Formal Methods and Functional Programming

Operational Semantics

Peter Müller

Chair of Programming Methodology
ETH Zurich

Operational Semantics of Statements

- Evaluation of an expression in a state yields a value

```
x + 2 * y
```

$\mathcal{A} : \text{Aexp} \to \text{State} \to \text{Val}$

- Execution of a statement modifies the state

```
x := 2 * y
```

- Operational semantics describe how the state is modified during the execution of a statement

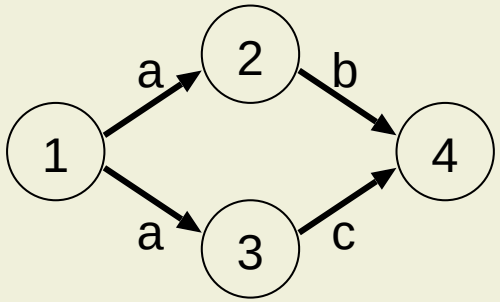
Big-Step and Small-Step Semantics

- Big-step semantics describe how the overall results of the executions are obtained
  - Natural semantics
- Small-step semantics describe how the individual steps of the computations take place
  - Structural operational semantics
  - Abstract state machines

2. Operational Semantics

Transition Systems

- A transition system is a tuple $(\Gamma, T, \rightarrow)$
  - Γ: a set of configurations
  - T: a set of terminal configurations, $T \subseteq \Gamma$
  - $\rightarrow$: a transition relation, $\rightarrow \subseteq \Gamma \times \Gamma$
- Example: Finite automaton



$$
\begin{aligned}
\Gamma &= \{\langle w, S\rangle \mid w \in \{a,b,c\}^*, S \in \{1,2,3,4\}\} \\
T &= \{\langle \epsilon, S\rangle \mid S \in \{1,2,3,4\}\} \\
\rightarrow &= \{(\langle aw,1\rangle \rightarrow \langle w,2\rangle), (\langle aw,1\rangle \rightarrow \langle w,3\rangle), \\
&\quad (\langle bw,2\rangle \rightarrow \langle w,4\rangle), (\langle cw,3\rangle \rightarrow \langle w,4\rangle) \mid w \in \{a,b,c\}^*\}
\end{aligned}
$$

Transitions in Natural Semantics

- Two types of configurations for operational semantics
  1. $\langle s, \sigma \rangle$, which represents that the statement s is to be executed in state σ
  2. σ, which represents a terminal state
- The transition relation $\rightarrow$ describes how executions take place
  - Typical transition: $\langle s, \sigma \rangle \rightarrow \sigma'$
  - Example: $\langle \texttt{skip}, \sigma \rangle \rightarrow \sigma$

$$\Gamma = \{\langle s, \sigma \rangle \mid s \in \mathsf{Stm}, \sigma \in \mathsf{State}\} \cup \mathsf{State}$$
$$T = \mathsf{State}$$
$$\rightarrow \subseteq \{\langle s, \sigma \rangle \mid s \in \mathsf{Stm}, \sigma \in \mathsf{State}\} \times \mathsf{State}$$

Rules

- We specify the transition relation by rules of the form

$$\text{Name}\ \frac{\varphi_1, \ldots, \varphi_n}{\psi}\ \text{if } \textit{Condition}$$

  where $\varphi_1, \ldots, \varphi_n$ and ψ are transitions

- Meaning of the rule

If *Condition* and $\varphi_1, \ldots, \varphi_n$ then ψ

- Terminology
  - $\varphi_1, \ldots, \varphi_n$ are called the premises of the rule
  - ψ is called the conclusion of the rule
  - A rule without premises is called axiom

Natural Semantics of IMP

- `skip` does not modify the state

$$\textsc{Skip}_{NS}\ \frac{}{\langle \texttt{skip}, \sigma \rangle \to \sigma}$$

- $x := e$ assigns the value of e to variable x

$$\textsc{Ass}_{NS}\ \frac{}{\langle x := e, \sigma \rangle \to \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]}$$

- Sequential composition $s_1 ; s_2$
  - First, s_1 is executed in state σ, leading to σ'
  - Then s_2 is executed in state σ', leading to σ''

$$\textsc{Seq}_{NS}\ \frac{\langle s_1, \sigma \rangle \to \sigma' \qquad \langle s_2, \sigma' \rangle \to \sigma''}{\langle s_1 ; s_2, \sigma \rangle \to \sigma''}$$

Natural Semantics of IMP (cont'd)

- Conditional statement `if` b `then` s_1 `else` s_2 `end`
  - If b holds, s_1 is executed
  - If b does not hold, s_2 is executed

$$\textsc{IfT}_{NS}\ \frac{\langle s_1, \sigma\rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \to \sigma'} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\textsc{IfF}_{NS}\ \frac{\langle s_2, \sigma\rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \to \sigma'} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = ff$$

Natural Semantics of IMP (cont'd)

- Loop statement `while` b `do` s `end`
  - If b holds, s is executed once, leading to state σ'
  - Then the whole while-statement is executed again in σ'

$$\text{WHT}_{NS}\ \frac{\langle s, \sigma\rangle \to \sigma' \qquad \langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma'\rangle \to \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

  - If b does not hold, the while-statement does not modify the state

$$\text{WHF}_{NS}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = ff$$

Rule Instantiations

- Rules are actually rule schemes
  - Meta-variables stand for any concrete variable, expression, statement, state, etc.
  - To apply rules, they have to be instantiated by selecting particular variables, expressions, statements, states, etc.

- Assignment rule scheme

$$\textsc{Ass}_{NS}\ \frac{}{\langle x := e, \sigma\rangle \to \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]}$$

- Assignment rule instance

$$\textsc{Ass}_{NS}\ \frac{}{\langle \mathrm{v} := \mathrm{v}+1, \sigma_{zero}\rangle \to \sigma_{zero}[\mathrm{v} \mapsto 1]}$$

Derivation Trees

- Rule instances can be combined to derive a transition $\langle s, \sigma \rangle \to \sigma'$
- The result is a derivation tree
  - The root is the transition $\langle s, \sigma \rangle \to \sigma'$
  - The leaves are axiom instances
  - The internal nodes are conclusions of rule instances and have the corresponding premises as immediate children
  - The conditions of all instantiated rules must be satisfied
- $\langle s, \sigma \rangle \to \sigma'$ is a transition in the transition system if and only if there is a finite derivation tree for $\langle s, \sigma \rangle \to \sigma'$
  - There can be several derivations for one transition (non-deterministic semantics)

Derivations: Example

- What is the result of executing statement

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

  in state $\sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0]$ (abbreviated by $[5,7,0]$)?

$$\textsc{Seq}\ \frac{\textsc{Seq}\ \dfrac{\textsc{Ass}\ \dfrac{}{\langle \mathtt{z:=x}, [5,7,0]\rangle \to [5,7,5]} \qquad \textsc{Ass}\ \dfrac{}{\langle \mathtt{x:=y}, [5,7,5]\rangle \to [7,7,5]}}{\langle \mathtt{z:=x;\ x:=y}, [5,7,0]\rangle \to [7,7,5]} \qquad \textsc{Ass}\ \dfrac{}{\langle \mathtt{y:=z}, [7,7,5]\rangle \to [7,5,5]}}{\langle \mathtt{(z:=x;\ x:=y);\ y:=z}, [5,7,0]\rangle \to [7,5,5]}$$

- In the above derivation, we assume some properties of state updates (such as $\sigma[x \mapsto v_x][y \mapsto v_y] = \sigma[y \mapsto v_y][x \mapsto v_x]$ if $x \neq y$), which will be proved in the exercises

Termination

- For an IMP statement s we define termination in the context of natural semantics as follows

- The execution of a statement s in state σ
  - terminates successfully iff there is a state σ' such that $\langle s, \sigma \rangle \to \sigma'$
  - loops iff there is no state σ' such that $\langle s, \sigma \rangle \to \sigma'$

2. Operational Semantics

Semantic Equivalence

- Definition

  Two statements s_1 and s_2 are semantically equivalent (denoted by $s_1 \equiv s_2$) if the following property holds for all states σ, σ':

  $$\langle s_1, \sigma \rangle \to \sigma' \Leftrightarrow \langle s_2, \sigma \rangle \to \sigma'$$

  - In definitions, lemmas, and proofs, we use a transition $\langle s, \sigma \rangle \to \sigma'$ as a predicate that expresses there is a derivation tree for this transition
  - Here: there is a derivation tree for $\langle s_1, \sigma \rangle \to \sigma'$ if and only if there is a derivation tree for $\langle s_2, \sigma \rangle \to \sigma'$

- Example

  `while` b `do` s `end` $\equiv$
  `if` b `then` s`;` `while` b `do` s `end end`

Unfolding Loops in C, C++, and Java

```
int i = 0;
while(i < 2 ) {


  while(i < 1)
    if(i == 0) break;

  i = i + 1;
}

printf("i = %d", i);
```

i = 2

```
int i = 0;
while(i < 2 ) {
  if(i < 1) {
    if(i == 0) break;
    while(i < 1)
      if(i == 0) break;
  }
  i = i + 1;
}

printf("i = %d", i);
```

i = 0

- Equivalence does not hold in these languages

Unfolding Loops in IMP

- We prove the equivalence based on the natural semantics

$$\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma \rangle \to \sigma'' \Leftrightarrow \qquad (*)$$
$$\langle \texttt{if } b \texttt{ then } s\texttt{; while } b \texttt{ do } s \texttt{ end end}, \sigma \rangle \to \sigma'' \qquad (**)$$

- Proof idea
  - Consider the derivation tree for one transition
  - Show that there is a derivation tree for the other transition

Proof: Case "⇒"

- Consider the derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''$
- The last rule application is one of the rules for `while`
- For the case WHT_{NS}

$$\text{WHT}_{NS}\ \frac{\langle s, \sigma\rangle \to \sigma' \qquad \langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma'\rangle \to \sigma''}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

  we know

  1. There is a derivation tree T_1 with root $\langle s, \sigma\rangle \to \sigma'$
  2. There is a derivation tree T_2 with root $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma'\rangle \to \sigma''$
  3. $\mathcal{B}[\![b]\!]\sigma = tt$

Proof: Case "⇒" (cont'd)

- We can construct the derivation tree

$$\textsc{Seq}_{NS}\,\frac{T_1 \qquad T_2}{\langle s;\texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''}$$

- Since $\mathcal{B}[\![b]\!]\sigma = tt$ we can use the rule for `if` to derive

$$\textsc{IfT}_{NS}\,\frac{\textsc{Seq}_{NS}\ \dfrac{T_1 \qquad T_2}{\langle s;\texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma''}}{\langle \texttt{if}\ b\ \texttt{then}\ s;\texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \texttt{else}\ \texttt{skip}\ \texttt{end}, \sigma\rangle \to \sigma''}$$

- We have a derivation tree for $(**)$, which completes this case

Proof: Case "⇒" (cont'd)

- For the case $\textsc{WhF}_{NS}$

$$\textsc{WhF}_{NS}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \to \sigma} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = ff$$

  we know

  1. $\sigma = \sigma''$
  2. $\mathcal{B}[\![b]\!]\sigma = ff$

- We can construct the derivation tree

$$\textsc{IfF}_{NS}\ \frac{\textsc{Skip}_{NS}\ \dfrac{}{\langle \texttt{skip}, \sigma\rangle \to \sigma''}}{\langle \texttt{if}\ b\ \texttt{then}\ s\texttt{;while}\ b\ \texttt{do}\ s\ \texttt{end else skip end}, \sigma\rangle \to \sigma''}$$

- We have a derivation tree for $(**)$, which completes Case "⇒"
- Case "⇐" will be discussed in the exercises

Deterministic Semantics

Lemma: The natural semantics of IMP is deterministic

- We prove

$$\langle s, \sigma \rangle \to \sigma' \wedge \langle s, \sigma \rangle \to \sigma'' \quad \Rightarrow \quad \sigma' = \sigma''$$

Proof Attempt: Structural Induction

- Reminder: abstract syntax of statements

```
data Stm = Skip
         | Assign String Aexp
         | Seq Stm Stm
         | If Bexp Stm Stm
         | While Bexp Stm
```

- Structural induction for statements

$$\frac{\begin{array}{c}\Gamma \vdash P(Skip) \quad \Gamma \vdash P(Assign\ x\ e) \\ \Gamma, P(s_1), P(s_2) \vdash P(Seq\ s_1\ s_2) \\ \Gamma, P(s_1), P(s_2) \vdash P(If\ b\ s_1\ s_2) \\ \Gamma, P(s) \vdash P(While\ b\ s)\end{array}}{\Gamma \vdash \forall s \in Stm.P(s)} \quad s \text{ not free in } \Gamma$$

Proof Attempt: Structural Induction (2)

- Case $s \equiv \texttt{skip}$
  - We know there is a derivation tree for $\langle \texttt{skip}, \sigma \rangle \to \sigma'$. The only tree with this consequence is an instantiation of the skip-axiom. Thus, we have $\sigma = \sigma'$
  - Analogously, from $\langle \texttt{skip}, \sigma \rangle \to \sigma''$, we get $\sigma = \sigma''$

- Case $s \equiv x := e$
  - Analogous

Proof Attempt: Structural Induction (3)

- Case $s \equiv \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$:
  - There is a derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma'$
  - There are two possibilities to derive this transition, depending on $\mathcal{B}[\![b]\!]\sigma$.
  - The case for $\mathcal{B}[\![b]\!]\sigma = ff$ is analogous to the case for `skip`
  - In the case for $\mathcal{B}[\![b]\!]\sigma = tt$, we conclude that there are derivation trees for $\langle s', \sigma\rangle \to \sigma_1$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$
  - Analogously, we derive from $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma''$ that there are derivation trees for $\langle s', \sigma\rangle \to \sigma_2$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_2\rangle \to \sigma''$
  - s' is a proper sub-statement of s. Therefore, we can apply the induction hypothesis to conclude from $\langle s', \sigma\rangle \to \sigma_1$ and $\langle s', \sigma\rangle \to \sigma_2$ that $\sigma_1 = \sigma_2$
  - It remains to show that $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma''$ imply $\sigma' = \sigma''$

  $\texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$ is obviously not a proper sub-statement of s!
  - So we cannot apply the induction hypothesis
  - The proof is stuck because the remaining proof goal is identical to the initial lemma
- Structural induction does not work since the transition relation is not defined inductively over the structure of the statements

Induction on Derivation Trees

- Induction on the shape of derivation trees
  To prove a property $P(t)$ for all derivation trees t, prove that $P(t)$ holds for an arbitrary derivation tree t under the assumption that $P(t')$ holds for all proper sub-trees t' of t

- Induction on derivations is a special case of well-founded (Noetherian) induction (derivation trees are finite)

- Proofs by induction on the shape of derivation trees typically proceed by case distinction on the rule applied at the root of the arbitrary derivation tree t. In each case, one may assume that:
  - the condition of the rule is satisfied
  - there is a derivation tree t' for each premise of t
  - $P(t')$ holds since t' is a proper sub-tree of t

New Proof Attempt: Induction on Shape of Derivation Tree

- We prove

$$\langle s, \sigma \rangle \to \sigma' \wedge \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma''$$

  by induction on the shape of the derivation tree for $\langle s, \sigma \rangle \to \sigma'$

- We perform a case distinction on the rule applied at the root of the derivation tree for $\langle s, \sigma \rangle \to \sigma'$, which yields seven cases for the seven rules of the natural semantics

- We could also do an induction on the shape of the derivation tree for $\langle s, \sigma \rangle \to \sigma''$

New Proof Attempt: Induction on Shape of Derivation Tree (2)

- Case $\textsc{Skip}_{NS}$: From the form of the rule, we know:
  - $s = \texttt{skip}$
  - $\sigma' = \sigma$

  Therefore, the derivation of $\langle s, \sigma \rangle \to \sigma''$ is actually a derivation of $\langle \texttt{skip}, \sigma \rangle \to \sigma''$. The last rule applied in this derivation must also have been the $\textsc{Skip}_{NS}$ rule, from which we obtain $\sigma'' = \sigma$
- Case $\textsc{Ass}_{NS}$: From the form of the rule, we know:
  - $s = x\texttt{:=}e$ for some x and e
  - $\sigma' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$

  Therefore, the derivation of $\langle s, \sigma \rangle \to \sigma''$ is actually a derivation of $\langle x\texttt{:=}e, \sigma \rangle \to \sigma''$. The last rule applied in this derivation must also have been the $\textsc{Ass}_{NS}$ rule, from which we obtain $\sigma'' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$
- Case $\textsc{WhF}_{NS}$: Analogously

New Proof Attempt: Induction on Shape of Derivation Tree (3)

- Case SEQ_{NS}: From the form of the rule, we know:
  - $s = s_1 ; s_2$ for some s_1 and s_2
  - There are derivation trees for $\langle s_1, \sigma \rangle \to \sigma_0$ and $\langle s_2, \sigma_0 \rangle \to \sigma'$ for some state σ_0

  Analogously to the previous cases, we can conclude from $\langle s, \sigma \rangle \to \sigma''$ that there are derivation trees for $\langle s_1, \sigma \rangle \to \sigma_1$ and $\langle s_2, \sigma_1 \rangle \to \sigma''$ for some state σ_1

  The derivation tree for $\langle s_1, \sigma \rangle \to \sigma_0$ is a proper sub-tree of the tree for $\langle s_1 ; s_2, \sigma \rangle \to \sigma'$. Therefore, we can apply the induction hypothesis to $\langle s_1, \sigma \rangle \to \sigma_0$ (with $\langle s_1, \sigma \rangle \to \sigma_1$) to obtain $\sigma_0 = \sigma_1$. By this equality, we conclude that there is a derivation tree for $\langle s_2, \sigma_0 \rangle \to \sigma''$

  Analogously, we can apply the induction hypothesis to $\langle s_2, \sigma_0 \rangle \to \sigma'$ (with $\langle s_2, \sigma_0 \rangle \to \sigma''$) to obtain $\sigma' = \sigma''$

New Proof Attempt: Induction on Shape of Derivation Tree (4)

- Case IFT_{NS}: From the form of the rule, we know:
  - $s = \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}$ for some b, s_1, and s_2
  - $\mathcal{B}[\![b]\!]\sigma = tt$
  - There is a derivation tree for $\langle s_1, \sigma\rangle \to \sigma'$

  Therefore, the last rule applied in the derivation tree for $\langle s, \sigma\rangle \to \sigma''$ must also have been the IFT_{NS} rule

  Consequently, there is a derivation tree for $\langle s_1, \sigma\rangle \to \sigma''$

  The derivation tree for $\langle s_1, \sigma\rangle \to \sigma'$ is a proper sub-tree of the tree for $\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}, \sigma\rangle \to \sigma'$. Therefore, we can apply the induction hypothesis to $\langle s_1, \sigma\rangle \to \sigma'$ (with $\langle s_1, \sigma\rangle \to \sigma''$) to obtain $\sigma' = \sigma''$.

- Case IFF_{NS}: Analogously

New Proof Attempt: Induction on Shape of Derivation Tree (5)

- Case WHT_{NS}: From the form of the rule, we know:
  - $s = \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$ for some b and s'
  - $\mathcal{B}[\![b]\!]\sigma = tt$
  - There are derivation trees for $\langle s', \sigma\rangle \to \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \to \sigma'$ for some state σ_0

  Therefore, the last rule applied in the derivation tree for $\langle s, \sigma\rangle \to \sigma''$ must also have been the WHT_{NS} rule Consequently, there are derivation trees for $\langle s', \sigma\rangle \to \sigma_1$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma''$ for some state σ_1

  The derivation tree for $\langle s', \sigma\rangle \to \sigma_0$ is a proper sub-tree of the derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma'$. Therefore, we can apply the induction hypothesis to $\langle s', \sigma\rangle \to \sigma_0$ (with $\langle s', \sigma\rangle \to \sigma_1$) to obtain $\sigma_0 = \sigma_1$. By this equality, we conclude that there is a derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \to \sigma''$

  Analogously, we can apply the induction hypothesis to $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \to \sigma'$ (with $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \to \sigma''$) to obtain $\sigma' = \sigma''$

2. Operational Semantics

Local Variable Declarations

- Statement `var` x`:=`e `in` s `end` declares a new variable that is visible in the sub-statement of the declaration, s
- Semantics
  - Expression e is evaluated in the initial state
  - Statement s is executed in a state in which x has the value of e
  - After the execution of s, the initial value of x is restored
- Rule

$$\text{LOC}_{NS}\ \frac{\langle s, \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]\rangle \to \sigma'}{\langle \texttt{var } x\texttt{:=}e\texttt{ in } s\texttt{ end}, \sigma\rangle \to \sigma'[x \mapsto \sigma(x)]}$$

Procedure Declarations and Calls

`procedure` $p(x_1, \ldots, x_n;\ y_1, \ldots, y_m)$ `begin` s `end`

- Formal parameters
  - $x_1, \ldots, x_n$ are value parameters (call-by-value)
  - $y_1, \ldots, y_m$ are variable parameters (call-by-name)
- Rules of the static semantics
  - The variables $x_1, \ldots, x_n, y_1, \ldots, y_m$ are distinct from each other
  - $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ are the only free variables in s (no global variables; local variables declared in s are permitted since they are bound)
  - For calls $p(e_1, \ldots, e_n; z_1, \ldots, z_m)$, the actual variable parameters z_k have to be pairwise disjoint (no aliasing)

Procedures: Example

```
procedure fac(n; res)
begin
  if n <= 1 then
    res := 1
  else
    fac( n-1; res );
    res := n * res
  end
end
```

Natural Semantics of Procedure Calls

- Procedure call $p(e_1, \ldots, e_n; z_1, \ldots, z_m)$ with declaration $\texttt{procedure}\ p(x_1, \ldots, x_n; y_1, \ldots, y_m)\ \texttt{begin}\ s\ \texttt{end}$
  - The call-by-value arguments $e_1, \ldots, e_n$ are evaluated in the initial state to values $v_1, \ldots, v_n$
  - The body of the procedure, s, is executed in a state σ_{init}, in which the value parameters are initialized with the values $v_1, \ldots, v_n$, and the variable parameters are initialized with the values of $z_1, \ldots, z_m$ in the initial state
  - After termination of p, execution continues in the initial state with the values of $y_1, \ldots, y_m$ assigned to the variables $z_1, \ldots, z_m$

$$\textsc{Call}_{NS}\ \frac{\langle s, \sigma_{init}\rangle \to \sigma'}{\langle p(e_1, \ldots, e_n; z_1, \ldots, z_m), \sigma\rangle \to \sigma[z_1 \mapsto \sigma'(y_1)] \ldots [z_m \mapsto \sigma'(y_m)]}$$

where

$$\sigma_{init} = \sigma_{zero}[x_1 \mapsto \mathcal{A}[\![e_1]\!]\sigma] \ldots [x_n \mapsto \mathcal{A}[\![e_n]\!]\sigma][y_1 \mapsto \sigma(z_1)] \ldots [y_m \mapsto \sigma(z_m)]$$

Abortion

- Statement `abort` stops the execution of the complete program
- Abortion is modeled in the operational semantics by ensuring that the configurations $\langle \texttt{abort}, \sigma \rangle$ are stuck, that is, that there is no state σ' such that $\langle \texttt{abort}, \sigma \rangle \rightarrow \sigma'$
- There is no additional rule for `abort` in the natural semantics

Abortion: Observations

- `abort` and `skip` are not semantically equivalent since there is a derivation tree for $\langle \texttt{skip}, \sigma \rangle \to \sigma$, but not for $\langle \texttt{abort}, \sigma \rangle \to \sigma'$
- `abort` and `while true do skip end` are semantically equivalent!
- Natural semantics cannot distinguish between looping and abnormal termination
  - Natural semantics is only concerned with programs that terminate normally
  - Abortion could be modeled by "normal termination" in a special error configuration

Non-determinism

- For the statement $s_1 \,[\!]\, s_2$ either s_1 or s_2 is non-deterministically chosen to be executed

- The statement

  $$\texttt{x:=1} \,[\!]\, (\texttt{x:=2; x:=x+2})$$

  will result in a state in which x either has the value 1 or 4

- Rules

$$\text{ND1}_{NS} \frac{\langle s_1, \sigma \rangle \to \sigma'}{\langle s_1 \,[\!]\, s_2, \sigma \rangle \to \sigma'} \qquad \text{ND2}_{NS} \frac{\langle s_2, \sigma \rangle \to \sigma'}{\langle s_1 \,[\!]\, s_2, \sigma \rangle \to \sigma'}$$

Non-determinism: Observations

- There are derivation trees for
  - $\langle \texttt{x:=1} \,[]\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \to \sigma[\texttt{x} \mapsto 1]$ and
  - $\langle \texttt{x:=1} \,[]\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \to \sigma[\texttt{x} \mapsto 4]$

- There is a derivation tree for

  $$\langle \texttt{while true do skip end} \,[]\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \to \sigma[\texttt{x} \mapsto 4]$$

- A natural semantics always chooses the "right" branch of a non-deterministic choice
  - In a natural semantics non-determinism will suppress looping, if possible
  - That is, natural semantics cannot describe the two possible outcomes of executing a non-deterministic choice in which one sub-statement terminates and the other one does not

Termination Revisited

- For an IMP statement s we define termination in the context of natural semantics as follows

- The execution of a statement s in state σ
  - terminates successfully iff there is a state σ' such that $\langle s, \sigma \rangle \to \sigma'$
  - loops iff there is no state σ' such that $\langle s, \sigma \rangle \to \sigma'$

- According to these definitions
  - `while true do skip end` $\square$ `skip` terminates
  - `abort` loops

  but we cannot give a more precise definition in natural semantics

Parallelism

- For the statement s_1 `par` s_2 both statements s_1 and s_2 are executed, but execution can be interleaved

- The statement

```
x:=1 par (x:=2; x:=x+2)
```

  could result in a state in which `x` has the value 4, 1, or 3

  - Execute `x:=1`, then `x:=2`, and then `x:=x+2`
  - Execute `x:=2`, then `x:=x+2`, and then `x:=1`
  - Execute `x:=2`, then `x:=1`, and then `x:=x+2`

Parallelism: Observations

- Attempt to define rules

$$\textsc{Par1}_{NS}\ \frac{\langle s_1, \sigma\rangle \to \sigma', \langle s_2, \sigma'\rangle \to \sigma''}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \to \sigma''}$$

$$\textsc{Par2}_{NS}\ \frac{\langle s_2, \sigma\rangle \to \sigma', \langle s_1, \sigma'\rangle \to \sigma''}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \to \sigma''}$$

- Rules do not allow interleaving execution

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

Problems of Natural Semantics

- Properties of looping programs cannot be expressed
- No distinction between abortion and looping
- Non-determinism suppresses looping (if possible)
- Parallelism cannot be modeled
- Definition of equivalence is too coarse
  - All sorting programs are equivalent
  - All looping programs are equivalent

Big-Step and Small-Step Semantics

- Big-step semantics describe how the overall results of the executions are obtained
  - Natural semantics
- Small-step semantics describe how the individual steps of the computations take place
  - Structural operational semantics (SOS)
  - Abstract state machines

2. Operational Semantics

Structural Operational Semantics

- The emphasis is on the individual steps of the execution
  - Execution of assignments
  - Execution of tests
- Describing small steps of the execution allows one to express the order of execution of individual steps
  - Interleaving computations
  - Evaluation order for expressions (not shown in the course)
- Always describing the next small step allows one to express properties of looping programs

Transitions in SOS

- The configurations are the same as for natural semantics
- The transition relation $\rightarrow_1$ can have two forms
- $\langle s, \sigma \rangle \rightarrow_1 \langle s', \sigma' \rangle$: the execution of s from σ is not completed and the remaining computation is expressed by the intermediate configuration $\langle s', \sigma' \rangle$
- $\langle s, \sigma \rangle \rightarrow_1 \sigma'$: the execution of s from σ has terminated and the final state is σ'
- A transition $\langle s, \sigma \rangle \rightarrow_1 \gamma$ describes the first step of the execution of s from σ
  - We use γ as meta-variable for (terminal or non-terminal) configurations

Transition System

$$\begin{aligned}
&\Gamma = \{\langle s, \sigma\rangle \mid s \in \mathsf{Stm}, \sigma \in \mathsf{State}\} \cup \mathsf{State}\\
&T = \mathsf{State}\\
&\rightarrow_1 \subseteq \{\langle s, \sigma\rangle \mid s \in \mathsf{Stm}, \sigma \in \mathsf{State}\} \times \Gamma
\end{aligned}$$

- We say that $\langle s, \sigma\rangle$ is stuck if there is no γ such that $\langle s, \sigma\rangle \rightarrow_1 \gamma$

SOS of IMP

- `skip` does not modify the state

$$\textsc{Skip}_{SOS}\ \frac{}{\langle \texttt{skip}, \sigma\rangle \to_1 \sigma}$$

- x:=e assigns the value of e to variable x

$$\textsc{Ass}_{SOS}\ \frac{}{\langle x\texttt{:=}e, \sigma\rangle \to_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]}$$

- `skip` and assignment require only one step
- Rules are analogous to natural semantics

$$\textsc{Skip}_{NS}\ \frac{}{\langle \texttt{skip}, \sigma\rangle \to \sigma}$$

$$\textsc{Ass}_{NS}\ \frac{}{\langle x\texttt{:=}e, \sigma\rangle \to \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]}$$

SOS of IMP: Sequential Composition

- Sequential composition $s_1 ; s_2$
- First step of executing $s_1 ; s_2$ is the first step of executing s_1
- s_1 is executed in one step

$$\textsc{Seq1}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \sigma'}{\langle s_1 ; s_2, \sigma\rangle \to_1 \langle s_2, \sigma'\rangle}$$

- s_1 is executed in several steps

$$\textsc{Seq2}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle}{\langle s_1 ; s_2, \sigma\rangle \to_1 \langle s_1' ; s_2, \sigma'\rangle}$$

SOS of IMP: Conditional Statement

- The first step of executing `if` b `then` s_1 `else` s_2 `end` is to determine the outcome of the test and thereby which branch to select

$$\textsc{IfT}_{sos}\ \frac{}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\textsc{IfF}_{sos}\ \frac{}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \rightarrow_1 \langle s_2, \sigma\rangle} \quad \text{if} \mathcal{B}[\![b]\!]\sigma = ff$$

Alternative for Conditional Statement

- The first step of executing `if` b `then` s_1 `else` s_2 `end` is the first step of the branch determined by the outcome of the test

$$\textsc{IfT1}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \sigma'}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \to_1 \sigma'} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\textsc{IfT2}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle}{\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

and two similar rules for $\mathcal{B}[\![b]\!]\sigma = ff$

- Alternatives are equivalent for IMP
- Choice is important for languages with parallel execution

SOS of IMP: Loop Statement

- The first step is to unroll the loop

$$\textsc{While}_{SOS}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 \langle \texttt{if}\ b\ \texttt{then}\ s\texttt{;while}\ b\ \texttt{do}\ s\ \texttt{end else skip end}, \sigma\rangle}$$

- Recall that `while` b `do` s `end` and `if` b `then` s`;while` b `do` s `end else skip end` are semantically equivalent in the natural semantics

Alternatives for Loop Statement

- The first step is to decide the outcome of the test and thereby whether to unroll the body of the loop or to terminate

$$\text{WHT}_{sos}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 \langle s; \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle}\quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\text{WHF}_{sos}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 \sigma}\quad \text{if } \mathcal{B}[\![b]\!]\sigma = ff$$

- Or combine with the alternative semantics of the conditional statement
- Alternatives are equivalent for IMP

Derivation Sequences

- A derivation sequence is a sequence $\gamma_0, \gamma_1, \gamma_2, \ldots$, where
  - $\gamma_i \rightarrow_1 \gamma_{i+1}$ for each $0 \leq i$ such that $i+1$ is in the range of the sequence
  - if the derivation sequence is finite then it ends with a configuration that is either a terminal configuration or a stuck configuration
- Notation
  - $\gamma_0 \rightarrow_1^i \gamma_i$ indicates that there are i steps in the execution from γ_0 to γ_i
  - $\gamma \rightarrow_1^0 \gamma'$ indicates that there are zero steps in the execution from γ to γ', that is, $\gamma = \gamma'$
  - $\gamma_0 \rightarrow_1^* \gamma_i$ indicates that there is a finite number of steps (possibly 0) in the execution from γ_0 to γ_i
  - We use these notations for derivation sequences as well as for sub-sequences of derivation sequences (for instance, for the first i transitions of a derivation sequence)

Derivation Sequences: Example

- What is the final state if statement

  $$\boxed{\texttt{(z:=x; x:=y); y:=z}}$$

  is executed in state $\sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0]$?

$$\begin{aligned}
&\langle \texttt{(z:=x; x:=y); y:=z}, \sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0]\rangle \\
&\rightarrow_1 \langle \texttt{x:=y; y:=z}, \sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0][\texttt{z} \mapsto 5]\rangle \\
&\rightarrow_1 \langle \texttt{y:=z}, \sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0][\texttt{z} \mapsto 5][\texttt{x} \mapsto 7]\rangle \\
&\rightarrow_1 \sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0][\texttt{z} \mapsto 5][\texttt{x} \mapsto 7][\texttt{y} \mapsto 5] \\
&= \sigma_{zero}[\texttt{x} \mapsto 7][\texttt{y} \mapsto 5][\texttt{z} \mapsto 5]
\end{aligned}$$

- The last step is justified by properties of state updates, which will be proved in the exercises

Derivation Trees

- Derivation trees explain why transitions take place
- For the first step

$$\langle (\texttt{z:=x;}\ \ \texttt{x:=y});\ \ \texttt{y:=z}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=y;}\ \ \texttt{y:=z}, \sigma[\texttt{z} \mapsto 5]\rangle$$

  where $\sigma = \sigma_{zero}[\texttt{x} \mapsto 5][\texttt{y} \mapsto 7][\texttt{z} \mapsto 0]$

  the derivation tree is

$$\textsc{Seq2}_{SOS}\ \frac{\textsc{Seq1}_{SOS}\ \dfrac{\textsc{Ass}_{SOS}\ \dfrac{}{\langle \texttt{z:=x}, \sigma\rangle \rightarrow_1 \sigma[\texttt{z} \mapsto 5]}}{\langle \texttt{z:=x;}\ \ \texttt{x:=y}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=y}, \sigma[\texttt{z} \mapsto 5]\rangle}}{\langle (\texttt{z:=x;}\ \ \texttt{x:=y});\ \ \texttt{y:=z}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=y;}\ \ \texttt{y:=z}, \sigma[\texttt{z} \mapsto 5]\rangle}$$

- The first transition for $\texttt{z:=x;}\ \ (\texttt{x:=y;}\ \ \texttt{y:=z})$ has a simpler tree with only two rule applications ($\textsc{Ass}_{SOS}$ and $\textsc{Seq1}_{SOS}$)

Derivation Sequences and Trees

- Natural (big-step) semantics
  - The execution of a statement (sequence) is described by one big transition
  - The big transition can be seen as trivial derivation sequence with exactly one transition
  - The derivation tree explains why this transition takes place

- Structural operational (small-step) semantics
  - The execution of a statement (sequence) is described by one or more transitions
  - Derivation sequences explain how a statement is executed
  - Derivation trees justify each individual step in a derivation sequence

Termination

- The execution of a statement s in state σ
  - terminates iff there is a finite derivation sequence starting with $\langle s, \sigma \rangle$
  - loops iff there is an infinite derivation sequence starting with $\langle s, \sigma \rangle$

- The execution of a statement s in state σ
  - terminates successfully if $\langle s, \sigma \rangle \rightarrow_1^* \sigma'$
  - In IMP, an execution terminates successfully iff it terminates (no stuck configurations)

- We will see later that non-deterministic statements may terminate and loop in a state σ

2. Operational Semantics

Induction on Derivations

- The length of a derivation sequence $\gamma_0, \gamma_1, \ldots$ is the number of transitions $\gamma_i \rightarrow_1 \gamma_{i+1}$ such that i and $i+1$ are in the range of the sequence
  - A finite derivation sequence $\gamma_0, \gamma_1, \ldots, \gamma_n$ has length n
  - A stuck or terminal configuration γ is a derivation sequence of length zero
- Induction on the length of derivation sequences
  - Induction hypothesis: Assume that the property holds for all derivation sequences of length less than k
  - Prove that it also holds for derivation sequences of length k
- Induction on the length of derivation sequences is an application of strong mathematical (Noetherian) induction.

Using Induction on Derivations

- The proof is often done by inspecting either
  - the structure of the statement or
  - the derivation tree validating the first transition of the derivation sequence
- Lemma

$$\begin{array}{l} \langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma'' \Rightarrow \\ \qquad \exists \sigma', k_1, k_2 : \quad \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge \\ \qquad\qquad\qquad\qquad\quad k_1 + k_2 = k \end{array}$$

  - If there is a derivation sequence $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma''$ with k steps then there are derivation sequences $\langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma'$ and $\langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma''$ such that their numbers of steps add up to k

Proof

- Proof by induction on k, that is, by induction on the length of the derivation sequence for $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma''$
- We assume that the lemma holds for $m < k$
- We prove that the lemma holds for k
- We may assume $k > 0$ since there is no step $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^0 \sigma''$
- The derivation sequence

  $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma''$ can be written as

  $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^{k-1} \sigma''$ for some configuration γ

Proof (cont'd)

- $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^{k-1} \sigma''$
- Consider the two rules that may justify the transition $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma$
- Case 1

$$\textsc{Seq1}_{sos} \frac{\langle s_1, \sigma \rangle \rightarrow_1 \sigma'}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle}$$

- Case 2

$$\textsc{Seq2}_{sos} \frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_1' ; s_2, \sigma' \rangle}$$

Proof: Case 1

- From the form of the rule $\textsc{Seq1}_{SOS}$ we obtain $\langle s_1, \sigma \rangle \rightarrow_1 \sigma'$ and $\gamma = \langle s_2, \sigma' \rangle$
- From $\gamma \rightarrow_1^{k-1} \sigma''$ we get $\langle s_2, \sigma' \rangle \rightarrow_1^{k-1} \sigma''$
- The required result follows by choosing $k_1 = 1$ and $k_2 = k - 1$

Proof: Case 2

- From the form of the rule $\textsc{Seq2}_{SOS}$ we obtain $\langle s_1, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle$ and $\gamma = \langle s_1'; s_2, \sigma'\rangle$
- From $\gamma \to_1^{k-1} \sigma''$ we get $\langle s_1'; s_2, \sigma'\rangle \to_1^{k-1} \sigma''$
- By applying the induction hypothesis, we get
  $\exists \sigma_0, l_1, l_2 : \langle s_1', \sigma'\rangle \to_1^{l_1} \sigma_0 \wedge \langle s_2, \sigma_0\rangle \to_1^{l_2} \sigma'' \wedge l_1 + l_2 = k - 1$
- From

  $\langle s_1, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle$ and $\langle s_1', \sigma'\rangle \to_1^{l_1} \sigma_0$

  we get $\langle s_1, \sigma\rangle \to_1^{l_1+1} \sigma_0$
- By

  $\langle s_2, \sigma_0\rangle \to_1^{l_2} \sigma''$ and $(l_1 + 1) + l_2 = k$

  we have proved the required result

Semantic Equivalence

Two statements s_1 and s_2 are semantically equivalent if for all states σ:

- $\langle s_1, \sigma \rangle \rightarrow_1^* \gamma$ iff $\langle s_2, \sigma \rangle \rightarrow_1^* \gamma$, whenever γ is a configuration that is either stuck or terminal, and
- there is an infinite derivation sequence starting in $\langle s_1, \sigma \rangle$ iff there is one starting in $\langle s_2, \sigma \rangle$

- Note: In the first case, the length of the two derivation sequences may be different

Determinism

Lemma: The structural operational semantics of IMP is deterministic. That is, for all s,σ,γ, and γ' we have that

$$\langle s, \sigma \rangle \rightarrow_1 \gamma \wedge \langle s, \sigma \rangle \rightarrow_1 \gamma' \Rightarrow \gamma = \gamma'$$

- The proof runs by induction on the shape of the derivation tree for the transition $\langle s, \sigma \rangle \rightarrow_1 \gamma$

Corollary: There is exactly one derivation sequence starting in configuration $\langle s, \sigma \rangle$

- The proof runs by induction on the length of the derivation sequence

2. Operational Semantics

Local Variable Declarations

- Local variable declaration $\texttt{var}\ x\texttt{:=}e\ \texttt{in}\ s\ \texttt{end}$
- The small steps are
  1. Assign e to x
  2. Execute s
  3. Restore the initial value of x
     (necessary if x exists in the enclosing scope)
- The first small step is trivial

$$\frac{}{\langle \texttt{var}\ x\texttt{:=}e\ \texttt{in}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 \langle s, \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]\rangle}$$

- But: when s terminates, how should we restore the initial value of x?
  - How do we recognize the termination of s?
  - How do we preserve the original value of x?

Artificial End Marker

- We extend the syntactic category Stm with a restore statement

$$\text{Stm} = \ldots \mid \texttt{'restore'}\ (\text{Var},\text{Val})$$

  - Note that the restore statement contains a value, not an expression
  - The restore statement is used internally by the semantics but must not occur in programs.

- Now we can use the restore statement to mark the end of the scope of a local variable and remember its original value:

$$\textsc{Loc}_{sos}\ \frac{}{\langle \texttt{var}\ x\texttt{:=}e\ \texttt{in}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 \langle s\texttt{;restore}\ (x, \sigma(x)), \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]\rangle}$$

$$\textsc{Ret}_{sos}\ \frac{}{\langle \texttt{restore}\ (var, val), \sigma\rangle \rightarrow_1 \sigma[var \mapsto val]}$$

- A more general solution is to model execution stacks
  - Stacks are useful to handle procedure calls

Abortion

- Statement `abort` stops the execution of the complete program
- Abortion is modeled by ensuring that the configurations $\langle \texttt{abort}, \sigma \rangle$ are stuck
- There is no additional rule for `abort` in the structural operational semantics
- `abort` and `skip` are not semantically equivalent
  - $\langle \texttt{abort}, \sigma \rangle$ is the only derivation sequence for `abort` starting in σ
  - $\langle \texttt{skip}, \sigma \rangle \rightarrow_1 \sigma$ is the only derivation sequence for `skip` starting in σ
  - `skip` terminates successfully in all states, whereas `abort` terminates in all states, but not successfully

Abortion: Observations

- `abort` and `while true do skip end` are not semantically equivalent:

$$\langle \texttt{while true do skip end}, \sigma \rangle \rightarrow_1$$
$$\langle \texttt{if true then skip;while true do skip end end}, \sigma \rangle \rightarrow_1$$
$$\langle \texttt{skip;while true do skip end}, \sigma \rangle \rightarrow_1$$
$$\langle \texttt{while true do skip end}, \sigma \rangle$$

- In a structural operational semantics,
  - looping is reflected by infinite derivation sequences
  - abnormal termination by finite derivation sequences ending in a stuck configuration

Non-determinism

- For the statement $s_1 \,[\!]\, s_2$ either s_1 or s_2 is non-deterministically chosen to be executed

- The statement

  $$\texttt{x:=1} \,[\!]\, (\texttt{x:=2; x:=x+2})$$

  will result in a state in which x either has the value 1 or 4

- Rules

$$\text{ND1}_{sos}\ \frac{}{\langle s_1 \,[\!]\, s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle} \qquad \text{ND2}_{sos}\ \frac{}{\langle s_1 \,[\!]\, s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma\rangle}$$

Non-determinism: Observations

- There are two derivation sequences
  - $\langle \texttt{x:=1} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \rightarrow_1^* \sigma[\texttt{x} \mapsto 1]$
  - $\langle \texttt{x:=1} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \rightarrow_1^* \sigma[\texttt{x} \mapsto 4]$
- There are also two derivation sequences for $\langle \texttt{while true do skip end} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle$
  - a finite derivation sequence leading to $\sigma[\texttt{x} \mapsto 4]$
  - an infinite derivation sequence
- A structural operational semantics can choose the "wrong" branch of a non-deterministic choice
- In a structural operational semantics non-determinism does not suppress looping

Parallelism

- For the statement s_1 par s_2 both statements s_1 and s_2 are executed, but execution can be interleaved

$$\textsc{Par1}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \langle s_1', \sigma'\rangle}{\langle s_1 \text{ par } s_2, \sigma\rangle \to_1 \langle s_1' \text{ par } s_2, \sigma'\rangle}$$

$$\textsc{Par2}_{sos}\ \frac{\langle s_1, \sigma\rangle \to_1 \sigma'}{\langle s_1 \text{ par } s_2, \sigma\rangle \to_1 \langle s_2, \sigma'\rangle}$$

$$\textsc{Par3}_{sos}\ \frac{\langle s_2, \sigma\rangle \to_1 \langle s_2', \sigma'\rangle}{\langle s_1 \text{ par } s_2, \sigma\rangle \to_1 \langle s_1 \text{ par } s_2', \sigma'\rangle}$$

$$\textsc{Par4}_{sos}\ \frac{\langle s_2, \sigma\rangle \to_1 \sigma'}{\langle s_1 \text{ par } s_2, \sigma\rangle \to_1 \langle s_1, \sigma'\rangle}$$

Example: Interleaving

- The statement

  ```
  x:=1 par (x:=2; x:=x+2)
  ```

  will result in a state in which `x` has the value 4, 1, or 3
  - Execute `x:=1`, then `x:=2`, and then `x:=x+2`
  - Execute `x:=2`, then `x:=x+2`, and then `x:=1`
  - Execute `x:=2`, then `x:=1`, and then `x:=x+2`

- In a structural operational semantics we can easily express interleaving of computations

Example: Derivation Sequences

$$\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\to_1 \langle \texttt{x:=2; x:=x+2}, \sigma[x \mapsto 1]\rangle \\
&\to_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\to_1 \sigma[x \mapsto 4]
\end{aligned}$$

$$\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\to_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\to_1 \langle \texttt{x:=1}, \sigma[x \mapsto 4]\rangle \\
&\to_1 \sigma[x \mapsto 1]
\end{aligned}$$

$$\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\to_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\to_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 1]\rangle \\
&\to_1 \sigma[x \mapsto 3]
\end{aligned}$$

Comparison: Summary

Natural Semantics

- Local variable declarations and procedures can be modeled easily
- No distinction between abortion and looping
- Non-determinism suppresses looping (if possible)
- Parallelism cannot be modeled

Structural Operational Semantics

- Local variable declarations and procedures require an explicit encoding of the original state
- Distinction between abortion and looping
- Non-determinism does not suppress looping
- Parallelism can be modeled

2. Operational Semantics

Semantic Functions

- The meaning of statements can be expressed as a partial function from State to State:

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

$$\mathcal{S}_{NS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma\rangle \to \sigma' \text{ for some } \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

$$\mathcal{S}_{SOS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma\rangle \to_1^* \sigma' \text{ for some } \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- The semantic functions are well-defined because the semantics are deterministic

Equivalence Theorem

Theorem: For every statement s of IMP we have
$$\mathcal{S}_{NS}[\![s]\!] = \mathcal{S}_{SOS}[\![s]\!]$$

- If the execution of s from some state terminates in one of the semantics then it also terminates in the other and the resulting states will be equal
- If the execution of s from some state loops in one of the semantics then it will also loop in the other

Equivalence Lemma 1

Lemma: For every statement s of IMP and states σ and σ' we have $\langle s, \sigma \rangle \to \sigma' \Rightarrow \langle s, \sigma \rangle \to_1^* \sigma'$

- If the execution of s from σ terminates in the natural semantics then it will terminate in the same state in the structural operational semantics
- The proof runs by induction on the shape of the derivation tree for $\langle s, \sigma \rangle \to \sigma'$ (see exercise)

Equivalence Lemma 2

Lemma: For every statement s of IMP, states σ and σ', and natural number k we have that $\langle s, \sigma \rangle \rightarrow_1^k \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow \sigma'$

- If the execution of s from σ terminates in the structural operational semantics then it will terminate in the same state in the natural semantics
- The proof runs by induction on the length of the derivation sequence for $\langle s, \sigma \rangle \rightarrow_1^k \sigma'$ (see exercise)

Equivalence Theorem: Proof

$$\mathcal{S}_{NS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma\rangle \to \sigma' \text{ for some } \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{S}_{SOS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma\rangle \to_1^* \sigma' \text{ for some } \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- We have proved: $\mathcal{S}_{NS}[\![s]\!]\sigma = \sigma' \Leftrightarrow \mathcal{S}_{SOS}[\![s]\!]\sigma = \sigma'$
- This is sufficient to prove $\mathcal{S}_{NS}[\![s]\!] = \mathcal{S}_{SOS}[\![s]\!]$ because one function is defined iff the other is defined

Equivalence: Summary

- The natural semantics and structural operational semantics are equivalent
  - Proof of Lemma 1 runs by induction on the shape of the derivation tree
  - Proof of Lemma 2 runs by induction on the length of the derivation sequence

- For extended languages, different formalization of the equivalence theorem could be necessary
  - Non-deterministic languages
  - Consider only finite derivation sequences that end in terminal configurations