
     4: x := x - 1
)
5:
```

Traces

```
1: y := 1;
 while 2: ¬(x=1) do (
      3: y := y * x;
      4: x := x - 1
)
5:
```

Set of traces is infinite therefore trace semantics is incomputable in general

⟨{x↦2,y↦3},1⟩ [**y:=1**]⟨{x↦2,y↦1},2⟩ [¬ **(x=1)**] ⟨{x↦2,y↦1},⟩3 [**y:=y*x**] ⟨{x↦2,y↦2},⟩4 [**x:=x-1**] ⟨{x↦1,y↦2}, ⟩2[¬ **(x=1)**] ⟨{x↦1,y↦2},⟩5

⟨{x↦3,y↦3},1⟩ [**y:=1**]⟨{x↦3,y↦1},2⟩ [¬ **(x=1)**] ⟨{x↦3,y↦1},⟩3 [**y:=y*x**] ⟨{x↦3,y↦3},⟩4 [**x:=x-1**] ⟨{x↦2,y↦3}, ⟩2[¬ **(x=1)**] ⟨{x↦2,y↦3},⟩3 [**y:=y*x**] ⟨{x↦2,y↦6},⟩4 [**x:=x-1**] ⟨{x↦1,y↦6}, ⟩2[¬ **(x=1)**] ⟨{x↦1,y↦6},⟩5

…

Operational semantics summary

- SOS is powerful enough to describe imperative programs
  - Can define the set of traces
  - Can represent program counter implicitly
  - Handle `goto` statements and other non-trivial control constructs (e.g., exceptions)
- Natural operational semantics is an abstraction
- Different semantics may be used to justify different behaviors
- Thinking in concrete semantics is essential for a analysis writer

The End