Concepts of Programming Languages – Recitation 3:

Natural and Structural Operational Semantics

Oded Padon

Reference:

Semantics with Applications by H. Nielson and F. Nielson – Ch. 2

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

Operational Semantics

- Formal definition of program semantics
- The meaning of the program is described "operationally"
- Natural Operational Semantics - NOS
- Structural Operational Semantics - SOS

- We can go **systematically** from operational semantics (NOS or SOS) to an **interpreter**

The **While** Programming Language

- Abstract syntax
  S::= x := a | **skip** | S_1 ; S_2 | **if** b **then** S_1 **else** S_2 |
  while b do S
- Use parenthesizes for precedence
- Informal Semantics
  - **skip** behaves like no-operation
  - Import meaning of arithmetic and Boolean operations

Semantics of arithmetic expressions

- Assume that arithmetic expressions are side-effect free
- A⟦ Aexp ⟧ : State $\rightarrow$ N
- Defined by structural induction on the syntax tree

$$
\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \star \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}
$$

Semantics of Boolean expressions

- Assume that Boolean expressions are side-effect free
- B⟦ Bexp ⟧ : State $\rightarrow$ T
- Defined by induction on the syntax tree:

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

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

$$\mathcal{B}[\![a_1 = a_2]\!]s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases}$$

$$\mathcal{B}[\![a_1 \leq a_2]\!]s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases}$$

$$\mathcal{B}[\![\neg\ b]\!]s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases}$$

$$\mathcal{B}[\![b_1 \wedge b_2]\!]s = \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}$$

Natural Operational Semantics

- Notations:
  - S – program construct (word in the While language)
  - s, s' – states (functions Var $\rightarrow$ N)
- <S, s> $\rightarrow$ s' means:
  If S is executed on state s, it terminates and the state after execution is s'
- Describe the "overall" effect of program constructs
- Ignores non terminating computations

Formally defining $\rightarrow$

- $\rightarrow$ is defined **inductively** using **inference rules**, with both **syntactic** conditions on S and **semantic** conditions on s

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

$$[\text{skip}_{\text{ns}}] \quad \langle \texttt{skip}, s\rangle \rightarrow s$$

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

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

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

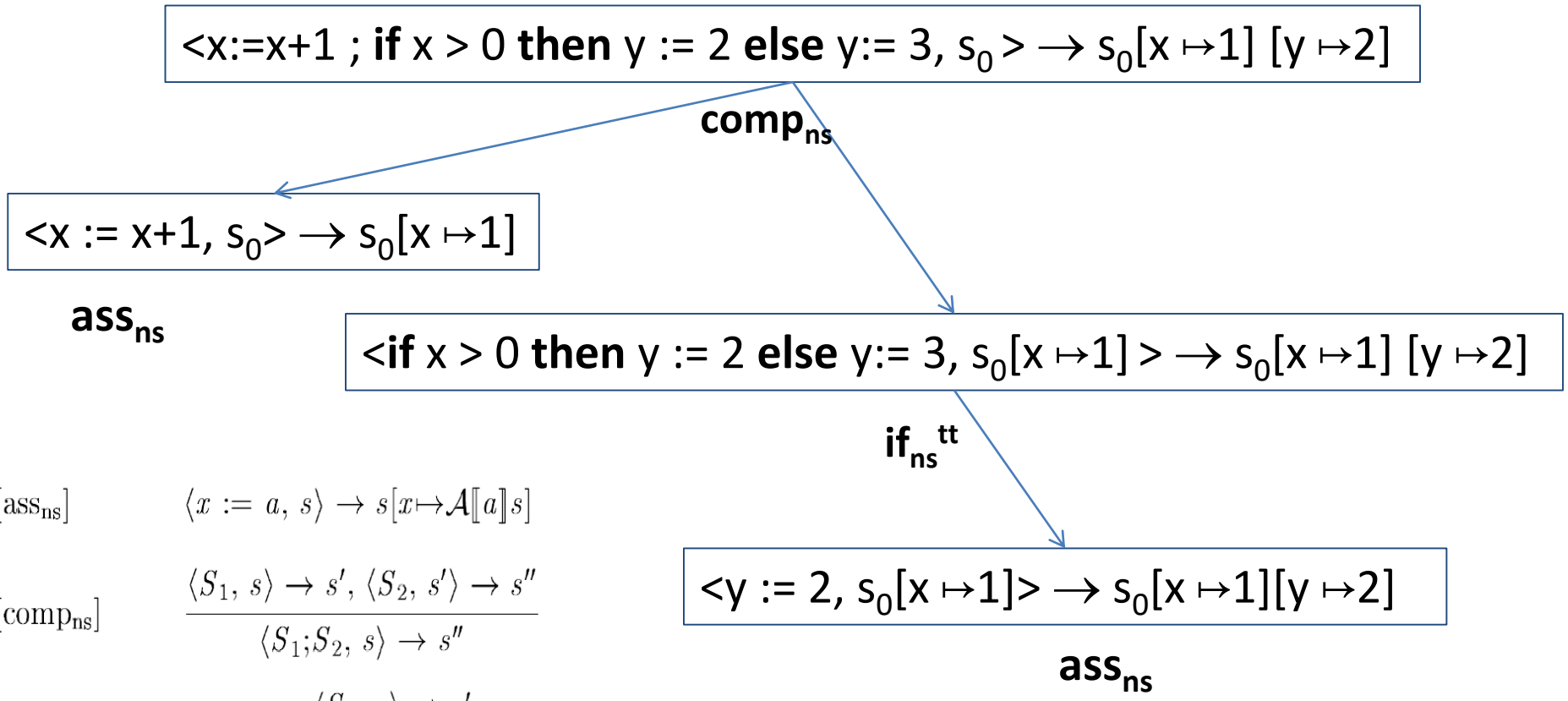
$$[\text{while}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \rightarrow s', \langle \texttt{while } b \texttt{ do } S, s'\rangle \rightarrow s''}{\langle \texttt{while } b \texttt{ do } S, s\rangle \rightarrow s''} \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

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

Derivation Trees

- A derivation tree is a way to write applications of inference rules
- A derivation tree is a "proof" that <S, s> $\rightarrow$s'
- The root of tree is <S, s> $\rightarrow$s'
- Each node is a conclusion from its children using an inference rule
- Leaves are instances of axioms (rules with no premises)
- Non-leaves are instances of inference rules with premises
  - Immediate children match rule premises
  - The semantic condition is satisfied

Example Derivation Tree



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

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

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

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

Alternative Notation

$$\textbf{ass}_{ns}\ \frac{}{\langle y := 2,\ s_0[x \mapsto 1]\rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2]}$$

$$\textbf{ass}_{ns}\ \frac{}{\langle x := x+1,\ s_0\rangle \rightarrow s_0[x \mapsto 1]} \qquad \frac{}{\langle \textbf{if } x > 0 \textbf{ then } y := 2 \textbf{ else } y := 3,\ s_0[x \mapsto 1]\rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2]}\ \textbf{if}_{ns}^{tt}$$

$$\textbf{comp}_{ns}\ \frac{}{\langle x := x+1 ;\ \textbf{if } x > 0 \textbf{ then } y := 2 \textbf{ else } y := 3,\ s_0\rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2]}$$

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

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

$$[\mathrm{if}_{\mathrm{ns}}^{\mathrm{tt}}] \qquad \frac{\langle S_1,\ s\rangle \rightarrow s'}{\langle \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\mathrm{if}_{\mathrm{ns}}^{\mathrm{ff}}] \qquad \frac{\langle S_2,\ s\rangle \rightarrow s'}{\langle \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

Semantic Equivalence

- Two statements S_1 and S_2 are **semantically equivalent** ($S_1 \approx S_2$) if for any two states s, s' we have:

$$\langle S_1, s\rangle \rightarrow s' \quad \textbf{iff} \quad \langle S_2, s\rangle \rightarrow s'$$

- Examples:
  - S ; skip ≈ S ?
  - skip ; S ≈ S ?
  - S1 ; S2 ≈ S2 ; S1 ?
  - ((S1 ; S2) ; S3) ≈ (S1 ;(S2 ; S3)) ?
  - x := 2 ; y := x + 3 ≈ y := 5; x := 2 ?
  - while b do S ≈ if b then (S ; while b do S) else skip ?
  - while x > 1 do skip ≈ while x > 2 do skip ?
  - while true do x:=x+1 ≈ while true do x:=x+2 ?

Proving Semantic Equivalence

- Two statements S_1 and S_2 are **semantically equivalent** ($S_1 \approx S_2$) if for any two states s, s' we have:

$$\langle S_1, s\rangle \rightarrow s' \quad \textbf{iff} \quad \langle S_2, s\rangle \rightarrow s'$$

- To prove semantic equivalence:
  1. Assume $\langle S_1, s\rangle \rightarrow s'$, show $\langle S_2, s\rangle \rightarrow s'$
  2. Assume $\langle S_2, s\rangle \rightarrow s'$, show $\langle S_1, s\rangle \rightarrow s'$

- The proof is usually by manipulating derivation trees

Example: S ; skip ≈ S

- To prove semantic equivalence:
  1. Assume <S, s> → s' , show <S;skip, s> → s'
  2. Assume <S;skip, s> → s' , show <S, s> → s'

1.

$$\frac{T}{\langle S, s\rangle \rightarrow s'} \quad \Rightarrow \quad \frac{\dfrac{T}{\langle S, s\rangle \rightarrow s'} \qquad \dfrac{}{\langle skip, s'\rangle \rightarrow s'}\ \textbf{skip}_{ns}}{\langle S;skip, s\rangle \rightarrow s'}\ \textbf{comp}_{ns}$$

Example: S ; skip ≈ S

- To prove semantic equivalence:
  1. Assume <S, s> → s' , show <S;skip, s> → s'
  2. Assume <S;skip, s> → s' , show <S, s> → s'

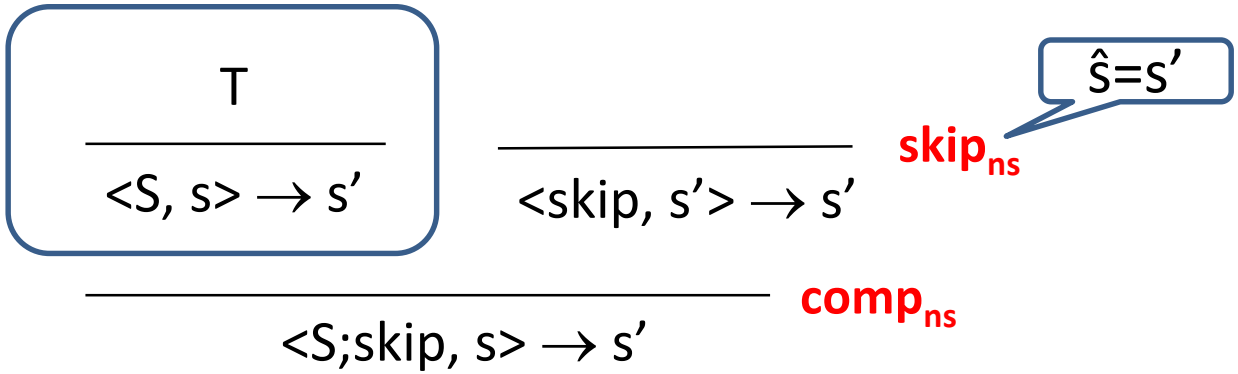
2.

$$\dfrac{\dfrac{T}{\langle S, s\rangle \to \hat{s}} \qquad \dfrac{}{\langle skip, \hat{s}\rangle \to s'}\ \textbf{skip}_{ns}}{\langle S;skip, s\rangle \to s'}\ \textbf{comp}_{ns}$$

Example: S ; skip ≈ S

- To prove semantic equivalence:
  1. Assume <S, s> → s' , show <S;skip, s> → s'
  2. Assume <S;skip, s> → s' , show <S, s> → s'

2.



Example: while loop

Prove: while b do S $\approx$ if b then (S ; while b do S) else skip

W

1. Assume <W, s> $\rightarrow$ s'' , show < if b then (S ; W) else skip, s> $\rightarrow$ s''
   Two cases:

$$\frac{\dfrac{T_1}{\langle S, s\rangle \rightarrow s'} \quad \dfrac{T_2}{\langle \text{while b do S}, s'\rangle \rightarrow s''}}{\langle \text{while b do S}, s\rangle \rightarrow s''}\ \textbf{while}_{ns}^{tt}$$

$\mathcal{B}$[b] s = **tt**

$$\frac{}{\langle \text{while b do S}, s\rangle \rightarrow s}\ \textbf{while}_{ns}^{ff}$$

$\mathcal{B}$[b] s = **ff**
s = s''

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

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

Case $\textbf{while}_{ns}^{ff}$

$$\frac{}{\langle \text{while b do S}, s\rangle \to s}\ \textbf{while}_{ns}^{ff} \qquad \mathcal{B}[\![b]\!]\, s = \textbf{ff},\ s = s''$$

$$\frac{}{\langle \text{skip}, s\rangle \to s}\ \textbf{skip}_{ns}$$

$$\frac{}{\langle \text{if b then (S;W) else skip}, s\rangle \to s}\ \textbf{if}_{ns}^{ff}$$

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

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

Case $\textbf{while}_{\textbf{ns}}^{\textbf{tt}}$

$$\frac{T_1 \qquad T_2}{\langle S, s\rangle \to s' \quad \langle \text{while b do S}, s'\rangle \to s''}\ \textbf{while}_{\textbf{ns}}^{\textbf{tt}}$$
$$\langle \text{while b do S}, s\rangle \to s''$$

$\mathcal{B}[\![b]\!]\, s = \textbf{tt}$

$$\frac{\dfrac{T_1 \qquad T_2}{\langle S, s\rangle \to s' \quad \langle W, s'\rangle \to s''}\ \textbf{comp}_{\textbf{ns}}}{\langle S;W, s\rangle \to s''}\ \textbf{if}_{\textbf{ns}}^{\textbf{tt}}$$
$$\langle \text{if b then (S;W) else skip}, s\rangle \to s''$$

Direction 2

2. Assume < if b then (S ; W) else skip, s> $\rightarrow$ s'', show <W, s> $\rightarrow$ s''
   Two cases:

$$\dfrac{\dfrac{\dfrac{T_3}{\langle S, s\rangle \rightarrow s'} \quad \dfrac{T_4}{\langle W, s'\rangle \rightarrow s''}}{\langle S;W, s\rangle \rightarrow s''}\ \mathbf{comp}_{ns}}{\langle \text{if b then (S;W) else skip}, s\rangle \rightarrow s''}\ \mathbf{if}_{ns}^{tt}$$

$\mathcal{B}$[b] s = **tt**

$$\dfrac{\dfrac{}{\langle \text{skip}, s\rangle \rightarrow s}\ \mathbf{skip}_{ns}}{\langle \text{if b then (S;W) else skip}, s\rangle \rightarrow s}\ \mathbf{if}_{ns}^{ff}$$

$\mathcal{B}$[b] s = **ff**

s = s''

Case $\mathbf{if}_{ns}^{ff}$

$$\frac{}{\langle skip, s\rangle \rightarrow s}\ \mathbf{skip}_{ns}$$

$$\frac{}{\langle \text{if b then (S;W) else skip}, s\rangle \rightarrow s}\ \mathbf{if}_{ns}^{ff}$$

$\mathcal{B}$[b] s = **ff**

s = s''

$$\frac{}{\langle \text{while b do S}, s\rangle \rightarrow s}\ \mathbf{while}_{ns}^{ff}$$

Case $\mathbf{if}_{ns}^{tt}$

$$\dfrac{\dfrac{\dfrac{T_3}{\langle S, s\rangle \to s'} \quad \dfrac{T_4}{\langle W, s'\rangle \to s''}}{\langle S;W, s\rangle \to s''}\ \mathbf{comp}_{ns}}{\langle \text{if } b \text{ then } (S;W) \text{ else skip}, s\rangle \to s''}\ \mathbf{if}_{ns}^{tt} \qquad \mathcal{B}[b]\, s = \mathbf{tt}$$

$$\dfrac{\dfrac{T_3}{\langle S, s\rangle \to s'} \quad \dfrac{T_4}{\langle \text{while } b \text{ do } S, s'\rangle \to s''}}{\langle \text{while } b \text{ do } S, s\rangle \to s''}\ \mathbf{while}_{ns}^{tt}$$

Adding Semantics of Conditions with Side Effects

Add semantics of:

$$\text{if } (x:=e) \text{ then } S_1 \text{ else } S_2$$

$$\textbf{if-ass}_{ns}^{tt} \quad \frac{\langle x:=e , s\rangle \rightarrow s' \qquad \langle S_1 , s'\rangle \rightarrow s''}{\langle \text{if } (x:=e) \text{ then } S_1 \text{ else } S_2, s\rangle \rightarrow s''} \qquad \text{if } \mathbf{A}[\![x]\!]s' \neq 0$$

$$\textbf{if-ass}_{ns}^{ff} \quad \frac{\langle x:=e , s\rangle \rightarrow s' \qquad \langle S_2 , s'\rangle \rightarrow s''}{\langle \text{if } (x:=e) \text{ then } S_1 \text{ else } S_2, s\rangle \rightarrow s''} \qquad \text{if } \mathbf{A}[\![x]\!]s' = 0$$

Adding Semantics of 3-Way Case

Add semantics of:

$$\text{case } (b_1: S_1),(b_2: S_2),(\text{else}: S_3)$$

$$\textbf{case}_{ns}^{1} \quad \frac{\langle S_1 , s\rangle \rightarrow s'}{\langle \text{case } (b_1: S_1),(b_2: S_2),(\text{else}: S_3) , s\rangle \rightarrow s'} \quad \text{if } \mathbf{B}[\![b_1]\!]s = \mathbf{tt}$$

$$\textbf{case}_{ns}^{2} \quad \frac{\langle S_2 , s\rangle \rightarrow s'}{\langle \text{case } (b_1: S_1),(b_2: S_2),(\text{else}: S_3) , s\rangle \rightarrow s'} \quad \text{if } \mathbf{B}[\![b_1]\!]s = \mathbf{ff} \text{ and } \mathbf{B}[\![b_2]\!]s = \mathbf{tt}$$

$$\textbf{case}_{ns}^{\text{else}} \quad \frac{\langle S_3 , s\rangle \rightarrow s'}{\langle \text{case } (b_1: S_1),(b_2: S_2),(\text{else}: S_3) , s\rangle \rightarrow s'} \quad \text{if } \mathbf{B}[\![b_1]\!]s = \mathbf{ff} \text{ and } \mathbf{B}[\![b_2]\!]s = \mathbf{ff}$$

Structural Operational Semantics

- Emphasizes the individual execution steps
- **<S, *s*>** $\Rightarrow \gamma$
  - the **"first"** step of executing **S** on state ***s*** leads to γ
- Two possibilities for γ
  - γ **= <S', s'>**
    The execution of S is not completed and, S' is the remaining computation to be performed on s'
  - γ **= s'**
    The execution of S has terminated with a final state s'
- γ is a **stuck** configuration when there are no transitions

Formally defining $\Rightarrow$

- $\Rightarrow$ is defined **inductively** using **inference rules**, with both **syntactic** conditions on S and **semantic** conditions on s

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

$$[\mathrm{skip}_{\mathrm{sos}}] \qquad \langle \mathtt{skip},\ s\rangle \Rightarrow s$$

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

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

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

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

$$[\mathrm{while}_{\mathrm{sos}}] \qquad \langle \mathtt{while}\ b\ \mathtt{do}\ S,\ s\rangle \Rightarrow \langle \mathtt{if}\ b\ \mathtt{then}\ (S;\ \mathtt{while}\ b\ \mathtt{do}\ S)\ \mathtt{else}\ \mathtt{skip},\ s\rangle$$

Examples for $\Rightarrow$

s_0: state which assigns zero to all variables

- $\langle y := 2, s_0[x \mapsto 1] \rangle \Rightarrow s_0[x \mapsto 1]\ [y \mapsto 2]$
- $\langle x := x+1, s_0 \rangle \Rightarrow s_0[x \mapsto 1]$
- $\langle x := x+2 \ ; \ x := x+3, s_0 \rangle \Rightarrow \langle x := x+3, s_0[x \mapsto 2] \rangle$
- $\langle$**if** $x > 0$ **then** $y := 2$ **else** $y := 3, s_0 \rangle \Rightarrow \langle y := 3, s_0 \rangle$
- $\langle$**if** $x > 0$ **then** $y := 2$ **else** $y := 3, s_0[x \mapsto 1] \rangle \Rightarrow \langle y := 2, s_0[x \mapsto 1] \rangle$
- $\langle$**while** $x < 5$ **do** $x := x+2, s_0 \rangle \Rightarrow$
  $\langle$**if** $x < 5$ **then** $(x := x+2$; **while** $x < 5$ **do** $x := x+2)$ **else** skip, $s_0 \rangle$
- $\langle$**while** $x < 0$ **do** $x := x+2, s_0 \rangle \Rightarrow$
  $\langle$**if** $x < 0$ **then** $(x := x+2$; **while** $x < 0$ **do** $x := x+2)$ **else** skip, $s_0 \rangle$

Finite Derivation Sequences

- Computation is modeled by **derivation sequences**
- A finite derivation sequence starting at <S, s>:
  - $\gamma_0, \gamma_1, \gamma_2 \ldots, \gamma_k$
  - $\gamma_0 =$ <S, s>
  - $\gamma_i \Rightarrow \gamma_{i+1}$
  - γ_k is either stuck configuration or a final state
- For each step there is a derivation tree
- $\gamma_0 \Rightarrow^k \gamma_k$ in k steps
- $\gamma_0 \Rightarrow^* \gamma$ in some finite number of steps
- Models terminating computation

Infinite Derivation Sequences

- An infinite derivation sequence starting at <S, s>:
  - $\gamma_0, \gamma_1, \gamma_2 \ldots$ such that
  - γ_0=<S, s>
  - $\gamma_i \Rightarrow \gamma_{i+1}$
- Example
  - S = while true do skip
  - $s=s_0$
- Models non-terminating computation

Program Termination

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

From SOS to an interpreter

- Implement semantics of arithmetic and Boolean expressions
- Implement SOS as a function:
  - input: S, s
  - ouput: γ such that <S, s> $\Rightarrow \gamma$
  - either γ = <S', s'> or γ = s'
- The function is recursive
  - call tree matches derivation tree of each step
- Iteratively call the SOS function until a final or stuck configuration is reached
- The call sequence matches the derivation sequence
- In While: the derivation sequence is unique, and there are no stuck configurations

LIVE CODE DEMO

`sos.py`