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
```

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

- 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

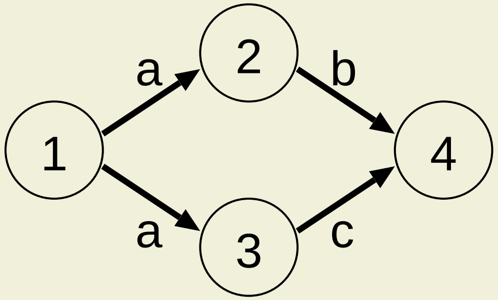
- Big-step semantics describe how the overall results of the executions are obtained
  - Natural semantics
- Small-step semantics describe how the individual steps of the computations take place
  - Structural operational semantics
  - Abstract state machines

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: $\langle \texttt{skip}, \sigma \rangle \rightarrow \sigma$

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

Rules

- Transition relation is specified by rules

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

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 premises
  - ψ is called conclusion
  - A rule without premises is called axiom

Notation

- Updating States: $\sigma[y \mapsto v]$ is the function that
  - overrides the association of y in σ by $y \mapsto v$ or
  - adds the new association $y \mapsto v$ to σ

$$(\sigma[y \mapsto v])(x) = \begin{cases} v & \text{if } x = y \\ \sigma(x) & \text{if } x \neq y \end{cases}$$

Natural Semantics of IMP

- skip does not modify the state

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

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

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

- 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'

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

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

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

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

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'

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

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

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

Rule Instantiations

- Rules are actually rule schemes
  - Meta-variables stand for arbitrary variables, expressions, statements, states, etc.
  - To apply rules, they have to be instantiated by selecting particular variables, expressions, statements, states, etc.

- Assignment rule scheme

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

- Assignment rule instance

$$\textsc{Ass}_{NS}\ \frac{}{\langle \mathtt{v := v+1}, \{\mathtt{v} \mapsto 3\}\rangle \rightarrow \{\mathtt{v} \mapsto 4\}}$$

Derivation Trees

- Rule instances can be combined to derive a transition $\langle s, \sigma \rangle \rightarrow \sigma'$

- The result is a derivation tree
    - The root is the transition $\langle s, \sigma \rangle \rightarrow \sigma'$
    - 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

- $\langle s, \sigma \rangle \rightarrow \sigma'$ is a transition in the transition system if and only if there is a finite derivation tree for $\langle s, \sigma \rangle \rightarrow \sigma'$
    - There can be several derivations for one transition (non-deterministic semantics)

Derivations: Example

- What is the final state if statement

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

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

$$\textsc{Seq}\ \dfrac{\textsc{Seq}\ \dfrac{\textsc{Ass}\ \dfrac{}{\langle \texttt{z:=x}, [5,7,0]\rangle \to [5,7,5]} \quad \textsc{Ass}\ \dfrac{}{\langle \texttt{x:=y}, [5,7,5]\rangle \to [7,7,5]}}{\langle \texttt{z:=x; x:=y}, [5,7,0]\rangle \to [7,7,5]} \quad \textsc{Ass}\ \dfrac{}{\langle \texttt{y:=z}, [7,7,5]\rangle \to [7,5,5]}}{\langle \texttt{z:=x; x:=y; y:=z}, [5,7,0]\rangle \to [7,5,5]}$$

Termination

- The execution of a statement s in state σ
  - terminates iff there is a state σ' such that $\langle s, \sigma \rangle \rightarrow \sigma'$
  - loops iff there is no state σ' such that $\langle s, \sigma \rangle \rightarrow \sigma'$
- A statement s
  - always terminates if the execution in a state σ terminates for all choices of σ
  - always loops if the execution in a state σ loops for all choices of σ

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':
  $$\langle s_1, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \langle s_2, \sigma \rangle \rightarrow \sigma'$$

  - *There is a derivation tree for $\langle s_1, \sigma \rangle \rightarrow \sigma'$ if and only if there is a derivation tree for $\langle s_2, \sigma \rangle \rightarrow \sigma'$*

- 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

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

- 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 $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma \rangle \rightarrow \sigma''$
- The last rule application is one of the rules for `while`
- For the case $\textsc{WhT}_{NS}$

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

we know

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

Proof: Case "$\Rightarrow$" (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

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

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

Proof: Case "⇒" (cont'd)

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

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

  we know

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

- We can construct the derivation tree

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

- 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

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

Proof Attempt: Structural Induction

$$
\begin{aligned}
\text{Stm} \quad &= \text{'}\texttt{skip}\text{'} \\
&\mid \text{Var '}\texttt{:=}\text{' Aexp} \\
&\mid \text{Stm '}\texttt{;}\text{' Stm} \\
&\mid \text{'}\texttt{if}\text{' Bexp '}\texttt{then}\text{' Stm '}\texttt{else}\text{' Stm '}\texttt{end}\text{'} \\
&\mid \text{'}\texttt{while}\text{' Bexp '}\texttt{do}\text{' Stm '}\texttt{end}\text{'}
\end{aligned}
$$

- We try to prove the lemma by structural induction on the statement s
  - Induction base: `skip` and assignment
  - Induction step: sequential composition, `if`, and `while`

- 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 assignment: analogous

Proof Attempt: Structural Induction (2)

- 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 $\mathcal{B}[\![b]\!]\sigma = \mathit{ff}$ is analogous to the case for `skip`
  - In the case for $\mathcal{B}[\![b]\!]\sigma = \mathit{tt}$, 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 definition of the transition relation is not inductive

Induction on Derivation Trees

- Induction on the shape of derivation trees
  1. Induction base: Prove that the property holds for all the simple derivation trees by showing that it holds for the axioms of the transition system
  2. Induction step: Prove that the property holds for all composite derivation trees:
     - Induction hypothesis: For each rule, assume that the property holds for its premises
     - Prove that it also holds for the conclusion, provided that the conditions of the rule are satisfied

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

New Proof Attempt: Induction on Shape of Derivation Tree

- We prove
$$\langle s, \sigma \rangle \rightarrow \sigma' \wedge \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma''$$
by induction on the shape of the derivation tree for $\langle s, \sigma \rangle \rightarrow \sigma'$
  - Induction base: axioms of natural semantics
    $\textsc{Skip}_{NS}$, $\textsc{Ass}_{NS}$, $\textsc{WhF}_{NS}$
  - Induction step: rules of natural semantics
    $\textsc{Seq}_{NS}$, $\textsc{IfT}_{NS}$, $\textsc{IfF}_{NS}$, $\textsc{WhT}_{NS}$

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

Induction Base

- Case $\textsc{Skip}_{NS}$: The derivation tree is the axiom instance
  $\textsc{Skip}_{NS}\,\dfrac{}{\langle \texttt{skip}, \sigma\rangle \to \sigma'}$ and we know:
  - $\sigma' = \sigma$
  - The only axiom or rule that gives $\langle \texttt{skip}, \sigma\rangle \to \sigma''$ is the skip-axiom, which implies, $\sigma'' = \sigma$

- Case $\textsc{Ass}_{NS}$: The derivation tree is the axiom instance
  $\textsc{Ass}_{NS}\,\dfrac{}{\langle x\texttt{:=}e, \sigma\rangle \to \sigma'}$ and we know:
  - $\sigma' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$
  - The only axiom or rule that gives $\langle x\texttt{:=}e, \sigma\rangle \to \sigma''$ is the assign-axiom, which implies, $\sigma'' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$

- Case $\textsc{WhF}_{NS}$: Analogously

Induction Step: Seq. Composition

- Case $\textsc{Seq}_{NS}$: The root of the derivation tree is $\langle s_1\,;s_2, \sigma\rangle \to \sigma'$.
    - 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
    - The only rule that gives $\langle s_1\,;s_2, \sigma\rangle \to \sigma''$ is the sequence-rule. Therefore, 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
    - By the induction hypothesis, $\langle s_1, \sigma\rangle \to \sigma_0$ and $\langle s_1, \sigma\rangle \to \sigma_1$ imply $\sigma_0 = \sigma_1$
    - By the induction hypothesis, $\langle s_2, \sigma_0\rangle \to \sigma'$ and $\langle s_2, \sigma_1\rangle \to \sigma''$ imply $\sigma' = \sigma''$

Induction Step: `if`

- Case IfT_{NS}: The root of the derivation tree is
  $\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}, \sigma\rangle \rightarrow \sigma'$
  - There is a derivation tree for $\langle s_1, \sigma\rangle \rightarrow \sigma'$
  - The only rule that gives $\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}, \sigma\rangle \rightarrow \sigma''$ is the if-rule. Since $\mathcal{B}[\![b]\!]\sigma = tt$, there is a derivation tree for $\langle s_1, \sigma\rangle \rightarrow \sigma''$
  - By the induction hypothesis, $\langle s_1, \sigma\rangle \rightarrow \sigma'$ and $\langle s_1, \sigma\rangle \rightarrow \sigma''$ imply $\sigma' = \sigma''$
- Case IfF_{NS}: Analogously

Induction Step: `while`

- Case WHT_{NS}: The root of the derivation tree is $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \rightarrow \sigma'$
  - There are derivation trees for $\langle s, \sigma\rangle \rightarrow \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \rightarrow \sigma'$ for some state σ_0
  - The derivation trees for $\langle s, \sigma\rangle \rightarrow \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \rightarrow \sigma'$ are proper sub-trees of the derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \rightarrow \sigma'$. Thus, we can apply the induction hypothesis
  - The only rule that gives $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \rightarrow \sigma''$ is the while-rule. Since $\mathcal{B}[\![b]\!]\sigma = tt$, there are derivation trees for $\langle s, \sigma\rangle \rightarrow \sigma_1$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \rightarrow \sigma''$ for some state σ_1
  - By the induction hypothesis, $\langle s, \sigma\rangle \rightarrow \sigma_0$ and $\langle s, \sigma\rangle \rightarrow \sigma_1$ imply $\sigma_0 = \sigma_1$
  - By the induction hypothesis, $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \rightarrow \sigma'$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \rightarrow \sigma''$ imply $\sigma' = \sigma''$

2. Operational Semantics

Local Variable Declarations

- Statement `var` x`:=`e `in` s `end` declares a new variable that is visible in the statement sequence of the declaration, s (block)
- 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

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

Procedure Declarations and Calls

$$\boxed{\texttt{procedure } p(x_1 \ldots x_n;\ y_1 \ldots y_m) \texttt{ begin } s \texttt{ end}}$$

- Formal parameters
  - $x_1 \ldots x_n$ are value parameters (call-by-value)
  - $y_1 \ldots y_m$ are variable parameters (call-by-name)
- Context conditions
  - The variables x_j and y_k are pairwise disjoint
  - $x_1 \ldots x_n$ and $y_1 \ldots y_m$ are the only free variables in s (no global variables)
  - For calls $p(e_1 \ldots e_n; y_1 \ldots y_m)$, the actual variable parameters y_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
```

Vector Notation

- To simplify notations for procedures, we write $\vec{x}$ for $x_1, x_2, \ldots, x_m (m \geq 0)$ and $\vec{e}$ for $e_1, e_2, \ldots, e_n (n \geq 0)$
- For state updates, we write $\sigma[\vec{y} \mapsto \vec{v}]$ for $\sigma[y_1 \mapsto v_1][y_2 \mapsto v_2] \ldots [y_n \mapsto v_n]$

Natural Semantics of Procedure Calls

- Procedure call $p(\vec{e}; \vec{z})$ with declaration
  `procedure` $p(\vec{x}; \vec{y})$ `begin` s `end`
  - The call-by-value arguments $\vec{e}$ are evaluated in the initial state to values $\vec{v}$
  - The body of the procedure, s, is executed in a new state in which the value parameters are initialized by the values $\vec{v}$, and the variable parameters are initialized by the values of $\vec{z}$ in the initial state
  - After termination of p, execution continues in the initial state with the values of $\vec{y}$ assigned to the variables $\vec{z}$

$$\textsc{Call}_{NS}\ \frac{\langle s, \{\vec{x} \mapsto \mathcal{A}[\![\vec{e}]\!]\sigma, \vec{y} \mapsto \sigma(\vec{z})\}\rangle \to \sigma'}{\langle p(\vec{e}; \vec{z}), \sigma\rangle \to \sigma[\vec{z} \mapsto \sigma'(\vec{y})]}$$

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
- 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 $\langle \texttt{skip}, \sigma \rangle \rightarrow \sigma$, but not for $\langle \texttt{abort}, \sigma \rangle \rightarrow \sigma'$
- `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 $s_1 \,\Box\, s_2$ either s_1 or s_2 is non-deterministically chosen to be executed
- The statement

  $$\boxed{\texttt{x:=1} \,\Box\, (\texttt{x:=2; x:=x+2})}$$

  could result in a state in which `x` has the value 1 or 4
- Rules

$$\mathrm{ND1}_{NS} \frac{\langle s_1, \sigma \rangle \to \sigma'}{\langle s_1 \,\Box\, s_2, \sigma \rangle \to \sigma'} \qquad \mathrm{ND2}_{NS} \frac{\langle s_2, \sigma \rangle \to \sigma'}{\langle s_1 \,\Box\, s_2, \sigma \rangle \to \sigma'}$$

Non-determinism: Observations

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

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

- A natural semantics always chooses the "right" branch of a non-deterministic choice
- In a natural semantics non-determinism will suppress looping, if possible

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

  $$\boxed{\texttt{x:=1 par (x:=2; x:=x+2)}}$$

  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)
- Describing always 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

Transition System

$$\Gamma = \{\langle s, \sigma\rangle \mid s \in \text{Stm}, \sigma \in \text{State}\} \cup \text{State}$$
$$T = \text{State}$$
$$\rightarrow_1 \subseteq \{\langle s, \sigma\rangle \mid s \in \text{Stm}, \sigma \in \text{State}\} \times \Gamma$$

- 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

$$\textsc{Skip}_{SOS}\ \frac{}{\langle \texttt{skip}, \sigma\rangle \rightarrow_1 \sigma}$$

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

$$\textsc{Ass}_{SOS}\ \frac{}{\langle x\texttt{:=}e, \sigma\rangle \rightarrow_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]}$$

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

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

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

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

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

- s_1 is executed in several steps

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

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

$$\textsc{IfT1}_{SOS}\ \frac{\langle s_1, \sigma \rangle \rightarrow_1 \sigma'}{\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}, \sigma \rangle \rightarrow_1 \sigma'} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\textsc{IfT2}_{SOS}\ \frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle}{\langle \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle} \quad \text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

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

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

- 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

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

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

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

Derivation Sequences

- A derivation sequence of a statement s starting in state σ 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
- A derivation sequence is either finite or infinite
  - Finite derivation sequences end 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
  - $\gamma_0 \rightarrow_1^* \gamma_i$ indicates that there is a finite number of steps in the execution from γ_0 to γ_i

Derivation Sequences: Example

- What is the final state if statement

```
z:=x; x:=y; y:=z
```

is executed in state $\{x \mapsto 5, y \mapsto 7, z \mapsto 0\}$?

$$
\begin{aligned}
&\langle \texttt{z:=x; x:=y; y:=z}, \{x \mapsto 5, y \mapsto 7, z \mapsto 0\}\rangle \\
\rightarrow_1\ &\langle \texttt{x:=y; y:=z}, \{x \mapsto 5, y \mapsto 7, z \mapsto 5\}\rangle \\
\rightarrow_1\ &\langle \texttt{y:=z}, \{x \mapsto 7, y \mapsto 7, z \mapsto 5\}\rangle \\
\rightarrow_1\ &\{x \mapsto 7, y \mapsto 5, z \mapsto 5\}
\end{aligned}
$$

Derivation Trees

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

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

  the derivation tree is

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

- `z:=x; ( x:=y; y:=z )` would lead to a simpler tree with only one rule application

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 are important
  - 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)

2. Operational Semantics

2.1 Big-Step Semantics

2.2 Small-Step Semantics

- 2.2.1 Structural Operational Semantics of IMP
- 2.2.2 Properties of the Semantics
- 2.2.3 Extensions of IMP

2.3 Equivalence

Induction on Derivations

- Induction on the length of derivation sequences
  - Induction hypothesis: Assume that the property holds for all derivation sequences of length at most k
  - Prove that it also holds for derivation sequences of length $k + 1$
- Induction on the length of derivation sequences is an application of strong mathematical induction.

Using Induction on Derivations

- The induction step is often done by inspecting either
  - the structure of the syntactic element or
  - the derivation tree validating the first transition of the derivation sequence

- Lemma

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

  - *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 $k+1$ of the derivation sequence for $\langle s_1\,;s_2,\sigma\rangle \rightarrow_1^k \sigma''$
- We assume that the lemma holds for $k \leq m-1$
- We prove that the lemma holds for m
- We may assume $m > 0$ since there is no step $\langle s_1\,;s_2,\sigma\rangle \rightarrow_1^0 \sigma''$
- The derivation sequence
  $\langle s_1\,;s_2,\sigma\rangle \rightarrow_1^m \sigma''$ can be written as
  $\langle s_1\,;s_2,\sigma\rangle \rightarrow_1 \gamma \rightarrow_1^{m-1} \sigma''$ for some configuration γ

Proof (cont'd)

- $\langle s_1\,;s_2,\sigma\rangle \rightarrow_1 \gamma \rightarrow_1^{m-1} \sigma''$
- Consider the two rules that could lead to 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
  $\langle s_1\,; s_2, \sigma\rangle \rightarrow_1 \gamma \rightarrow_1^{m-1} \sigma''$ and $\langle s_1\,; s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma'\rangle$
  we conclude $\langle s_2, \sigma'\rangle \rightarrow_1^{m-1} \sigma''$
- The required result follows by choosing $k_1 = 1$ and $k_2 = m - 1$

Proof: Case 2

- From
  $\langle s_1\,; s_2, \sigma\rangle \rightarrow_1 \gamma \rightarrow_1^{m-1} \sigma''$ and $\langle s_1\,; s_2, \sigma\rangle \rightarrow_1 \langle s_1'\,; s_2, \sigma'\rangle$
  we conclude $\langle s_1'\,; s_2, \sigma'\rangle \rightarrow_1^{m-1} \sigma''$
- By applying the induction hypothesis, we get
  $\exists \sigma_0, l_1, l_2 : \langle s_1', \sigma'\rangle \rightarrow_1^{l_1} \sigma_0 \wedge \langle s_2, \sigma_0\rangle \rightarrow_1^{l_2} \sigma'' \wedge l_1 + l_2 = m - 1$
- From
  $\langle s_1, \sigma\rangle \rightarrow_1 \langle s_1', \sigma'\rangle$ and $\langle s_1', \sigma'\rangle \rightarrow_1^{l_1} \sigma_0$
  we get $\langle s_1, \sigma\rangle \rightarrow_1^{l_1+1} \sigma_0$
- By
  $\langle s_2, \sigma_0\rangle \rightarrow_1^{l_2} \sigma''$ and $(l_1 + 1) + l_2 = m$
  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 return statement

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

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

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

$$\textsc{Loc}_{SOS}\ \frac{}{\langle \texttt{var}\ x \texttt{:=} e\ \texttt{in}\ s\ \texttt{end}, \sigma\rangle \to_1 \langle s\texttt{;return}\ (x, \sigma(x)), \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]\rangle}$$

$$\textsc{Ret}_{SOS}\ \frac{}{\langle \texttt{return}\ (\mathit{var}, \mathit{val}), \sigma\rangle \to_1 \sigma[\mathit{var} \mapsto \mathit{val}]}$$

- 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

Abortion: Observations

- $\texttt{abort}$ and $\texttt{while true do skip end}$ are not semantically equivalent:

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

- 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 $s_1 \square s_2$ either s_1 or s_2 is non-deterministically chosen to be executed
- The statement

$$\boxed{\texttt{x:=1} \square (\texttt{x:=2; x:=x+2})}$$

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

- Rules

$$\text{ND1}_{SOS} \frac{}{\langle s_1 \square s_2, \sigma \rangle \rightarrow_1 \langle s_1, \sigma \rangle} \qquad \text{ND2}_{SOS} \frac{}{\langle s_1 \square s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma \rangle}$$

Non-determinism: Observations

- There are two derivation sequences
  - $\langle \texttt{x:=1} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \rightarrow_1^* \sigma[\mathrm{x} \mapsto 1]$
  - $\langle \texttt{x:=1} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \rightarrow_1^* \sigma[\mathrm{x} \mapsto 4]$
- 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 $\sigma[\mathrm{x} \mapsto 4]$
  - 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

$$\textsc{Par1}_{sos}\ \frac{\langle s_1, \sigma\rangle \rightarrow_1 \langle s_1', \sigma'\rangle}{\langle s_1 \ \texttt{par}\ s_2, \sigma\rangle \rightarrow_1 \langle s_1' \ \texttt{par}\ s_2, \sigma'\rangle}$$

$$\textsc{Par2}_{sos}\ \frac{\langle s_1, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \ \texttt{par}\ s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma'\rangle}$$

$$\textsc{Par3}_{sos}\ \frac{\langle s_2, \sigma\rangle \rightarrow_1 \langle s_2', \sigma'\rangle}{\langle s_1 \ \texttt{par}\ s_2, \sigma\rangle \rightarrow_1 \langle s_1 \ \texttt{par}\ s_2', \sigma'\rangle}$$

$$\textsc{Par4}_{sos}\ \frac{\langle s_2, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \ \texttt{par}\ s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma'\rangle}$$

Example: Interleaving

- The statement

  $$\boxed{\texttt{x:=1 par (x:=2; x:=x+2)}}$$

  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

$$
\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\rightarrow_1 \langle \texttt{x:=2; x:=x+2}, \sigma[x \mapsto 1]\rangle \\
&\rightarrow_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\rightarrow_1 \sigma[x \mapsto 4]
\end{aligned}
$$

$$
\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\rightarrow_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\rightarrow_1 \langle \texttt{x:=1}, \sigma[x \mapsto 4]\rangle \\
&\rightarrow_1 \sigma[x \mapsto 1]
\end{aligned}
$$

$$
\begin{aligned}
\langle \texttt{x:=1 par (x:=2; x:=x+2)}, \sigma\rangle &\rightarrow_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle \\
&\rightarrow_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 1]\rangle \\
&\rightarrow_1 \sigma[x \mapsto 3]
\end{aligned}
$$

Comparison: Summary

Natural Semantics	Structural Operational Semantics
• Local variable declarations and procedures can be modeled easily	• Local variable declarations and procedures require an explicit encoding of the original state
• No distinction between abortion and looping	• Distinction between abortion and looping
• Non-determinism suppresses looping (if possible)	• Non-determinism does not suppress looping
• Parallelism cannot be modeled	• Parallelism can be modeled

2. Operational Semantics

Semantic Functions

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

$$\mathcal{S}_{NS} : \text{Stm} \rightarrow (\text{State} \hookrightarrow \text{State})$$
$$\mathcal{S}_{NS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma \rangle \rightarrow \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{S}_{SOS} : \text{Stm} \rightarrow (\text{State} \hookrightarrow \text{State})$$
$$\mathcal{S}_{SOS}[\![s]\!]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma \rangle \rightarrow_1^* \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- 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 $\langle s, \sigma \rangle \rightarrow \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow_1^* \sigma'$

- 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 $\langle s, \sigma \rangle \rightarrow \sigma'$

Induction Base

- Case $\textsc{Ass}_{NS}$:

  The derivation tree is the axiom instance

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

  From the SOS rule we get

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

- Case $\textsc{Skip}_{NS}$: Analogously

- Case $\textsc{WhF}_{NS}$: Analogously

Induction Step: Seq. Composition

- Case $\textsc{Seq}_{NS}$:
  The root of the derivation tree is $\langle s_1\,;s_2, \sigma\rangle \to \sigma'$.
  - 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
  - By the induction hypothesis, we get $\langle s_1, \sigma\rangle \to_1^* \sigma_0$ and $\langle s_2, \sigma_0\rangle \to_1^* \sigma'$
  - By the result of an exercise, we get $\langle s_1\,;s_2, \sigma\rangle \to_1^* \langle s_2, \sigma_0\rangle$
  - Finally, $\langle s_1\,;s_2, \sigma\rangle \to_1^* \langle s_2, \sigma_0\rangle$ and $\langle s_2, \sigma_0\rangle \to_1^* \sigma'$ imply $\langle s_1\,;s_2, \sigma\rangle \to_1^* \sigma'$

Induction Step: `if`

- Case IfT_{NS}:
  The root of the derivation tree is $\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma \rangle \rightarrow \sigma'$
  - There is a derivation tree for $\langle s_1, \sigma \rangle \rightarrow \sigma'$
  - By $\mathcal{B}[\![b]\!]\sigma = tt$ and the induction hypothesis, we get
    $\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma \rangle \rightarrow_1 \langle s_1, \sigma \rangle \rightarrow_1^* \sigma'$
- Case IfF_{NS}: Analogously

Induction Step: `while`

- Case WhT_{NS}:
  The root of the derivation tree is $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow \sigma'$
  - There are derivation trees for $\langle s, \sigma\rangle \rightarrow \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma_0\rangle \rightarrow \sigma'$ for some state σ_0
  - By the induction hypothesis, we get $\langle s, \sigma\rangle \rightarrow_1^* \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma_0\rangle \rightarrow_1^* \sigma'$
  - We derive:

$$\begin{array}{ll} \langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1 & [\text{While}_{SOS}] \\ \langle \texttt{if}\ b\ \texttt{then}\ s\texttt{;while}\ b\ \texttt{do}\ s\ \texttt{end}\ \texttt{end}, \sigma\rangle \rightarrow_1 & [\text{IfT}_{SOS}] \\ \langle s\texttt{;while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma\rangle \rightarrow_1^* & [\text{Exercise}] \\ \langle \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}, \sigma_0\rangle \rightarrow_1^* \sigma' & \end{array}$$

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 k, that is, by induction on the length $k+1$ of the derivation sequence for $\langle s, \sigma \rangle \rightarrow_1^k \sigma'$

Equivalence Lemma 2: Proof

- Assume that lemma holds for $k \leq m - 1$
- Prove that lemma holds for m: $\langle s, \sigma \rangle \rightarrow_1^m \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow \sigma'$
- We may assume $m > 0$ since there is no step $\langle s, \sigma \rangle \rightarrow_1^0 \sigma'$
- We consider the first step of the derivation sequence
  $\langle s, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^{m-1} \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow \sigma'$
- We inspect the derivation tree for the first step $\langle s, \sigma \rangle \rightarrow_1 \gamma$

Equivalence Lemma 2: Proof (cont'd)

- Case $\textsc{Ass}_{SOS}$
  - We have $\langle x := e, \sigma \rangle \rightarrow_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$
  - In this case $\gamma = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma] = \sigma'$ is a state and $m - 1 = 0$
  - By $\textsc{Ass}_{NS}$, we get $\langle s, \sigma \rangle \rightarrow \sigma'$
- Case $\textsc{Skip}_{SOS}$: Analogously
- Case Sequential composition
  - We have $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^m \sigma'$
  - There are a state σ'' and numbers k_1, k_2 such that $\langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma''$ and $\langle s_2, \sigma'' \rangle \rightarrow_1^{k_2} \sigma'$ where $k_1 + k_2 = m$
  - We have $k_1, k_2 > 0$ and thus $k_1, k_2 < m$
  - By the induction hypothesis, we get $\langle s_1, \sigma \rangle \rightarrow \sigma''$ and $\langle s_2, \sigma'' \rangle \rightarrow \sigma'$
  - By $\textsc{Seq}_{NS}$, we get $\langle s_1 ; s_2, \sigma \rangle \rightarrow \sigma'$

Equivalence Lemma 2: Proof (cont'd)

- Case $\textsc{IfT}_{SOS}$
  - We have $\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle \rightarrow_1^{m-1} \sigma'$
  - By the induction hypothesis, we get $\langle s_1, \sigma\rangle \rightarrow \sigma'$
  - By $\textsc{IfT}_{NS}$, we get $\langle \texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ end}, \sigma\rangle \rightarrow \sigma'$
- Case $\textsc{IfF}_{SOS}$: Analogously
- Case $\textsc{While}_{SOS}$
  - We have
    $\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \rightarrow_1 \langle \texttt{if } b \texttt{ then } s\texttt{;while } b \texttt{ do } s \texttt{ end end}, \sigma\rangle \rightarrow_1^{m-1} \sigma'$
  - By the induction hypothesis, we get
    $\langle \texttt{if } b \texttt{ then } s\texttt{;while } b \texttt{ do } s \texttt{ end end}, \sigma\rangle \rightarrow \sigma'$
  - By the lemma about unfolding loops, we get $\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \rightarrow \sigma'$

Equivalence Theorem: Proof

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

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

- 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