Semantics of Programming Languages

Operational Semantics

Prof. Peter Müller

Software Component Technology

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

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

- 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

$$\langle \texttt{skip}, \sigma \rangle \rightarrow_1 \sigma$$

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

$$\langle x := e, \sigma \rangle \rightarrow_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$$

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

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

$$\langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$$

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

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \sigma'}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle}$$

- s_1 is executed in several steps

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_1' ; s_2, \sigma' \rangle}$$

SOS of IMP: Conditional Statement

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

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

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

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Alternative for Conditional Statement

- The first step of executing $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}$ is the first step of the branch determined by the outcome of the test

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

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

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

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

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SOS of IMP: Loop Statement

- The first step is to unrole the loop

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

- Recall that $\texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}$ and $\texttt{if}\ b\ \texttt{then}\ s\texttt{;while}\ b\ \texttt{do}\ s\ \texttt{end}\ \texttt{else}\ \texttt{skip}\ \texttt{end}$ are semantically equivalent in the natural semantics

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Alternatives for Loop Statement

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

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

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

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

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Derivation Sequences

- A **derivation sequence** of a statement s starting in state σ is a sequence $\gamma_0, \gamma_1, \gamma_2, \ldots$, where
  - $\gamma_0 = \langle s, \sigma \rangle$
  - $\gamma_i \rightarrow_1 \gamma_{i+1}$ for $0 \leq i$
- 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
  - $\gamma_0 \rightarrow_1^i \gamma_i$ and $\gamma_0 \rightarrow_1^* \gamma_i$ need **not** be derivation sequences

Derivation Sequences: Example

- What is the final state if statement

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

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

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

Derivation Trees

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

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

the derivation tree is

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

- `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

Induction on Derivations

Induction on the length of derivation sequences

1. **Induction base**: Prove that the property holds for all derivation sequences of length 0
2. **Induction step**: Prove that the property holds for all other 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

$$\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma'' \Rightarrow$$
$$\exists \sigma', k_1, k_2 : \ \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge$$
$$k_1 + k_2 = k$$

Proof

- Proof by induction on k, that is, by induction on the length of the derivation sequence for $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma''$
- Induction base: $k = 0$: There is no derivation sequence of length 0 for $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^k \sigma''$
- Induction step
  - We assume that the lemma holds for $k \leq m$
  - We prove that the lemma holds for $m + 1$
  - The derivation sequence $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^{m+1} \sigma''$ can be written as $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma''$ for some configuration γ

Induction Step

- $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma''$
- Consider the two rules that could lead to the transition $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma$
- Case 1

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \sigma'}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle}$$

- Case 2

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle}{\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_1' ; s_2, \sigma' \rangle}$$

Induction Step: Case 1

- From $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma''$ and $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle$ we conclude $\langle s_2, \sigma' \rangle \rightarrow_1^m \sigma''$
- The required result follows by choosing $k_1 = 1$ and $k_2 = m$

Induction Step: Case 2

- From $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma''$ and $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1 \langle s_1' ; s_2, \sigma' \rangle$ we conclude $\langle s_1' ; s_2, \sigma' \rangle \rightarrow_1^m \sigma''$
- By applying the induction hypothesis, we get $\exists \sigma_0, l_1, l_2 : \langle s_1', \sigma' \rangle \rightarrow_1^{l_1} \sigma_0 \wedge \langle s_2, \sigma_0 \rangle \rightarrow_1^{l_2} \sigma'' \wedge l_1 + l_2 = m$
- 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 $(l_1 + 1) + l_2 = m + 1$ 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 `var` x `:=` e `in` s `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)
- Problem: There is no history of states that could be used to restore the value of x
- Idea: Represent states as execution stacks

Modelling Execution Stacks

- We model execution stacks by providing a mapping Var $\longrightarrow$ Val for each scope

$$\text{State} : \textit{stack of}(\text{Var} \rightarrow \text{Val})$$

- Assignment and lookup have to determine the highest stack element in which a variable is defined
- Example: $\sigma(x) = 3$

$z \longmapsto 4$
$x \longmapsto 3$
$x \longmapsto 1, y \longmapsto 2$

SOS for Variable Declarations

- The small steps are
  1. Create new scope and assign e to x in this scope
  2. Execute s
  3. Restore the initial value of x using a `return` statement

$$\langle \texttt{var } x \texttt{:=} e \texttt{ in } s \texttt{ end}, \sigma\rangle \rightarrow_1 \langle s\texttt{;return}, \textit{push}(\{x \mapsto \mathcal{A}[\![e]\!]\sigma\}, \sigma)\rangle$$

$$\langle \texttt{return}, \sigma\rangle \rightarrow_1 \textit{pop}(\sigma)$$

- Similar techniques can be used for 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 is s
  - $\langle \texttt{skip}, \sigma\rangle \rightarrow_1 \sigma$ is the only derivation sequence for `skip` starting is s

Abortion: Observations

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

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

- 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

$$\boxed{\texttt{x:=1} [] \texttt{x:=2; x:=x+2}}$$

  could result in a state in which $\texttt{x}$ has the value 1 or 4
- Rules

$$\langle s_1 [] s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle \qquad \langle s_1 [] s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma\rangle$$

Non-determinism: Observations

- There are two derivation sequences
  - $\langle \texttt{x:=1} [] \texttt{x:=2; x:=x+2}, \sigma\rangle \rightarrow_1^* \sigma[\texttt{x} \mapsto 1]$
  - $\langle \texttt{x:=1} [] \texttt{x:=2; x:=x+2}, \sigma\rangle \rightarrow_1^* \sigma[\texttt{x} \mapsto 4]$
- There are also two derivation sequences for $\langle \texttt{while true do skip end} [] \texttt{x:=2; x:=x+2}, \sigma\rangle$
  - an 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 $s_1 \texttt{ par } s_2$ both statements s_1 and s_2 are executed, but execution can be **interleaved**

$$\frac{\langle s_1, \sigma\rangle \rightarrow_1 \langle s_1', \sigma'\rangle}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1' \texttt{ par } s_2, \sigma'\rangle}$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma'\rangle}$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow_1 \langle s_2', \sigma'\rangle}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1 \texttt{ par } s_2', \sigma'\rangle}$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \texttt{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma'\rangle}$$

Example: Interleaving

- The statement

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

  could result in a state in which $\texttt{x}$ 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

$$\langle \texttt{x:=1 par x:=2; x:=x+2}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=2; x:=x+2}, \sigma[x \mapsto 1]\rangle$$
$$\rightarrow_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 2]\rangle$$
$$\rightarrow_1 \sigma[x \mapsto 4]$$

$$\langle \texttt{x:=1 par x:=2; x:=x+2}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle$$
$$\rightarrow_1 \langle \texttt{x:=1}, \sigma[x \mapsto 4]\rangle$$
$$\rightarrow_1 \sigma[x \mapsto 1]$$

$$\langle \texttt{x:=1 par x:=2; x:=x+2}, \sigma\rangle \rightarrow_1 \langle \texttt{x:=1 par x:=x+2}, \sigma[x \mapsto 2]\rangle$$
$$\rightarrow_1 \langle \texttt{x:=x+2}, \sigma[x \mapsto 1]\rangle$$
$$\rightarrow_1 \sigma[x \mapsto 3]$$

Comparison: Summary

Natural Semantics	Structural Operational Semantics
▸ Local variable declarations and procedures can be modeled easily	▸ Local variable declarations and procedures require modeling the execution stack
▸ 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