

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

$$x + 2 * y$$
$$\mathcal{A} : \text{Aexp} \rightarrow \text{State} \rightarrow \text{Val}$$

- 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

2.1 Big-Step Semantics

2.1.1 Natural Semantics of IMP

2.1.2 Properties of the Semantics

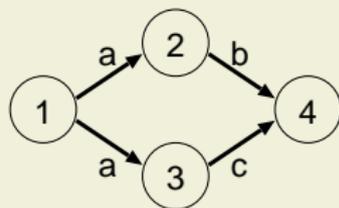
2.1.3 Extensions of IMP

2.2 Small-Step Semantics

2.3 Equivalence

Transition Systems

- A transition system is a tuple (Γ, 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



$$\Gamma = \{ \langle w, S \rangle \mid w \in \{a, b, c\}^*, S \in \{1, 2, 3, 4\} \}$$

$$T = \{ \langle \epsilon, S \rangle \mid S \in \{1, 2, 3, 4\} \}$$

$$\rightarrow = \{ (\langle aw, 1 \rangle \rightarrow \langle w, 2 \rangle), (\langle aw, 1 \rangle \rightarrow \langle w, 3 \rangle), \\ (\langle bw, 2 \rangle \rightarrow \langle w, 4 \rangle), (\langle cw, 3 \rangle \rightarrow \langle w, 4 \rangle) \mid w \in \{a, b, c\}^* \}$$

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

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

Rules

- Transition relation is specified by rules

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

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

- Meaning of the rule

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

- Terminology

- $\varphi_1, \dots, \varphi_n$ 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

$$\overline{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

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

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

- 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'

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

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

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

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

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'

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

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

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

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**

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

- Assignment rule **instance**

$$\langle v := v+1, \{v \mapsto 3\} \rangle \rightarrow \{v \mapsto 4\}$$

Derivations: Example

- What is the final state if statement

$$z:=x; x:=y; y:=z$$

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

$$\frac{\langle z:=x, [5, 7, 0] \rangle \rightarrow [5, 7, 5], \langle x:=y, [5, 7, 5] \rangle \rightarrow [7, 7, 5]}{\langle z:=x; x:=y, [5, 7, 0] \rangle \rightarrow [7, 7, 5], \quad \langle y:=z, [7, 7, 5] \rangle \rightarrow [7, 5, 5]} \\ \langle z:=x; x:=y; y:=z, [5, 7, 0] \rangle \rightarrow [7, 5, 5]$$

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
- There can be several derivations for one transition (non-deterministic semantics)

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

2.1 Big-Step Semantics

2.1.1 Natural Semantics of IMP

2.1.2 Properties of the Semantics

2.1.3 Extensions of IMP

2.2 Small-Step Semantics

2.3 Equivalence

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' :

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

- 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

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

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

Proof: Case “ \Rightarrow ”

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

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

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 $\langle \text{while } b \text{ do } s \text{ end}, \sigma' \rangle \rightarrow \sigma''$
3. $\mathcal{B}[[b]]\sigma = tt$

Proof: Case “ \Rightarrow ” (cont'd)

- We can construct the derivation tree

$$\frac{T_1, T_2}{\langle s; \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma''}$$

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

$$\frac{\frac{T_1, T_2}{\langle s; \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma''}}{\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end else skip end}, \sigma \rangle \rightarrow \sigma''}$$

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

Proof: Case “ \Rightarrow ” (cont'd)

- For the case

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

we know

1. $\sigma = \sigma''$
 2. $\mathcal{B}[[b]]\sigma = ff$
- We can construct the derivation tree

$$\frac{\langle \text{skip}, \sigma \rangle \rightarrow \sigma''}{\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end else skip end}, \sigma \rangle \rightarrow \sigma''}$$

- We have a derivation tree for $(**)$, which completes Case “ \Rightarrow ”
- Case “ \Leftarrow ” will be discussed in the exercises

Induction on Derivations

- 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** (derivations are finite)

Using Induction on Derivations

Lemma: The natural semantics of IMP is deterministic

- We prove

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

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

- Structural induction does not work since definition of transition relation is not inductive

Induction Base

- Case skip-axiom: The derivation tree is the axiom instance $\langle \text{skip}, \sigma \rangle \rightarrow \sigma'$ and we know:
 - $\sigma' = \sigma$
 - The only axiom or rule that gives $\langle \text{skip}, \sigma \rangle \rightarrow \sigma''$ is the skip-axiom, which implies, $\sigma'' = \sigma$
- Case assign-axiom: The derivation tree is the axiom instance $\langle x := e, \sigma \rangle \rightarrow \sigma'$ and we know:
 - $\sigma' = \sigma[x \mapsto \mathcal{A}[[e]]\sigma]$
 - The only axiom or rule that gives $\langle x := e, \sigma \rangle \rightarrow \sigma''$ is the assign-axiom, which implies, $\sigma'' = \sigma[x \mapsto \mathcal{A}[[e]]\sigma]$
- Case while-rule ($\mathcal{B}[[b]]\sigma = ff$): Analogously

Induction Step: Seq. Composition

- Case sequence-rule: The root of the derivation tree is $\langle s_1; s_2, \sigma \rangle \rightarrow \sigma'$.
 - There are derivation trees for $\langle s_1, \sigma \rangle \rightarrow \sigma_0$ and $\langle s_2, \sigma_0 \rangle \rightarrow \sigma'$ for some state σ_0
 - The only rule that gives $\langle s_1; s_2, \sigma \rangle \rightarrow \sigma''$ is the sequence-rule. Therefore, there are derivation trees for $\langle s_1, \sigma \rangle \rightarrow \sigma_1$ and $\langle s_2, \sigma_1 \rangle \rightarrow \sigma''$ for some state σ_1
 - By the induction hypothesis, $\langle s_1, \sigma \rangle \rightarrow \sigma_0$ and $\langle s_1, \sigma \rangle \rightarrow \sigma_1$ imply $\sigma_0 = \sigma_1$
 - By the induction hypothesis, $\langle s_2, \sigma_0 \rangle \rightarrow \sigma'$ and $\langle s_2, \sigma_1 \rangle \rightarrow \sigma''$ imply $\sigma' = \sigma''$

Induction Step: if

- Case if-rule ($\mathcal{B}[[b]]\sigma = tt$): The root of the derivation tree is $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma'$
 - There is a derivation tree for $\langle s_1, \sigma \rangle \rightarrow \sigma'$
 - The only rule that gives $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma''$ 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 if-rule ($\mathcal{B}[[b]]\sigma = ff$): Analogously

Induction Step: while

- Case while-rule ($\mathcal{B}[[b]]\sigma = tt$): The root of the derivation tree is $\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'$
 - There are derivation trees for $\langle s, \sigma \rangle \rightarrow \sigma_0$ and $\langle \text{while } b \text{ do } s \text{ end}, \sigma_0 \rangle \rightarrow \sigma'$ for some state σ_0
 - The only rule that gives $\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma''$ is the while-rule. Since $\mathcal{B}[[b]]\sigma = tt$, there are derivation trees for $\langle s, \sigma \rangle \rightarrow \sigma_1$ and $\langle \text{while } b \text{ do } s \text{ end}, \sigma_1 \rangle \rightarrow \sigma''$ 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, $\langle \text{while } b \text{ do } s \text{ end}, \sigma_0 \rangle \rightarrow \sigma'$ and $\langle \text{while } b \text{ do } s \text{ end}, \sigma_1 \rangle \rightarrow \sigma''$ imply $\sigma' = \sigma''$
- Case while-rule ($\mathcal{B}[[b]]\sigma = ff$): See induction base

2. Operational Semantics

2.1 Big-Step Semantics

2.1.1 Natural Semantics of IMP

2.1.2 Properties of the Semantics

2.1.3 Extensions of IMP

2.2 Small-Step Semantics

2.3 Equivalence

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

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

Procedure Declarations and Calls

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

- Formal parameters
 - $x_1 \dots x_n$ are value parameters (call-by-value)
 - $y_1 \dots y_m$ are variable parameters (call-by-name)
- Context conditions
 - The variables x_j and y_k are pairwise disjoint
 - $x_1 \dots x_n$ and $y_1 \dots y_m$ are the only free variables in s (no global variables)
 - For calls $p(e_1 \dots e_n; y_1 \dots y_m)$, 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, \dots, x_m ($m \geq 0$) and \vec{e} for e_1, e_2, \dots, e_n ($n \geq 0$)
- For state updates, we write $\sigma[\vec{y} \mapsto f(\vec{v})]$ for $\sigma[y_1 \mapsto f(v_1)][y_2 \mapsto f(v_2)] \dots [y_n \mapsto f(v_n)]$

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}

$$\frac{\langle s, \{\vec{x} \mapsto \mathcal{A}[[\vec{e}]]\sigma, \vec{y} \mapsto \sigma(\vec{z})\} \rangle \rightarrow \sigma'}{\langle p(\vec{e}; \vec{z}), \sigma \rangle \rightarrow \sigma[\vec{z} \mapsto \sigma'(\vec{y})]}$$

Abortion

- Statement `abort` stops the execution of the complete program
- Abortion is modeled in the operational semantics by ensuring that the configurations $\langle \text{abort}, \sigma \rangle$ 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 $\langle \text{skip}, \sigma \rangle \rightarrow \sigma$, but not for $\langle \text{abort}, \sigma \rangle \rightarrow \sigma'$
- `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 $s_1 \square s_2$ either s_1 or s_2 is non-deterministically chosen to be executed
- The statement

$$x:=1 \square (x:=2; x:=x+2)$$

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

- Rules

$$\frac{\langle s_1, \sigma \rangle \rightarrow \sigma'}{\langle s_1 \square s_2, \sigma \rangle \rightarrow \sigma'}$$

$$\frac{\langle s_2, \sigma \rangle \rightarrow \sigma'}{\langle s_1 \square s_2, \sigma \rangle \rightarrow \sigma'}$$

Non-determinism: Observations

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

- There is a derivation tree for

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

- 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

$x:=1$ 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

$$\frac{\langle s_1, \sigma \rangle \rightarrow \sigma', \langle s_2, \sigma' \rangle \rightarrow \sigma''}{\langle s_1 \text{ par } s_2, \sigma \rangle \rightarrow \sigma''}$$

$$\frac{\langle s_2, \sigma \rangle \rightarrow \sigma', \langle s_1, \sigma' \rangle \rightarrow \sigma''}{\langle s_1 \text{ par } s_2, \sigma \rangle \rightarrow \sigma''}$$

- 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

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

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

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

- $x := e$ 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

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

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

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 `if b then s1 else s2 end` is to determine the outcome of the test and thereby which branch to select

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

Alternative for Conditional Statement

- The first step of executing `if b then s1 else s2 end` is the first step of the branch determined by the outcome of the test

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

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s'_1, \sigma' \rangle}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle s'_1, \sigma' \rangle} \quad \text{if } \mathcal{B}[[b]]\sigma = tt$$

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 unrole the loop

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

- 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

$$\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow_1 \langle s; \text{while } b \text{ do } s \text{ end}, \sigma \rangle$$

if $\mathcal{B}[[b]]\sigma = tt$

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

- 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 $\gamma_0, \gamma_1, \gamma_2, \dots$, 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

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

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

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

Derivation Trees

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

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

the derivation tree is

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

- $z:=x; (x:=y; y:=z)$ 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
 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

$$\langle s_1; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma'' \text{ and } \langle s_1; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle$$

$$\text{we conclude } \langle s_2, \sigma' \rangle \rightarrow_1^m \sigma''$$

- The required result follows by choosing $k_1 = 1$ and $k_2 = m$

Induction Step: Case 2

- From

$$\langle s_1; s_2, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma'' \text{ and } \langle s_1; s_2, \sigma \rangle \rightarrow_1 \langle s'_1; s_2, \sigma' \rangle$$

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

$$\langle s_1, \sigma \rangle \rightarrow_1 \langle s'_1, \sigma' \rangle \text{ and } \langle s'_1, \sigma' \rangle \rightarrow_1^{l_1} \sigma_0$$

we get $\langle s_1, \sigma \rangle \rightarrow_1^{l_1+1} \sigma_0$

- By

$$\langle s_2, \sigma_0 \rangle \rightarrow_1^{l_2} \sigma'' \text{ and } (l_1 + 1) + l_2 = m + 1$$

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

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

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 $\text{Var} \rightarrow \text{Val}$ for each scope

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

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

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

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

$$\begin{aligned} \langle \text{var } x := e \text{ in } s \text{ end}, \sigma \rangle &\rightarrow_1 \\ \langle s; \text{return}, \text{push}(\{x \mapsto \mathcal{A}[[e]]\sigma\}, \sigma) \rangle \\ \langle \text{return}, \sigma \rangle &\rightarrow_1 \text{pop}(\sigma) \end{aligned}$$

- 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 $\langle \text{abort}, \sigma \rangle$ are stuck
- There is no additional rule for `abort` in the structural operational semantics
- `abort` and `skip` are not semantically equivalent
 - $\langle \text{abort}, \sigma \rangle$ is the only derivation sequence for `abort` starting in σ
 - $\langle \text{skip}, \sigma \rangle \rightarrow_1 \sigma$ is the only derivation sequence for `skip` starting in σ

Abortion: Observations

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

$\langle \text{while true do skip end}, \sigma \rangle \rightarrow_1$

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

$\langle \text{skip; while true do skip end} \rangle \rightarrow_1$

$\langle \text{while true do skip end}, \sigma \rangle$

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

$$x:=1 \square (x:=2; x:=x+2)$$

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

- Rules

$$\langle s_1 \square s_2, \sigma \rangle \rightarrow_1 \langle s_1, \sigma \rangle$$
$$\langle s_1 \square s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma \rangle$$

Non-determinism: Observations

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

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s'_1, \sigma' \rangle}{\langle s_1 \text{ par } s_2, \sigma \rangle \rightarrow_1 \langle s'_1 \text{ par } s_2, \sigma' \rangle}$$

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

$$\frac{\langle s_2, \sigma \rangle \rightarrow_1 \langle s'_2, \sigma' \rangle}{\langle s_1 \text{ par } s_2, \sigma \rangle \rightarrow_1 \langle s_1 \text{ par } s'_2, \sigma' \rangle}$$

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

Example: Interleaving

- The statement

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

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 $x:=x+2$
- In a structural operational semantics we can easily express interleaving of computations

Example: Derivation Sequences

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

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

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

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 modeling the execution stack
- Distinction between abortion and looping
- Non-determinism does not suppress looping
- Parallelism can be modeled

2. Operational Semantics

2.1 Big-Step Semantics

2.2 Small-Step Semantics

2.3 Equivalence

Semantic Functions

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

$$\mathcal{S}_{NS} : \text{Stm} \rightarrow (\text{State} \leftrightarrow \text{State})$$

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

$$\mathcal{S}_{SOS} : \text{Stm} \rightarrow (\text{State} \leftrightarrow \text{State})$$

$$\mathcal{S}_{SOS}[[s]]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma \rangle \rightarrow_1^* \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- 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 assign-axiom:

The derivation tree is the axiom instance

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

From the SOS rule we get

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

- Case skip-axiom: Analogously
- Case while-rule ($\mathcal{B}[[b]]\sigma = ff$): Analogously

Induction Step: Seq. Composition

- Case sequence-rule:

The root of the derivation tree is $\langle s_1; s_2, \sigma \rangle \rightarrow \sigma'$.

- There are derivation trees for $\langle s_1, \sigma \rangle \rightarrow \sigma_0$ and $\langle s_2, \sigma_0 \rangle \rightarrow \sigma'$ for some state σ_0
- By the induction hypothesis, we get $\langle s_1, \sigma \rangle \rightarrow_1^* \sigma_0$ and $\langle s_2, \sigma_0 \rangle \rightarrow_1^* \sigma'$
- By the result of an exercise, we get $\langle s_1; s_2, \sigma \rangle \rightarrow_1^* \langle s_2, \sigma_0 \rangle$
- Finally, $\langle s_1; s_2, \sigma \rangle \rightarrow_1^* \langle s_2, \sigma_0 \rangle$ and $\langle s_2, \sigma_0 \rangle \rightarrow_1^* \sigma'$ imply $\langle s_1; s_2, \sigma \rangle \rightarrow_1^* \sigma'$

Induction Step: `if`

- Case `if`-rule ($\mathcal{B}[[b]]\sigma = tt$):

The root of the derivation tree is $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma'$

- 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 $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle s_1, \sigma \rangle \rightarrow_1^* \sigma'$

- Case `if`-rule ($\mathcal{B}[[b]]\sigma = ff$): Analogously

Induction Step: while

- Case while-rule ($\mathcal{B}[[b]]\sigma = tt$):

The root of the derivation tree is $\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'$

- There are derivation trees for $\langle s, \sigma \rangle \rightarrow \sigma_0$ and $\langle \text{while } b \text{ do } s \text{ end}, \sigma_0 \rangle \rightarrow \sigma'$ for some state σ_0
- By the induction hypothesis, we get $\langle s, \sigma \rangle \rightarrow_1^* \sigma_0$ and $\langle \text{while } b \text{ do } s \text{ end}, \sigma_0 \rangle \rightarrow_1^* \sigma'$
- We derive:

$$\begin{array}{ll} \langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow_1 & [\text{while-rule}] \\ \langle \text{if } b \text{ then } s ; \text{while } b \text{ do } s \text{ end end}, \sigma \rangle \rightarrow_1 & [\mathcal{B}[[b]]\sigma = tt] \\ \langle s ; \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow_1^* & [\text{Exercise}] \\ \langle \text{while } b \text{ do } s \text{ end}, \sigma_0 \rangle \rightarrow_1^* \sigma' & \end{array}$$

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 the length of the derivation sequence for $\langle s, \sigma \rangle \rightarrow_1^k \sigma'$, that is, by induction on k
- Induction base: For $k = 0$, the result holds trivially

Induction Step

- Assume that lemma holds for $k \leq m$
- Prove that lemma holds for $m + 1$: $\langle s, \sigma \rangle \rightarrow_1^{m+1} \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow \sigma'$
- We consider the first step of the derivation sequence
 $\langle s, \sigma \rangle \rightarrow_1 \gamma \rightarrow_1^m \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow \sigma'$
- We inspect the derivation tree for the first step $\langle s, \sigma \rangle \rightarrow_1 \gamma$

Induction Step (cont'd)

- Case assign-axiom
 - 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 $m = 0$
 - From the NS-axiom we get $\langle s, \sigma \rangle \rightarrow \sigma'$
- Case skip-axiom: Analogously
- Case Sequential composition
 - We have $\langle s_1 ; s_2, \sigma \rangle \rightarrow_1^{m+1} \sigma'$
 - 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 $k_1 + k_2 = m + 1$
 - By the induction hypothesis, we get $\langle s_1, \sigma \rangle \rightarrow \sigma''$ and $\langle s_2, \sigma'' \rangle \rightarrow \sigma'$
 - By the NS-rule, we get $\langle s_1 ; s_2, \sigma \rangle \rightarrow \sigma'$

Induction Step (cont'd)

- Case if-axiom ($\mathcal{B}[[b]]\sigma = tt$)
 - We have $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow_1 \langle s_1, \sigma \rangle \rightarrow_1^m \sigma'$
 - By the induction hypothesis, we get $\langle s_1, \sigma \rangle \rightarrow \sigma'$
 - By the NS-rule, we get $\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma'$
- Case if-axiom ($\mathcal{B}[[b]]\sigma = ff$): Analogously
- Case while-axiom
 - We have $\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow_1 \langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end end}, \sigma \rangle \rightarrow_1^m \sigma'$
 - By the induction hypothesis, we get $\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end end}, \sigma \rangle \rightarrow \sigma'$
 - By the lemma about unfolding loops, we get $\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'$

Equivalence Theorem: Proof

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

$$\mathcal{S}_{SOS}[[s]]\sigma = \begin{cases} \sigma' & \text{if } \langle s, \sigma \rangle \rightarrow_1^* \sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

- 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