Concepts of Programming Languages – Recitation 2: Natural Operational Semantics

Oded Padon
odedp@mail.tau.ac.il

Reference:
Semantics with Applications by H. Nielson and F. Nielson – Ch. 2
http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html

Formal Semantics

- Operational Semantics
  - The meaning of the program is described "operationally"
  - **Natural Operational Semantics ← today!**
  - Structural Operational Semantics
- Denotational Semantics
  - The meaning of the program is an input/output relation
  - Mathematically challenging but complicated
- Axiomatic Semantics
  - The meaning of the program are observed properties

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

Example While Program

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

General Notations

- Syntactic categories
  - Var the set of program variables
  - Aexp the set of arithmetic expressions
  - Bexp the set of Boolean expressions
  - Stm set of program statements
- Semantic categories
  - Natural values N={0, 1, 2, …}
  - Truth values T={ff, tt}
  - States State = Var $\rightarrow$ N
  - Lookup in a state s: s x
  - Update of a state s: s [x $\mapsto$ 5]

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

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

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

Semantic Equivalence

- We say that e_1 is semantically equivalent to e_2 ($e_1 \approx e_2$) when $A[\![e_1]\!] = A[\![e_2]\!]$
- Which of the following expressions are equivalent?
  - $2 \approx 1 + 1$?
  - $x + y \approx y + x$?
  - $x + x \approx 2 * x$?
  - $(x + y) * (x - y) \approx x*x - y*y$?

  - x := 10; y := 20; z := x+y; z := 30;
    x+y ≈ 30 ?

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

Examples for $\rightarrow$

- $\langle y := 2, s_0[x \mapsto 1] \rangle \rightarrow s_0[x \mapsto 1]\ [y \mapsto 2]$
- $\langle x := x+1, s_0 \rangle \rightarrow s_0[x \mapsto 1]$
- $\langle x := x+1, s_0[x \mapsto 1] \rangle \rightarrow s_0[x \mapsto 2]$
- $\langle x := x+1 ; x := x+1, s_0 \rangle \rightarrow s_0[x \mapsto 2]$
- $\langle$**if** $x > 0$ **then** $y := 2$ **else** $y := 3, s_0 \rangle \rightarrow s_0[y \mapsto 3]$
- $\langle$**if** $x > 0$ **then** $y := 2$ **else** $y := 3, s_0[x \mapsto 1] \rangle \rightarrow s_0[x \mapsto 1]\ [y \mapsto 2]$
- $\langle x := x+1 ;$ **if** $x > 0$ **then** $y := 2$ **else** $y := 3, s_0 \rangle \rightarrow s_0[x \mapsto 1]\ [y \mapsto 2]$
- $\langle$**while** $x < 5$ **do** $(x := x+2 ; y := y+10), s_0 \rangle \rightarrow s_0[x \mapsto 6]\ [y \mapsto 30]$
- $\langle$(**while** $x < 5$ **do** $(x := x+2 ; y := y+10)) ;$
  (**while** $x > 0$ **do** $x := x-5), s_0 \rangle \rightarrow s_0[x \mapsto -4]\ [y \mapsto 30]$
- $\langle$**while** $x \geq 0$ **do** $x := x+1, s_0 \rangle \rightarrow$?
- NOT $\langle$**while** $x \geq 0$ **do** $x := x+1, s_0 \rangle \rightarrow s'$ for any s'

s_0: state which assigns zero to all variables

Formally defining $\rightarrow$

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

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

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

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

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

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

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

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

Example of Inference

<x:=x+1 ; **if** x > 0 **then** y := 2 **else** y:= 3, s_0 > $\rightarrow$ s_0[x $\mapsto$1] [y $\mapsto$2]

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

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

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

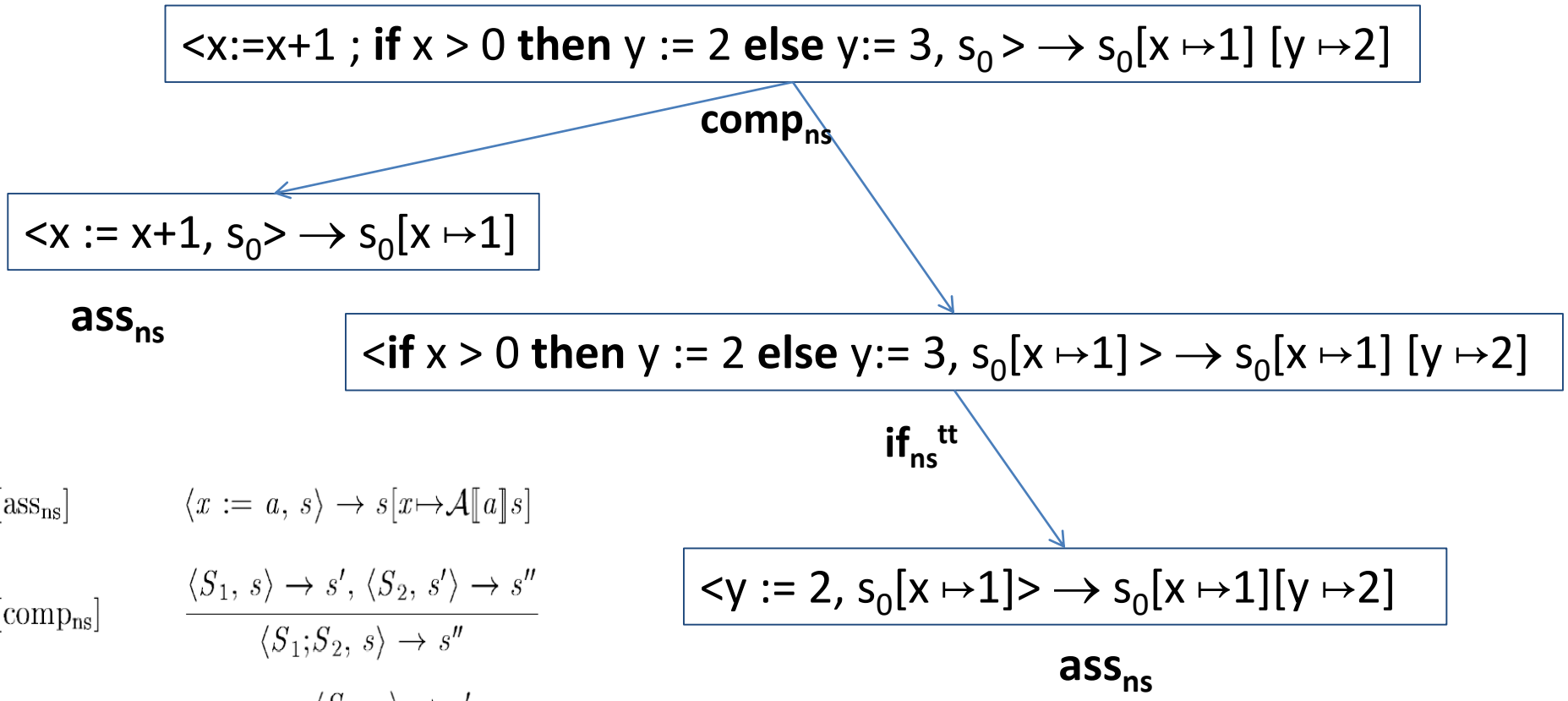
$$[\mathrm{if}^{\mathrm{ff}}_{\mathrm{ns}}] \qquad \frac{\langle S_2,\ s\rangle \rightarrow s'}{\langle \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2,\ s\rangle \rightarrow s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

1. <y := 2, s_0[x $\mapsto$1]> $\rightarrow$ s_0[x $\mapsto$1] [y $\mapsto$2] # by **ass_{ns}**
2. <**if** x > 0 **then** y := 2 **else** y:= 3, s_0[x $\mapsto$1] > $\rightarrow$ s_0[x $\mapsto$1] [y $\mapsto$2] # by **if_{ns}^{tt}** (1)
3. <x := x+1, s_0> $\rightarrow$ s_0[x $\mapsto$1] # by **ass_{ns}**
4. <x:=x+1 ; **if** x > 0 **then** y := 2 **else** y:= 3, s_0 > $\rightarrow$ s_0[x $\mapsto$1] [y $\mapsto$2] # by **$comp_{ns}$** (2,3)

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



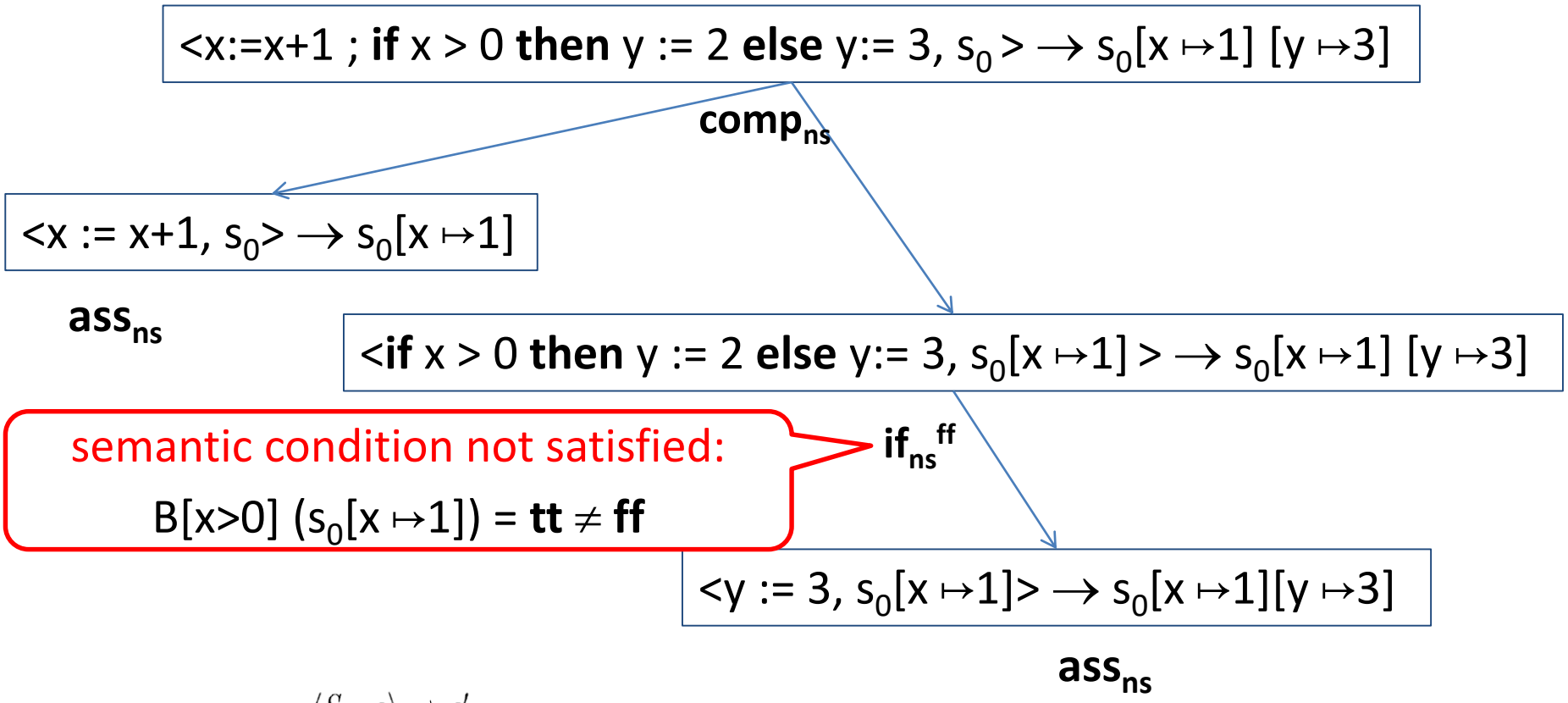
$$[\text{ass}_{\text{ns}}] \quad \langle x := a,\ s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

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

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

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

Bad Derivation Tree



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

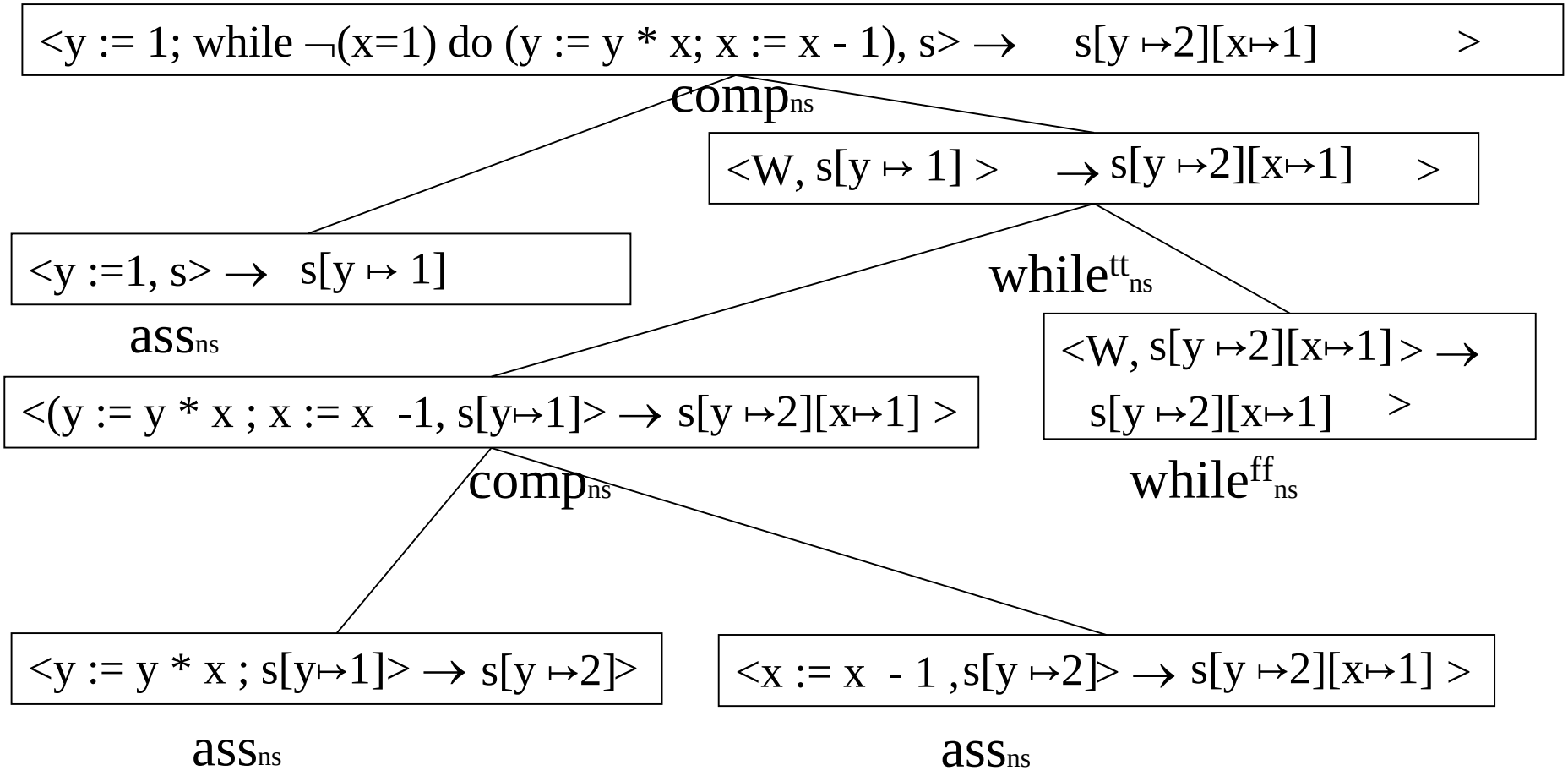
$$[\text{if}_{\text{ns}}^{\text{ff}}] \quad \frac{\langle S_2, s\rangle \to s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \to s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \text{ff}$$

Top Down Evaluation of Derivation Trees

- Given a program S and an input state s
- Find an output state s' such that
  $\langle S, s \rangle \rightarrow s'$
- 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

Example of Top Down Tree Construction

- Input state s such that s x = 2
- Factorial program



From NOS to an interpreter

- Implement semantics of arithmetic and Boolean expressions
- Implement NOS as a function:
  - input: S, s
  - ouput: s' such that <S, s> $\rightarrow$s'
- The function is recursive
  - Call tree matches derivation tree
  - Axioms are the recursion base cases
- While is deterministic: s' and the derivation tree is unique

LIVE CODE DEMO

1. `while_ast.py`
2. `expr.py`
3. `nos.py`
4. `nos_tree.py`