Concepts of Programming Languages – Recitation 4:

Structural Operational Semantics (SOS) and From Semantics to Interpreters

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

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

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

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

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

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

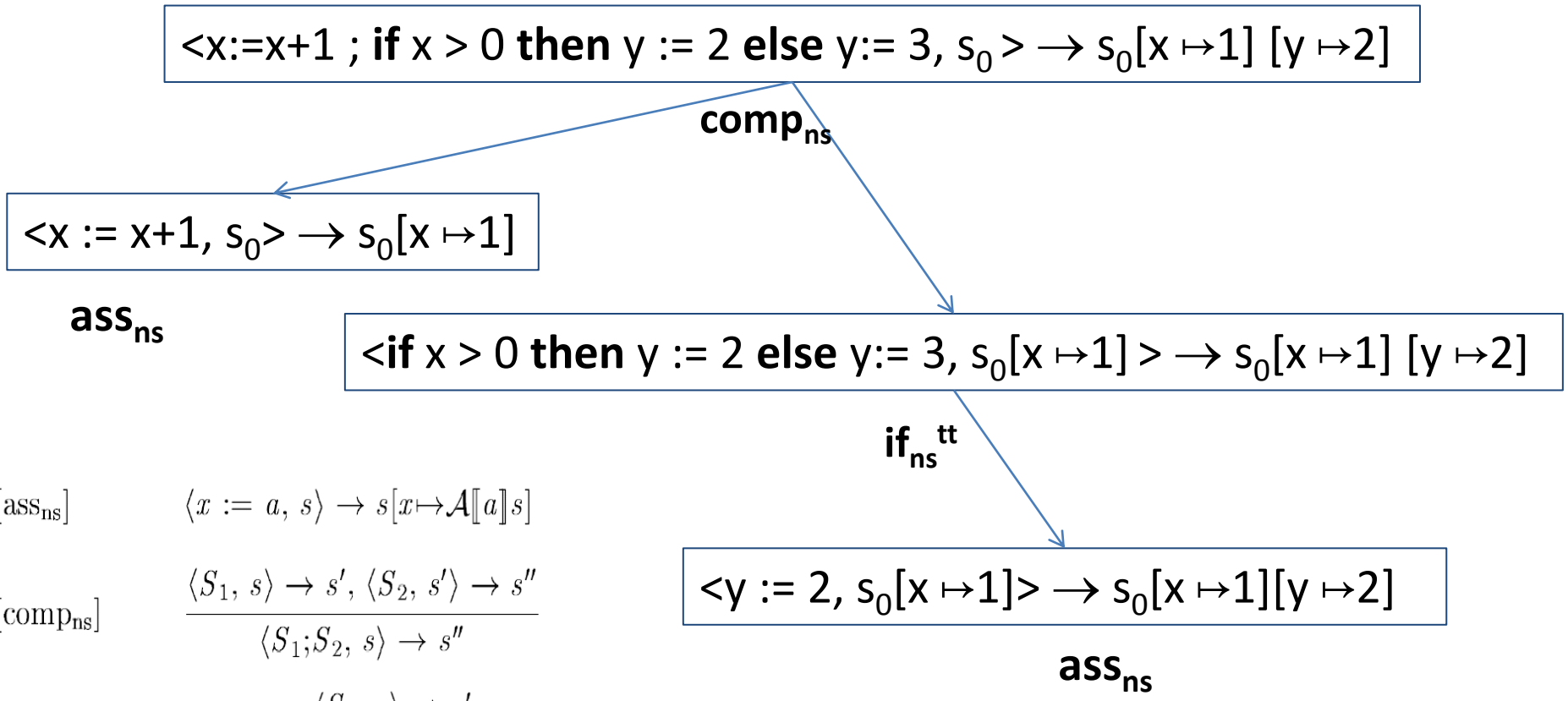
$$[\text{while}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \rightarrow s', \langle \texttt{while } b \texttt{ do } S, s'\rangle \rightarrow s''}{\langle \texttt{while } b \texttt{ do } S, s\rangle \rightarrow s''} \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

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

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



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

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

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

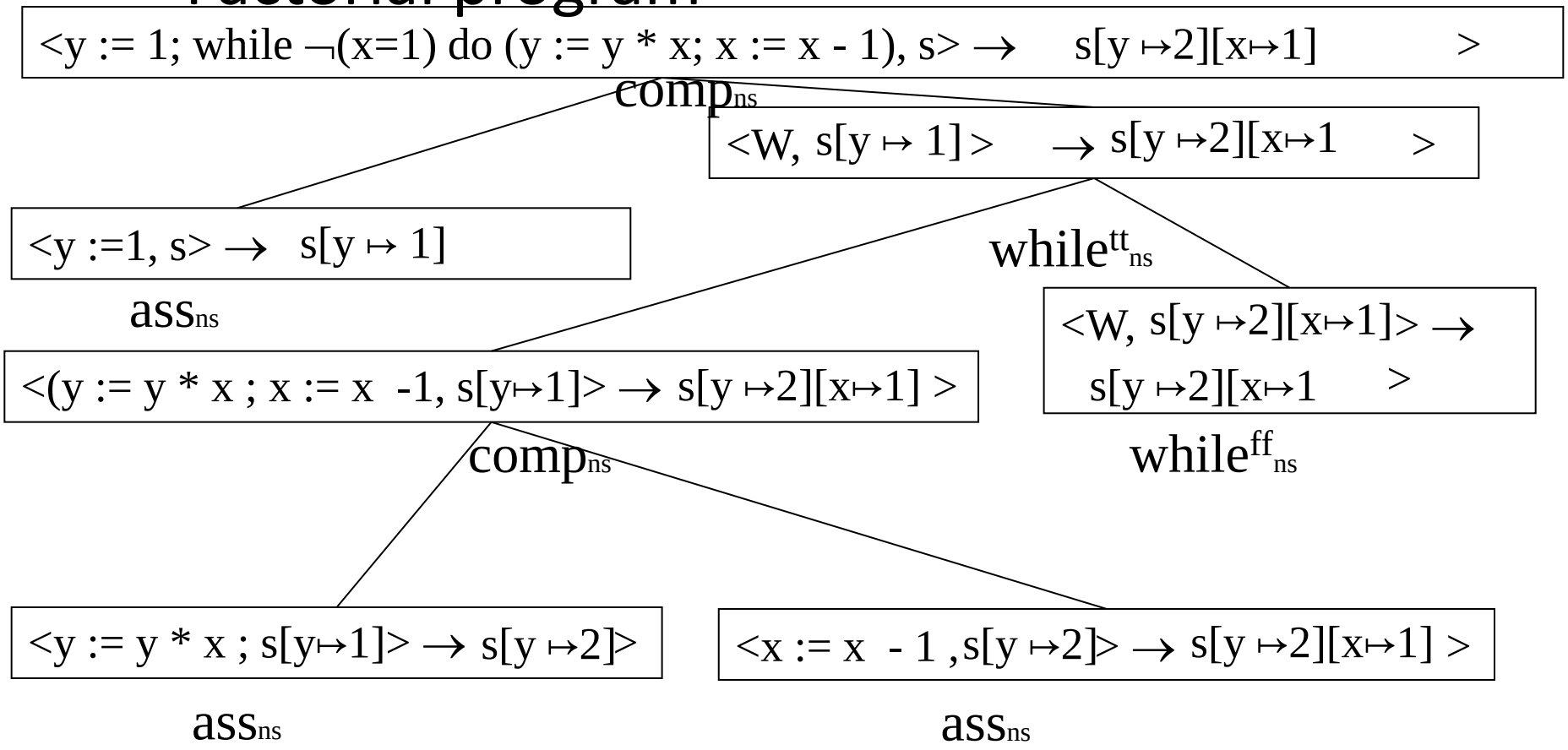
$$[\text{if}^{\text{ff}}_{\text{ns}}] \qquad \frac{\langle S_2,\ s\rangle \to s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2,\ s\rangle \to s'} \quad \text{if } \mathcal{B}[\![b]\!]s = \text{ff}$$

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`

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`

Proving properties of SOS

- Many properties are proved by induction on the length of derivation sequence
- Prove that the property holds for all derivation sequences of length 0
- Prove that the property holds for all other derivation sequences:
  - Assume the property holds for all sequences of length k or less (induction hypothesis)
  - Show that the property holds for sequences of length k+1

Example Property: Sequential Composition

- If **$\langle S_1 ; S_2, s\rangle \Rightarrow^k s''$** then there exists a state **s'** and numbers **k_1** and **k_2** such that:
  - $\langle S_1, s\rangle \Rightarrow^{k1} s'$
  - $\langle S_2, s'\rangle \Rightarrow^{k2} s''$
  - $k = k_1 + k_2$

Proof

- If $\langle S_1 ; S_2, s \rangle \Rightarrow^k s''$ then there exists a state s' and numbers k_1 and k_2 such that: $\langle S_1, s \rangle \Rightarrow^{k1} s'$, $\langle S_2, s' \rangle \Rightarrow^{k2} s''$ and $k = k_1 + k_2$

- Base case: k=0, holds vacuously

Inductive Step

- **Assume for all $j \leq k$**: if $\langle S_1 ; S_2, s\rangle \Rightarrow^j s''$ then there exists a state s' and numbers j_1 and j_2 such that: $\langle S_1, s\rangle \Rightarrow^{j1} s'$, $\langle S_2, s'\rangle \Rightarrow^{j2} s''$ and $j = j_1 + j_2$
- Assume $\langle S_1 ; S_2, s\rangle \Rightarrow^{k+1} s''$, then, by definition of $\Rightarrow^n$:
  - $\langle S_1 ; S_2, s\rangle \Rightarrow \gamma$
  - $\gamma \Rightarrow^k s''$
  - γ is not a final or stuck configuration
- Split according to two cases of $\langle S_1 ; S_2, s\rangle \Rightarrow \gamma$

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

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

Case $\mathbf{comp}_{\mathbf{sos}}^{\mathbf{1}}$

- $\gamma = \langle S'_1 ; S_2, s' \rangle$
- $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S'_1 ; S_2, s' \rangle$ $\quad [\mathrm{comp}^1_{\mathrm{sos}}] \quad \dfrac{\langle S_1, s\rangle \Rightarrow \langle S'_1, s'\rangle}{\langle S_1;S_2, s\rangle \Rightarrow \langle S'_1;S_2, s'\rangle}$
- $\langle S'_1 ; S_2, s' \rangle \Rightarrow^k s''$
- By induction hypothesis on $\langle S'_1 ; S_2, s' \rangle \Rightarrow^k s''$ we get j_1, j_2 and state $\hat{s}$ such that:
  - $k = j_1 + j_2$
  - $\langle S'_1, s' \rangle \Rightarrow^{j1} \hat{s}$
  - $\langle S_2, \hat{s} \rangle \Rightarrow^{j2} s''$
- Set $k_1 = j_1 + 1$, $k_2 = j_2$ and get:
  - $k+1 = k_1 + k_2$
  - $\langle S_1, s \rangle \Rightarrow^{k1} \hat{s}$ since $\langle S_1 , s \rangle \Rightarrow \langle S'_1 , s' \rangle \Rightarrow^{j1} \hat{s}$
  - $\langle S_2, \hat{s} \rangle \Rightarrow^{k2} s''$

Case $\mathbf{comp}_{\mathbf{sos}}^{\mathbf{2}}$

- $\gamma = \langle S_2, s' \rangle$
- $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle$
- $\langle S_2, s' \rangle \Rightarrow^k s''$

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

- Set $k_1 = 1$, $k_2 = k$ and get:
  - $k+1 = k_1 + k_2$
  - $\langle S_1, s \rangle \Rightarrow^{k1} s'$
  - $\langle S_2, s' \rangle \Rightarrow^{k2} s''$