x = 1

let x = 1 in ...

x(1).

!x(1)

x.set(1)

Programming Language Theory

Small-step Operational Semantics (aka Structured Operational Semantics)

Ralf Lämmel

Big-step style

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

Easier to understand

Small-step style

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

"More versatile"

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

SOS (statements)

$[\text{ass}_{\text{sos}}]$	$\langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$
$[\text{skip}_{\text{sos}}]$	$\langle \texttt{skip},\ s\rangle \Rightarrow s$
$[\text{comp}^1_{\text{sos}}]$	$\dfrac{\langle S_1,\ s\rangle \Rightarrow \langle S'_1,\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S'_1;S_2,\ s'\rangle}$
$[\text{comp}^2_{\text{sos}}]$	$\dfrac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$
$[\text{if}^{\text{tt}}_{\text{sos}}]$	$\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$
$[\text{if}^{\text{ff}}_{\text{sos}}]$	$\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$
$[\text{while}_{\text{sos}}]$	$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \Rightarrow \langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip},\ s\rangle$

Prolog as a sandbox for small-step operational semantics

https://slps.svn.sourceforge.net/svnroot/slps/topics/NielsonN07/Prolog/While/SOS/

Architecture of the interpreter

- **Makefile**: see “make test”
- **main.pro**: main module to compose all other modules
- **exec.pro**: statement execution
- **eval.pro**: expression evaluation
- **map.pro**: abstract data type for maps (states)
- **test.pro**: framework for unit testing

Same as NS

Empty statement

```
step( (skip,M),
      M ).
```

Sequential composition

```
step( (seq(S1,S2),M1),
      (seq(S3,S2),M2) )
 :-
    step((S1,M1),(S3,M2)).

step( (seq(S1,S2),M1),
      (S2,M2) )
 :-
    step((S1,M1),M2),
    \+ M2 = (_,_).
```

Assignment

```
step( (assign(X,A),M1),
       M2 )
 :-
   evala(A,M1,Y),
   update(M1,X,Y,M2).
```

Conditional statement

```
step( (ifthenelse(B,S1,_),M),
         (S1,M) )
 :-
    evalb(B,M,tt).

step( (ifthenelse(B,_,S2),M),
         (S2,M) )
 :-
    evalb(B,M,ff).
```

Loop statement

```
step( (while(B,S),M),
        (ifthenelse(B,seq(S,while(B,S)),skip),M) ).
```

Transitive closure

```
execute( (S1,M1),
         M3 )
 :-
   step((S1,M1),(S2,M2)),
   execute((S2,M2),M3).

execute( (S1,M1),
         M2 )
 :-
   step((S1,M1),M2),
   \+ M2 = (_,_).
```

Transition systems in semantics

Semantics of statements

Syntactic Category

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

Meaning of the syntactic category:

$\mathcal{S} : \mathsf{Stm} \to (\mathsf{State} \hookrightarrow \mathsf{State})$

Two operational semantics

- Natural Semantics
- Structural Operational Semantics

specified by transition systems

Transition systems

$(\Gamma, T, \triangleright)$

- Γ: a set of configurations
- T: a set of terminal configurations
  $T \subseteq \Gamma$
- $\triangleright$: a transition relation
  $\triangleright \subseteq \Gamma \times \Gamma$

Big step versus small step

- Big-step semantics
  - ✦ aka Natural semantics
  - ✦ ***One*** (fewer) transition(s)
  - ✦ *Computation steps modeled by derivation tree*
- Small-step semantics
  - ✦ aka Structured Operational Semantics (SOS)
  - ✦ ***Many*** transitions
  - ✦ *Computation steps modeled by transitions*

Big step operational semantics: describe how the "final" result of the computation is obtained.

Transition system: $(\Gamma, T, \rightarrow)$

- $\Gamma = \{(S, s) \mid S \in \text{While}, s \in \text{State}\}$
  $\cup$ State
- $T =$ State
- $\rightarrow \subseteq \{(S, s) \mid S \in \text{While}, s \in \text{State}\}$
  $\times$ State

Typical transition:

$$(S, s) \rightarrow s'$$

where

S is the program
s is the initial state
s' is the final state

Small step operational semantics: describe how the individual steps of the computation take place.

Transition system: $(\Gamma, T, \Rightarrow)$

- $\Gamma = \{(S, s) \mid S \in \text{While}, s \in \text{State}\}$
  $\cup$ State
- $T = \text{State}$
- $\Rightarrow \subseteq \{(S, s) \mid S \in \text{While}, s \in \text{State}\}$
  $\times\ \Gamma$

Two typical transitions:

- the computation has not been completed after one step of computation:
  $$(S, s) \Rightarrow (S', s')$$
- the computation is completed after one step of computation:
  $$(S, s) \Rightarrow s'$$

Big step versus small step

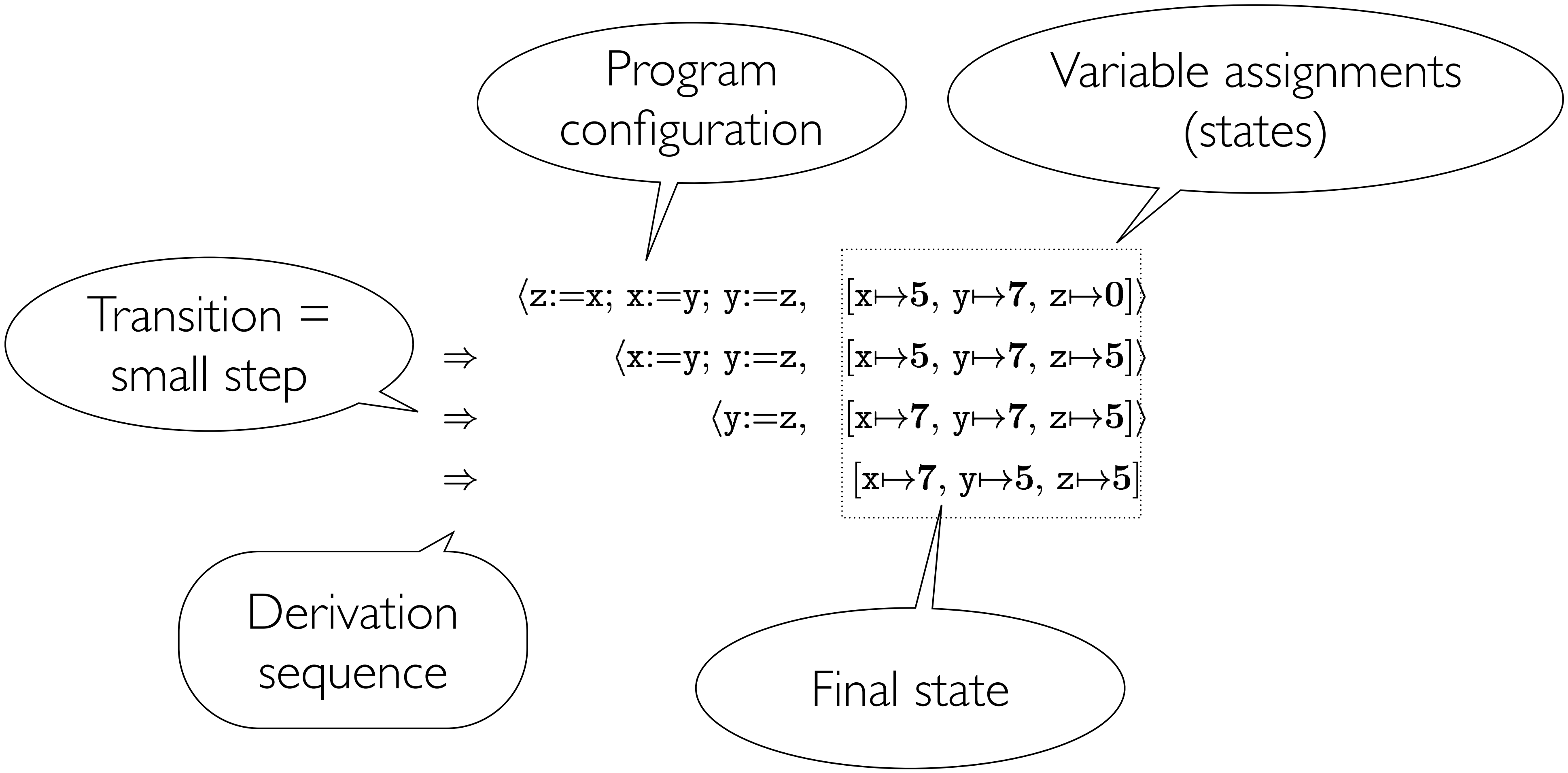
$$\cfrac{\cfrac{\langle \text{z:=x},\ s_0\rangle \to s_1 \qquad \langle \text{x:=y},\ s_1\rangle \to s_2}{\langle \text{z:=x; x:=y},\ s_0\rangle \to s_2} \qquad \langle \text{y:=z},\ s_2\rangle \to s_3}{\langle \text{z:=x; x:=y; y:=z},\ s_0\rangle \to s_3}$$

Derivation tree

Transition = big step

$$\begin{array}{lcl} s_0 & = & [\text{x}\mapsto\mathbf{5},\ \text{y}\mapsto\mathbf{7},\ \text{z}\mapsto\mathbf{0}] \\ s_1 & = & [\text{x}\mapsto\mathbf{5},\ \text{y}\mapsto\mathbf{7},\ \text{z}\mapsto\mathbf{5}] \\ s_2 & = & [\text{x}\mapsto\mathbf{7},\ \text{y}\mapsto\mathbf{7},\ \text{z}\mapsto\mathbf{5}] \\ s_3 & = & [\text{x}\mapsto\mathbf{7},\ \text{y}\mapsto\mathbf{5},\ \text{z}\mapsto\mathbf{5}] \end{array}$$

Big step versus **small step**



$$
\begin{array}{lrl}
 & \langle \mathrm{z{:=}x;\ x{:=}y;\ y{:=}z}, & [\mathrm{x}\mapsto\mathbf{5},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{0}]\rangle \\
\Rightarrow & \langle \mathrm{x{:=}y;\ y{:=}z}, & [\mathrm{x}\mapsto\mathbf{5},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{5}]\rangle \\
\Rightarrow & \langle \mathrm{y{:=}z}, & [\mathrm{x}\mapsto\mathbf{7},\ \mathrm{y}\mapsto\mathbf{7},\ \mathrm{z}\mapsto\mathbf{5}]\rangle \\
\Rightarrow & & [\mathrm{x}\mapsto\mathbf{7},\ \mathrm{y}\mapsto\mathbf{5},\ \mathrm{z}\mapsto\mathbf{5}]
\end{array}
$$

Big step versus **small step**

Derivation sequence (many transitions)

$$\begin{aligned}
&\langle (\mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y});\ \mathtt{y} := \mathtt{z},\ s_0\rangle \\
&\qquad \Rightarrow \langle \mathtt{x} := \mathtt{y};\ \mathtt{y} := \mathtt{z},\ s_0[\mathtt{z}\mapsto\mathbf{5}]\rangle \\
&\qquad \Rightarrow \langle \mathtt{y} := \mathtt{z},\ (s_0[\mathtt{z}\mapsto\mathbf{5}])[\mathtt{x}\mapsto\mathbf{7}]\rangle \\
&\qquad \Rightarrow ((s_0[\mathtt{z}\mapsto\mathbf{5}])[\mathtt{x}\mapsto\mathbf{7}])[\mathtt{y}\mapsto\mathbf{5}]
\end{aligned}$$

Derivation tree (for each single step)

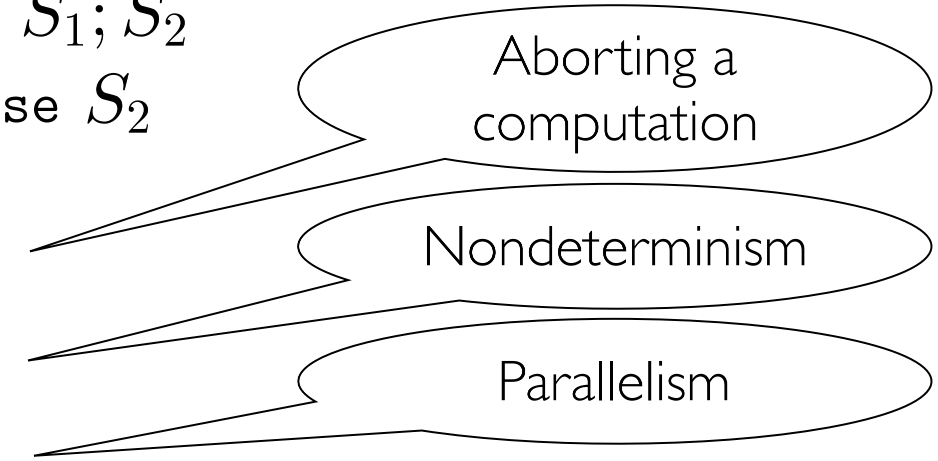
$$\dfrac{\dfrac{\langle \mathtt{z} := \mathtt{x},\ s_0\rangle \Rightarrow s_0[\mathtt{z}\mapsto\mathbf{5}]}{\langle \mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y},\ s_0\rangle \Rightarrow \langle \mathtt{x} := \mathtt{y},\ s_0[\mathtt{z}\mapsto\mathbf{5}]\rangle}}{\langle (\mathtt{z} := \mathtt{x};\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

Execution of $\langle S,s\rangle$ terminates successfully if $\langle S,s\rangle \Rightarrow^k s'$ for some k and s'.

Execution loops if there is an infinite derivation sequence.

Extensions of ***While***

$$
\begin{aligned}
S \;::=\;& x := a \mid \texttt{skip} \mid S_1; S_2 \\
\mid\;& \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\
\mid\;& \texttt{while } b \texttt{ do } S \\
\mid\;& \texttt{abort} \\
\mid\;& S_1 \texttt{ or } S_2 \\
\mid\;& S_1 \texttt{ par } S_2
\end{aligned}
$$



Adding **abort**

Configurations:

$$\{(S, s) \mid S \in \mathsf{While}^{abort},\ s \in \mathsf{State}\} \\ \cup\ \mathsf{State}$$

Transition relation for NS:

unchanged

Transition relation for SOS:

unchanged

NS vs. SOS

`abort` vs. `while true do skip` **?**

- ***Natural semantics***: We cannot distinguish between abnormal termination and nontermination. (One could extend the set of final configurations to specifically distinguish "stuck" configurations due to abort.)

- ***SOS***: Nontermination is reflected by infinite derivation sequences while abortion is reflected by finite derivation sequences ending in a "stuck" configuration.

Adding nondeterminism

$\texttt{x} := \texttt{1 or (x} := \texttt{2; x} := \texttt{x} + \texttt{2)}$ assigns 1 or 4 to x.

$[\mathrm{or}^1_{\mathrm{sos}}]$	$\langle S_1 \texttt{ or } S_2,\, s\rangle \Rightarrow \langle S_1,\, s\rangle$
$[\mathrm{or}^2_{\mathrm{sos}}]$	$\langle S_1 \texttt{ or } S_2,\, s\rangle \Rightarrow \langle S_2,\, s\rangle$
$[\mathrm{or}^1_{\mathrm{ns}}]$	$\dfrac{\langle S_1,\, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2,\, s\rangle \to s'}$
$[\mathrm{or}^2_{\mathrm{ns}}]$	$\dfrac{\langle S_2,\, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2,\, s\rangle \to s'}$

NS vs. SOS

Does the following program terminate?

$(\texttt{while true do skip})\ \texttt{or}\ (\texttt{x} := 2;\ \texttt{x} := \texttt{x}+2)$

- ***Natural semantics***: Nondeterminism suppresses looping, if possible. That is, we obtain *one* derivation tree (transition) for the terminating option.

- ***SOS***: Nondeterminism does not suppress looping. That is, we obtain *two* transition sequences, and one of them is non-terminating.

Adding parallelism

$\texttt{x := 1 par (x := 2; x := x+2)}$ assigns 1, 3, or 4 to x.

Transition relation for SOS:

$$\frac{(S_1, s) \Rightarrow (S_1', s')}{(S_1 \ \texttt{par}\ S_2, s) \Rightarrow (S_1' \ \texttt{par}\ S_2, s')}$$

$$\frac{(S_1, s) \Rightarrow s'}{(S_1 \ \texttt{par}\ S_2, s) \Rightarrow (S_2, s')}$$

$$\frac{(S_2, s) \Rightarrow (S_2', s')}{(S_1 \ \texttt{par}\ S_2, s) \Rightarrow (S_1 \ \texttt{par}\ S_2', s')}$$

$$\frac{(S_2, s) \Rightarrow s'}{(S_1 \ \texttt{par}\ S_2, s) \Rightarrow (S_1, s')}$$

Transition relation for NS:

$$\frac{(S_1, s) \to s', \quad (S_2, s') \to s''}{(S_1 \ \texttt{par}\ S_2, s) \to s''}$$

$$\frac{(S_2, s) \to s', \quad (S_1, s') \to s''}{(S_1 \ \texttt{par}\ S_2, s) \to s''}$$

NS vs. SOS

- ***Nat. sem.***: Each constituent of par is executed in one big step. Hence, interleaving of computations is *not* achieved.

  `x := 1 par (x := 2; x := x+2)` evaluates to 1, 4.

- ***SOS***: The constituents of par are executed in many small steps. Hence interleaving of computations is achieved.

  `x := 1 par (x := 2; x := x+2)` evaluates to 1, 3, or 4.

Semantics and proofs

- Three approaches to semantics:
  - Compositional definitions
  - Natural semantics
  - SOS
- Three corresponding proof principles:
  - Induction on the syntactic structure
  - Induction on the shape of derivation trees
  - Induction on the length of derivation sequences

Induction on the Length of Derivation Sequences

1: Prove that the property holds for all derivation sequences of length 0.

2: Prove that the property holds for all other derivation sequences: Assume that the property holds for all derivation sequences of length at most k (this is called the *induction hypothesis*) and show that it holds for derivation sequences of length k+1.

Equivalence of semantics

$\mathcal{S}_{\text{ns}}$: $\mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$

$\mathcal{S}_{\text{sos}}$: $\mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$

$$\mathcal{S}_{\text{ns}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\, s\rangle \to s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\, s\rangle \Rightarrow^{*} s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

Only for basic While!

Theorem 2.26 For every statement S of **While** we have $\mathcal{S}_{\text{ns}}[\![S]\!] = \mathcal{S}_{\text{sos}}[\![S]\!]$.

Theorem 2.26 For every statement S of **While** we have $\mathcal{S}_{\mathrm{ns}}[\![S]\!] = \mathcal{S}_{\mathrm{sos}}[\![S]\!]$.

Proof Summary for While:

Equivalence of two Operational Semantics

1: Prove by *induction on the shape of derivation trees* that for each derivation tree in the natural semantics there is a corresponding finite derivation sequence in the structural operational semantics.

2: Prove by *induction on the length of derivation sequences* that for each finite derivation sequence in the structural operational semantics there is a corresponding derivation tree in the natural semantics.

Theorem 2.26 For every statement S of **While** we have $\mathcal{S}_{\mathrm{ns}}[\![S]\!] = \mathcal{S}_{\mathrm{sos}}[\![S]\!]$.

Lemma 2.27 For every statement S of **While** and states s and s' we have

$$\langle S,\, s\rangle \to s' \text{ implies } \langle S,\, s\rangle \Rightarrow^{*} s'.$$

So if the execution of S from s terminates in the natural semantics then it will terminate in the same state in the structural operational semantics.

Lemma 2.28 For every statement S of **While**, states s and s' and natural number k we have that

$$\langle S,\, s\rangle \Rightarrow^{\mathrm{k}} s' \text{ implies } \langle S,\, s\rangle \to s'.$$

So if the execution of S from s terminates in the structural operational semantics then it will terminate in the same state in the natural semantics.

Let's focus on this lemma for the sake of exercising induction on length of derivation sequences.

$\langle S,\ s\rangle \Rightarrow^{\mathrm{k}} s'$ implies $\langle S,\ s\rangle \rightarrow s'$.

Proof: The proof proceeds by induction on the length of the derivation sequence $\langle S,\ s\rangle \Rightarrow^{\mathrm{k}} s'$, that is by induction on k.

If k=0 then the result holds vacuously.

To prove the induction step we assume that the lemma holds for $\mathrm{k} \leq \mathrm{k}_0$ and we shall then prove that it holds for k_0+1. We proceed by cases on how the first step of $\langle S,\ s\rangle \Rightarrow^{\mathrm{k}_0+1} s'$ is obtained, that is by inspecting the derivation tree for the first step of computation in the structural operational semantics.

The case [ass$_{\mathrm{sos}}$]: Straightforward (and $\mathrm{k}_0 = 0$).

The case [skip$_{\mathrm{sos}}$]: Straightforward (and $\mathrm{k}_0 = 0$).

Clearly, the cases for compound statement forms somehow have to take apart the *k* transitions to account for the transitions needed for the constituents.

Taking apart statement composition

Lemma [2.19] If $(S_1; S_2, s) \Rightarrow^k s''$ then
there exists s', k_1 and k_2 such that

$(S_1, s) \Rightarrow^{k_1} s'$,
$(S_2, s') \Rightarrow^{k_2} s''$ and
$k = k_1 + k_2$

Proof We proceed by induction on the number k.

Proof by induction on the length of derivation sequences

Lemma 2.19 If $\langle S_1;S_2,\ s\rangle \Rightarrow^{\mathrm{k}} s''$ then there exists a state s' and natural numbers k_1 and k_2 such that $\langle S_1,\ s\rangle \Rightarrow^{\mathrm{k}_1} s'$ and $\langle S_2,\ s'\rangle \Rightarrow^{\mathrm{k}_2} s''$ where $\mathrm{k} = \mathrm{k}_1+\mathrm{k}_2$.

Proof: The proof is by induction on the number k, that is by induction on the length of the derivation sequence $\langle S_1;S_2,\ s\rangle \Rightarrow^{\mathrm{k}} s''$.

If $\mathrm{k} = 0$ then the result holds vacuously.

For the induction step we assume that the lemma holds for $\mathrm{k} \leq \mathrm{k}_0$ and we shall prove it for k_0+1. So assume that

$$\langle S_1;S_2,\ s\rangle \Rightarrow^{\mathrm{k}_0+1} s''$$

This means that the derivation sequence can be written as

$$\langle S_1;S_2,\ s\rangle \Rightarrow \gamma \Rightarrow^{\mathrm{k}_0} s''$$

for some configuration γ. Now one of two cases applies depending on which of the two rules $[\mathrm{comp}^1_{\mathrm{sos}}]$ and $[\mathrm{comp}^2_{\mathrm{sos}}]$ was used to obtain $\langle S_1;S_2,\ s\rangle \Rightarrow \gamma$.

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

In the first case where [comp$^1_{\text{sos}}$] is used we have

$$\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle$$

because

$$\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle$$

We therefore have

$$\langle S_1';S_2,\ s'\rangle \Rightarrow^{\mathrm{k}_0} s''$$

and the induction hypothesis can be applied to this derivation sequence because it is shorter than the one we started with. This means that there is a state s_0 and natural numbers k_1 and k_2 such that

$$\langle S_1',\ s'\rangle \Rightarrow^{\mathrm{k}_1} s_0 \text{ and } \langle S_2,\ s_0\rangle \Rightarrow^{\mathrm{k}_2} s''$$

where $\mathrm{k}_1+\mathrm{k}_2=\mathrm{k}_0$. Using that $\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle$ and $\langle S_1',\ s'\rangle \Rightarrow^{\mathrm{k}_1} s_0$ we get

$$\langle S_1,\ s\rangle \Rightarrow^{\mathrm{k}_1+1} s_0$$

We have already seen that $\langle S_2,\ s_0\rangle \Rightarrow^{\mathrm{k}_2} s''$ and since $(\mathrm{k}_1+1)+\mathrm{k}_2 = \mathrm{k}_0+1$ we have proved the required result.

$$\boxed{\frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}}$$

The second possibility is that $[\mathrm{comp}^2_{\mathrm{sos}}]$ has been used to obtain the derivation $\langle S_1;S_2,\ s\rangle \Rightarrow \gamma$. Then we have

$$\langle S_1,\ s\rangle \Rightarrow s'$$

and γ is $\langle S_2,\ s'\rangle$ so that

$$\langle S_2,\ s'\rangle \Rightarrow^{\mathrm{k}_0} s''$$

The result now follows by choosing $\mathrm{k}_1=1$ and $\mathrm{k}_2=\mathrm{k}_0$. $\square$

$\langle S,\ s\rangle \Rightarrow^{\mathrm{k}} s'$ implies $\langle S,\ s\rangle \rightarrow s'$. cont’d

The cases $[\text{comp}^1_{\text{sos}}]$ and $[\text{comp}^2_{\text{sos}}]$: In both cases we assume that

$$\langle S_1;S_2,\ s\rangle \Rightarrow^{\mathrm{k}_0+1} s''$$

We can now apply Lemma 2.19 and get that there exists a state s' and natural numbers k_1 and k_2 such that

$$\langle S_1,\ s\rangle \Rightarrow^{\mathrm{k}_1} s' \text{ and } \langle S_2,\ s'\rangle \Rightarrow^{\mathrm{k}_2} s''$$

where $\mathrm{k}_1+\mathrm{k}_2=\mathrm{k}_0+1$. The induction hypothesis can now be applied to each of these derivation sequences because $\mathrm{k}_1 \leq \mathrm{k}_0$ and $\mathrm{k}_2 \leq \mathrm{k}_0$. So we get

$$\langle S_1,\ s\rangle \rightarrow s' \text{ and } \langle S_2,\ s'\rangle \rightarrow s''$$

Using $[\text{comp}_{\text{ns}}]$ we now get the required $\langle S_1;S_2,\ s\rangle \rightarrow s''$.

Further composites omitted.

- **Summary**: *Small-step operational semantics*
  - ✦ *Transitions are steps of computation.*
  - ✦ *Computations are derivation sequences.*
  - ✦ *Some extensions are more convenient with SOS.*
- **Prepping**: *"Semantics with applications"*
  - ✦ *Chapter 2.2 - Chapter 2.5*
- **Lab**: *Operational Semantics in Prolog*
- **Outlook**:
  - ✦ Type systems
  - ✦ The lambda calculus