

# Formal Methods and Functional Programming

## Operational Semantics

Peter Müller

Chair of Programming Methodology  
ETH Zurich

## Big-Step and Small-Step Semantics

- Big-step semantics describe how the overall results of the executions are obtained
  - The system in this course is also called Natural Semantics
- Small-step semantics describe how the individual steps of the computations take place
  - The system in this course is called Structural Operational Semantics
  - Alternative approaches exist, e.g., abstract state machines

## Operational Semantics of Statements

- Recall: 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

ETH zürich

Peter Müller—Formal Methods and Functional Programming, SS14 p. 69

## 3. Operational Semantics

### 3.1 Big-Step Semantics

#### 3.1.1 Natural Semantics of IMP

#### 3.1.2 Proving Properties of the Semantics

#### 3.1.3 Extensions of IMP

### 3.2 Small-Step Semantics

### 3.3 Equivalence

## Transition Systems

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

## Transition Systems

- A transition system is a tuple  $(\Gamma, T, \rightarrow)$ 
  - $\Gamma$ : a set of configurations
  - $T$ : a set of terminal configurations,  $T \subseteq \Gamma$
  - $\rightarrow$ : a transition relation,  $\rightarrow \subseteq \Gamma \times \Gamma$
- Operational semantics includes two types of configurations
  1.  $\langle s, \sigma \rangle$ , which represents that the statement  $s$  is to be executed in state  $\sigma$
  2.  $\sigma$ , which represents a final state (terminal configuration)
- The transition relation  $\rightarrow$  describes how executions take place
  - Big-step transitions are of the form  $\langle s, \sigma \rangle \rightarrow \sigma'$
  - Example:  $\langle \text{skip}, \sigma \rangle \rightarrow \sigma$

## Transition Systems

- A transition system is a tuple  $(\Gamma, T, \rightarrow)$ 
  - $\Gamma$ : a set of configurations
  - $T$ : a set of terminal configurations,  $T \subseteq \Gamma$
  - $\rightarrow$ : a transition relation,  $\rightarrow \subseteq \Gamma \times \Gamma$
- Operational semantics includes two types of configurations
  1.  $\langle s, \sigma \rangle$ , which represents that the statement  $s$  is to be executed in state  $\sigma$
  2.  $\sigma$ , which represents a final state (terminal configuration)

## Transition Systems

- A transition system is a tuple  $(\Gamma, T, \rightarrow)$ 
  - $\Gamma$ : a set of configurations
  - $T$ : a set of terminal configurations,  $T \subseteq \Gamma$
  - $\rightarrow$ : a transition relation,  $\rightarrow \subseteq \Gamma \times \Gamma$
- Operational semantics includes two types of configurations
  1.  $\langle s, \sigma \rangle$ , which represents that the statement  $s$  is to be executed in state  $\sigma$
  2.  $\sigma$ , which represents a final state (terminal configuration)
- The transition relation  $\rightarrow$  describes how executions take place
  - Big-step transitions are of the form  $\langle s, \sigma \rangle \rightarrow \sigma'$
  - Example:  $\langle \text{skip}, \sigma \rangle \rightarrow \sigma$

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

## Inference Rules

- We specify the transition relation by rules of the form

$$\frac{\varphi_1 \quad \dots \quad \varphi_n}{\psi} (Name)^*$$

\* (optional: side-condition)

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

- Meaning of the rule

$$\text{If } \varphi_1, \dots, \varphi_n \text{ are transitions} \\ \text{(and side-condition is true)} \\ \text{then } \psi \text{ is a transition}$$

- Terminology

- $\varphi_1, \dots, \varphi_n$  are called the **premises** of the rule
- $\psi$  is called the **conclusion** of the rule
- A rule without premises is sometimes called an **axiom rule**

## Big-Step Semantics of IMP (cont'd)

- Conditional statement if  $b$  then  $s$  else  $s'$  end

- If  $b$  holds,  $s$  is executed
- If  $b$  does not hold,  $s'$  is executed

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}'}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}'} (\text{IFT}_{NS}) \quad \text{if } \mathcal{B}[[\underline{b}]]\underline{\sigma} = tt$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \rightarrow \underline{\sigma}'}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}'} (\text{IFF}_{NS}) \quad \text{if } \mathcal{B}[[\underline{b}]]\underline{\sigma} = ff$$

## Big-Step Semantics of IMP

- skip does not modify the state

$$\frac{}{\langle \text{skip}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}} (\text{SKIP}_{NS})$$

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

$$\frac{}{\langle \underline{x} := \underline{e}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}[\underline{x} \mapsto \mathcal{A}[[\underline{e}]]\underline{\sigma}]} (\text{ASS}_{NS})$$

- Sequential composition  $s; s'$

- First,  $s$  is executed in state  $\sigma$ , leading to  $\sigma'$
- Then  $s'$  is executed in state  $\sigma'$ , leading to  $\sigma''$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}' \quad \langle \underline{s}', \underline{\sigma}' \rangle \rightarrow \underline{\sigma}''}{\langle \underline{s}; \underline{s}', \underline{\sigma} \rangle \rightarrow \underline{\sigma}''} (\text{SEQ}_{NS})$$

## Big-Step Semantics of IMP (cont'd)

- Loop statement while  $b$  do  $s$  end

- If  $b$  holds,  $s$  is executed once, leading to state  $\sigma'$
- Then the whole while-statement is executed again in  $\sigma'$

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

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

$$\frac{}{\langle \text{while } \underline{b} \text{ do } \underline{s} \text{ end}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}} (\text{WHF}_{NS}) \quad \text{if } \mathcal{B}[[\underline{b}]]\underline{\sigma} = ff$$

## Rule Schemes and Instantiations

- Inference rule definitions are actually **rule schemes**
  - Meta-variables in rule definitions are placeholders for statements, states, etc.
  - A rule scheme describes infinitely many **rule instances**
- A rule is **instantiated** when all meta-variables are replaced with syntactic elements
- By convention, we write meta-variables in **rule schemes** with underlines
- Assignment rule scheme

$$\frac{}{\langle \underline{x} := \underline{e}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}[\underline{x} \mapsto \mathcal{A}[\underline{e}]]\underline{\sigma}} \text{ (ASS}_{NS}\text{)}$$

- Assignment rule instance

$$\frac{}{\langle v := v+1, \sigma_{zero} \rangle \rightarrow \sigma_{zero}[v \mapsto 1]} \text{ (ASS}_{NS}\text{)}$$

## Derivations: Example

- What is the result of executing statement

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

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

$$\frac{\frac{\frac{}{\langle z:=x, [5, 7, 0] \rangle \rightarrow [5, 7, 5]} \text{ (ASS}_{NS}\text{)}}{\langle z:=x; x:=y, [5, 7, 0] \rangle \rightarrow [7, 7, 5]} \text{ (SEQ}_{NS}\text{)}}{\langle (z:=x; x:=y); y:=z, [5, 7, 0] \rangle \rightarrow [7, 5, 5]} \text{ (ASS}_{NS}\text{)}$$

- In the above derivation, we assume some properties of state updates (such as  $\sigma[x \mapsto v_1][y \mapsto v_2] = \sigma[y \mapsto v_2][x \mapsto v_1]$  if  $x \neq y$ ), which will be proved in the exercises

## Derivation Trees

- Rule instances can be combined to derive a transition  $\langle s, \sigma \rangle \rightarrow \sigma'$
- The result is a **derivation tree**  $T$ 
  - The root of  $T$  is  $\langle s, \sigma \rangle \rightarrow \sigma'$ , written as  $root(T) = \langle s, \sigma \rangle \rightarrow \sigma'$
  - The leaves of  $T$  are axiom rule instances
  - The internal nodes of  $T$  are conclusions of rule instances and have the corresponding premises as immediate children
  - The side-conditions of all instantiated rules must be satisfied
- The transition system permits a transition  $\langle s, \sigma \rangle \rightarrow \sigma'$ , written as  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$ , if and only if there exists a **finite** derivation tree ending in  $\langle s, \sigma \rangle \rightarrow \sigma'$ 
  - $\vdash \langle s, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \exists T. root(T) = \langle s, \sigma \rangle \rightarrow \sigma'$

## Termination

- For an IMP statement  $s$  we define termination in the context of big-step semantics as follows
- The execution of a statement  $s$  in state  $\sigma$ 
  - terminates successfully iff there exists a state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
  - fails to terminate iff there is no state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
- For example, while true do skip end fails to terminate

## 3. Operational Semantics

### 3.1 Big-Step Semantics

- 3.1.1 Natural Semantics of IMP
- 3.1.2 Proving Properties of the Semantics
- 3.1.3 Extensions of IMP

### 3.2 Small-Step Semantics

### 3.3 Equivalence

## Semantic Equivalence

- Definition

Two statements  $s_1$  and  $s_2$  are semantically equivalent (written  $s_1 \simeq s_2$ ) if:

$$\forall \sigma, \sigma'. ( \vdash \langle s_1, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \vdash \langle s_2, \sigma \rangle \rightarrow \sigma' )$$

- Examples

```
while false do s end  $\simeq$  skip
```

```
while b do s end  $\simeq$   
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);
```

```
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);
```

## 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 in the big-step semantics for IMP

$$\forall b, s. ( \text{while } b \text{ do } s \text{ end} \simeq \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end end} )$$

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

- Proof idea
  - Prove the equivalence by showing implication in both directions
  - In each direction, consider the derivation tree for one transition
  - Show that there is a derivation tree for the other transition
  - We show only the  $\Rightarrow$  direction of the proof here ( $\Leftarrow$  in exercises)

### Proof: Case “ $\Rightarrow$ ” (cont’d)

- Using  $T_1$  and  $T_2$ , we can construct the derivation tree

$$\frac{\frac{\text{[Diagram } T_1 \text{]} \quad \langle s, \sigma \rangle \rightarrow \sigma'' \quad \text{[Diagram } T_2 \text{]} \quad \langle \text{while } b \text{ do } s \text{ end}, \sigma'' \rangle \rightarrow \sigma'}{\langle s; \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'} (\text{SEQ}_{NS})$$

- Since  $\mathcal{B}[[b]]\sigma = tt$  we can use the rule  $\text{IF}_{NS}$  to obtain a derivation for the “unfolded” loop, as required:

$$\frac{\frac{\text{[Diagram } T_1 \text{]} \quad \langle s, \sigma \rangle \rightarrow \sigma'' \quad \text{[Diagram } T_2 \text{]} \quad \langle \text{while } b \text{ do } s \text{ end}, \sigma'' \rangle \rightarrow \sigma'}{\langle s; \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'} (\text{SEQ}_{NS})}{\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end else skip end}, \sigma \rangle \rightarrow \sigma'} (\text{IF}_{NS})$$

## Proof (“ $\Rightarrow$ ” direction only)

- Let  $b, s, \sigma, \sigma'$  be arbitrary.
- We assume  $\vdash \langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'$ . That is, there is some derivation tree  $T$  such that  $\text{root}(T) = \langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'$
- The last rule application in  $T$  must be one of the (two) rules for while
- (Case: **WhT<sub>NS</sub>** is the last rule applied) i.e.,  $T$  is of the form:

$$\frac{\langle s, \sigma \rangle \rightarrow \sigma'' \quad \langle \text{while } b \text{ do } s \text{ end}, \sigma'' \rangle \rightarrow \sigma'}{\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma'} (\text{WhT}_{NS})$$

Thus, we know that (for some  $\sigma''$ ):

1. There is a derivation tree  $T_1$  with  $\text{root}(T_1) = \langle s, \sigma \rangle \rightarrow \sigma''$
2. There is a derivation tree  $T_2$  with  $\text{root}(T_2) = \langle \text{while } b \text{ do } s \text{ end}, \sigma'' \rangle \rightarrow \sigma'$
3.  $\mathcal{B}[[b]]\sigma = tt$  (from the side-condition)

### Proof: Case “ $\Rightarrow$ ” (cont’d)

- (Case: **WhF<sub>NS</sub>** is the last rule applied in  $T$ ) i.e.,  $T$  is of the form:

$$\frac{}{\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma} (\text{WhF}_{NS})$$

Therefore, we know that

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

- We can construct the derivation tree

$$\frac{\frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma'} (\text{SKIP}_{NS})}{\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end else skip end}, \sigma \rangle \rightarrow \sigma'} (\text{IF}_{NS})$$

- Thus, we have a derivation of  $\langle \text{if } b \text{ then } s; \text{while } b \text{ do } s \text{ end else skip end}, \sigma \rangle \rightarrow \sigma'$ , as required

## Deterministic Semantics

Lemma: The big-step semantics of IMP is deterministic

- We prove

$$\forall s, \sigma, \sigma', \sigma''. (\vdash \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

## Proof Attempt: Structural Induction (2)

- Define  $P(s) \equiv$

$$\forall \sigma, \sigma', \sigma''. (\vdash \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

- **(Case:  $s \equiv \text{skip}$ )**

- Let  $\sigma, \sigma', \sigma''$  be arbitrary
- We assume  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$  and  $\vdash \langle s, \sigma \rangle \rightarrow \sigma''$  and seek to prove that  $\sigma' = \sigma''$
- Our assumption tells us that there exists a derivation tree for  $\langle \text{skip}, \sigma \rangle \rightarrow \sigma'$ . The only tree with this consequence is simply an instantiation of the  $(\text{SKIP}_{NS})$  rule. Thus, we must have  $\sigma = \sigma'$
- Analogously, from  $\vdash \langle \text{skip}, \sigma \rangle \rightarrow \sigma''$ , we can deduce  $\sigma = \sigma''$

- **(Case:  $s \equiv x := e$ )**

- Analogously to the previous case.

## Proof Attempt: Structural Induction

- Reminder: abstract syntax of statements

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

- Structural induction for statements (for  $x, e, b, s, s_1, s_2$  not free in  $\Gamma$ ):

$$\frac{\begin{array}{l} \Gamma \vdash P(\text{skip}) \quad \Gamma \vdash P(x := e) \\ \Gamma, P(s_1), P(s_2) \vdash P(s_1; s_2) \\ \Gamma, P(s_1), P(s_2) \vdash P(\text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}) \\ \Gamma, P(s) \vdash P(\text{while } b \text{ do } s \text{ end}) \end{array}}{\Gamma \vdash \forall s \in \text{Stm}. P(s)}$$

## Proof Attempt: Structural Induction (3)

- **(Case:  $s \equiv \text{while } b \text{ do } s' \text{ end}$ )**

- Let  $\sigma, \sigma', \sigma''$  be arbitrary
- We assume there is a derivation tree for  $\langle \text{while } b \text{ do } s' \text{ end}, \sigma \rangle \rightarrow \sigma'$
- There are two last rules that could have been applied in this derivation
- In the case for  $\text{WHT}$ , we conclude that (for some  $\sigma_1$ ) we have  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma'$

## Proof Attempt: Structural Induction (3)

- (Case:  $s \equiv \text{while } b \text{ do } s' \text{ end}$ )
  - Let  $\sigma, \sigma', \sigma''$  be arbitrary
  - We assume there is a derivation tree for  $\langle \text{while } b \text{ do } s' \text{ end}, \sigma \rangle \rightarrow \sigma'$
  - There are two last rules that could have been applied in this derivation
  - In the case for WHT, we conclude that (for some  $\sigma_1$ ) we have  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma'$
  - Analogously, we derive from  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma \rangle \rightarrow \sigma''$  that  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_2$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_2 \rangle \rightarrow \sigma''$
  - $s'$  is a proper sub-statement of  $s$ . Therefore, we can apply the induction hypothesis ( $P(s')$ ), which allows us to conclude from  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_2$  that  $\sigma_1 = \sigma_2$
  - It remains to show that  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma'$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma''$  imply  $\sigma' = \sigma''$

## Induction on Derivation Trees

- New proof technique: induction on the shape of derivation trees  
To prove a property  $P(T)$  for all derivation trees  $T$ , prove that  $P(T)$  holds for an arbitrary derivation tree  $T$  under the assumption (I.H.) that  $P(T')$  holds for all sub-trees  $T'$  of  $T$ .
- Induction on derivation trees is a special case of well-founded (Noetherian) induction
  - Define  $T' \sqsubset T$  (where  $T, T'$  are derivation trees) to mean that  $T'$  is a proper sub-tree of  $T$
  - $\sqsubset$  is a well-founded ordering, since derivation trees are finite
  - We call  $T'$  a sub-derivation of  $T$  if  $T' \sqsubset T$

## Proof Attempt: Structural Induction (3)

- (Case:  $s \equiv \text{while } b \text{ do } s' \text{ end}$ )
  - Let  $\sigma, \sigma', \sigma''$  be arbitrary
  - We assume there is a derivation tree for  $\langle \text{while } b \text{ do } s' \text{ end}, \sigma \rangle \rightarrow \sigma'$
  - There are two last rules that could have been applied in this derivation
  - In the case for WHT, we conclude that (for some  $\sigma_1$ ) we have  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma'$
  - Analogously, we derive from  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma \rangle \rightarrow \sigma''$  that  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_2$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_2 \rangle \rightarrow \sigma''$
  - $s'$  is a proper sub-statement of  $s$ . Therefore, we can apply the induction hypothesis ( $P(s')$ ), which allows us to conclude from  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_2$  that  $\sigma_1 = \sigma_2$
  - It remains to show that  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma'$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma''$  imply  $\sigma' = \sigma''$

But while  $b \text{ do } s' \text{ end}$  is obviously not a proper sub-statement of  $s$ !

- So we cannot apply the induction hypothesis
- The proof is stuck; our goal even seems similar to the original one
- Structural induction does not work since the transition relation is not defined inductively over the structure of the statements

## Induction on Derivation Trees

- New proof technique: induction on the shape of derivation trees  
To prove a property  $P(T)$  for all derivation trees  $T$ , prove that  $P(T)$  holds for an arbitrary derivation tree  $T$  under the assumption (I.H.) that  $P(T')$  holds for all sub-trees  $T'$  of  $T$ .
- Induction on derivation trees is a special case of well-founded (Noetherian) induction
  - Define  $T' \sqsubset T$  (where  $T, T'$  are derivation trees) to mean that  $T'$  is a proper sub-tree of  $T$
  - $\sqsubset$  is a well-founded ordering, since derivation trees are finite
  - We call  $T'$  a sub-derivation of  $T$  if  $T' \sqsubset T$
- Hint: proofs by induction on the shape of derivation trees typically proceed by case distinction on the rule applied at the root of the arbitrary derivation tree  $T$ . This provides us more information about the structure of the derivation; in particular, it may tell us about sub-derivations, to which our induction hypothesis applies.



## New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

## New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

- This can be written more explicitly (using  $\vdash$  definition) as:

$$\forall s, \sigma, \sigma', \sigma''. (\exists T. \text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma') \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma''$$

- This is logically equivalent to:

$$\forall T. \forall s, \sigma, \sigma', \sigma''. (\text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

Thus, induction can be applied

## New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

- This can be written more explicitly (using  $\vdash$  definition) as:

$$\forall s, \sigma, \sigma', \sigma''. (\exists T. \text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma') \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma''$$

## New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

- This can be written more explicitly (using  $\vdash$  definition) as:

$$\forall s, \sigma, \sigma', \sigma''. (\exists T. \text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma') \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma''$$

- This is logically equivalent to:

$$\forall T. \forall s, \sigma, \sigma', \sigma''. (\text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

Thus, induction can be applied

- We define:

$$P(T) \equiv (\forall s, \sigma, \sigma', \sigma''. \text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

and prove  $\forall T. P(T)$  by induction on the shape of the derivation  $T$

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

$$P(T) \equiv (\forall s, \sigma, \sigma', \sigma''. \text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma' \wedge \vdash \langle s, \sigma \rangle \rightarrow \sigma'' \Rightarrow \sigma' = \sigma'')$$

- Let  $T$  be an arbitrary derivation tree, and let  $s, \sigma, \sigma', \sigma''$  be arbitrary
- We assume  $\text{root}(T) = \langle s, \sigma \rangle \rightarrow \sigma'$  and  $\vdash \langle s, \sigma \rangle \rightarrow \sigma''$
- We now need to prove that  $\sigma' = \sigma''$
- We perform a case distinction on the last rule applied in  $T$ ; this yields seven different cases  
(one for each of the seven rules of the big-step semantics)

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

- (**Case** SEQ<sub>NS</sub>): From the form of the rule, we know:
  - $s = s_1 ; s_2$  for some  $s_1$  and  $s_2$
  - $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma_0$  and  $\vdash \langle s_2, \sigma_0 \rangle \rightarrow \sigma'$ , for some state  $\sigma_0$   
(where these are sub-derivations of  $T$ )

Analogously to the previous cases, we can conclude from  $\vdash \langle s, \sigma \rangle \rightarrow \sigma''$  that the same last rule must be applied, and so  $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle s_2, \sigma_1 \rangle \rightarrow \sigma''$  for some state  $\sigma_1$

The derivation tree for  $\langle s_1, \sigma \rangle \rightarrow \sigma_0$  is a proper sub-tree of  $T$ . Therefore, we can apply the induction hypothesis to  $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma_0$  (with  $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma_1$ ) to obtain  $\sigma_0 = \sigma_1$ . By this equality, we conclude  $\vdash \langle s_2, \sigma_0 \rangle \rightarrow \sigma''$

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

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

- (**Case** SKIP<sub>NS</sub>): From the form of the rule, we know:
  - $s = \text{skip}$
  - $\sigma' = \sigma$

Therefore, the derivation of  $\langle s, \sigma \rangle \rightarrow \sigma''$  is actually a derivation of  $\langle \text{skip}, \sigma \rangle \rightarrow \sigma''$ . The last rule applied in this derivation must also have been the SKIP<sub>NS</sub> rule, from which we obtain  $\sigma'' = \sigma$

- (**Case** ASS<sub>NS</sub>): From the form of the rule, we know:
  - $s = x := e$  for some  $x$  and  $e$
  - $\sigma' = \sigma[x \mapsto \mathcal{A}[[e]]\sigma]$

Therefore, the derivation of  $\langle s, \sigma \rangle \rightarrow \sigma''$  is actually a derivation of  $\langle x := e, \sigma \rangle \rightarrow \sigma''$ . The last rule applied in this derivation must also have been the ASS<sub>NS</sub> rule, from which we obtain  $\sigma'' = \sigma[x \mapsto \mathcal{A}[[e]]\sigma]$

- (**Case** WHF<sub>NS</sub>): Analogously

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

- (**Case** IF<sub>TNS</sub>): From the form of the rule, we know:
  - $s = \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}$  for some  $b, s_1$ , and  $s_2$
  - $\mathcal{B}[[b]]\sigma = tt$
  - There is a derivation tree for  $\langle s_1, \sigma \rangle \rightarrow \sigma'$  (and it is a subderivation of  $T$ )

Therefore, the last rule applied in the derivation tree for  $\langle s, \sigma \rangle \rightarrow \sigma''$  must also have been the IF<sub>TNS</sub> rule

Consequently, we also have  $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma''$

Since the derivation tree for  $\langle s_1, \sigma \rangle \rightarrow \sigma'$  is a proper sub-derivation of  $T$ , we can apply the induction hypothesis to this subderivation (with  $\vdash \langle s_1, \sigma \rangle \rightarrow \sigma''$ ) to obtain  $\sigma' = \sigma''$ .

- (**Case** IF<sub>FNS</sub>): Analogously

## New Proof Attempt: Induction on Shape of Derivation Tree (6)

- (Case  $\text{WHT}_{NS}$ ): From the form of the rule, we know:

- $s = \text{while } b \text{ do } s' \text{ end}$  for some  $b$  and  $s'$
- $\mathcal{B}[[b]]\sigma = tt$
- There are sub-derivations of  $T$  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  $\sigma_0$

Therefore, the last rule applied in the derivation tree for  $\langle s, \sigma \rangle \rightarrow \sigma''$  must also have been the  $\text{WHT}_{NS}$  rule. Consequently, we know that  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$  and  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_1 \rangle \rightarrow \sigma''$ , for some state  $\sigma_1$ .

The derivation tree for  $\langle s', \sigma \rangle \rightarrow \sigma_0$  is a proper sub-derivation of the derivation  $T$ . Therefore, we can apply the induction hypothesis to this sub-derivation (with  $\vdash \langle s', \sigma \rangle \rightarrow \sigma_1$ ) to obtain  $\sigma_0 = \sigma_1$ . By this equality, we conclude that  $\vdash \langle \text{while } b \text{ do } s' \text{ end}, \sigma_0 \rangle \rightarrow \sigma''$ .

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

## Extensions of IMP

- We examine several language extensions for IMP
- For each, we then consider appropriate big-step semantics rules
- Language extensions explored here:
  - Local variables / scopes
  - Procedure declarations / calls
  - Abrupt termination (“abort”)
  - Non-deterministic choice
  - Parallel composition (attempt)
  - More in the exercises
- We will compare other semantics for these extensions later in the course
- Note: these extensions are not part of the core IMP language
  - In particular, when we speak of “IMP”, we still mean the language without extensions, unless we mention extensions explicitly

## 3. Operational Semantics

### 3.1 Big-Step Semantics

- 3.1.1 Natural Semantics of IMP
- 3.1.2 Proving Properties of the Semantics
- 3.1.3 Extensions of IMP

### 3.2 Small-Step Semantics

### 3.3 Equivalence

## Local Variable Declarations

- Idea: statement  $\text{var } x := e \text{ in } s \text{ end}$  declares a local variable that is visible in the sub-statement of the declaration,  $s$
- Semantics
  - Expression  $e$  is evaluated in the initial state
  - Statement  $s$  is executed in a state in which  $x$  has the value of  $e$
  - After the execution of  $s$ , the initial value of  $x$  is restored
- Big-step semantics rule:

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

## 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
  - $y_1, \dots, y_m$  are variable parameters; they can be used to assign values back to the procedure caller.
- Procedures can be declared (syntax as above) as part of a source program, and called (in a statement). We apply the following restrictions:
  - In a procedure declaration, the formal parameter names  $x_1, \dots, x_n, y_1, \dots, y_m$  must be distinct from each other
  - $x_1, \dots, x_n$  and  $y_1, \dots, y_m$  are the only free variables in  $s$  (local variables declared in  $s$  are permitted since they are bound)
  - For procedure calls  $p(e_1, \dots, e_n; z_1, \dots, z_m)$ , the actual variable parameters  $z_1, \dots, z_m$  have to be distinct from each other (no aliasing)

## Natural Semantics of Procedure Calls

- Procedure call  $p(e_1, \dots, e_n; z_1, \dots, z_m)$  in state  $\sigma$ , with declaration `procedure  $p(x_1, \dots, x_n; y_1, \dots, y_m)$  begin  $s$  end`
  - The value arguments  $\vec{e}_i = e_1, \dots, e_n$  are evaluated in the initial state  $\sigma$  to values  $\mathcal{A}[[e_1]]\sigma, \dots, \mathcal{A}[[e_n]]\sigma$
  - The body of the procedure,  $s$ , is executed in a state, in which the value parameters  $\vec{x}_i = x_1, \dots, x_n$  are initialized with the values  $\mathcal{A}[[e_1]]\sigma, \dots, \mathcal{A}[[e_n]]\sigma$ , and the variable parameters  $\vec{y}_j = y_1, \dots, y_m$  are initialized with the values of  $\vec{z}_j = z_1, \dots, z_m$  in the initial state
  - After termination of  $p$ , execution continues in the initial state with the values of  $\vec{y}_j$  assigned to the variables  $\vec{z}_j$

$$\frac{\langle \underline{s}, \sigma_{zero}[\vec{x}_i \mapsto \mathcal{A}[[e_i]]\sigma][\vec{y}_j \mapsto \underline{\sigma}(z_j)] \rangle \rightarrow \underline{\sigma}'}{\langle \underline{p}(\vec{e}_i; \vec{z}_j), \underline{\sigma} \rangle \rightarrow \underline{\sigma}[\vec{z}_j \mapsto \underline{\sigma}'(\underline{y}_j)]} \text{ (CALL}_{NS}\text{)}$$

The notation  $\sigma[\vec{x}_i \mapsto \vec{v}_i]$  abbreviates  $\sigma[x_1 \mapsto v_1][x_2 \mapsto v_2] \dots [x_n \mapsto v_n]$

## Procedures: Example

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

## Abort

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

## Abort: 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!
- Big-step semantics cannot distinguish between non-termination and abnormal termination (being stuck)
  - Natural semantics is only concerned with programs that terminate successfully (reach a final state)
  - Aborting could be modeled by “successful termination” in a special error configuration

## Non-determinism: Observations

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

## Non-determinism

- Idea: for the statement  $s \parallel s'$  either  $s$  or  $s'$  is non-deterministically chosen to be executed
- The statement

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

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

- Rules

$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}'}{\langle \underline{s} \parallel \underline{s}', \underline{\sigma} \rangle \rightarrow \underline{\sigma}'} \text{ (ND1}_{NS}\text{)}$	$\frac{\langle \underline{s}', \underline{\sigma} \rangle \rightarrow \underline{\sigma}'}{\langle \underline{s} \parallel \underline{s}', \underline{\sigma} \rangle \rightarrow \underline{\sigma}'} \text{ (ND2}_{NS}\text{)}$
--	---

## Non-determinism: Observations

- There are derivation trees for
  - $\langle x:=1 \parallel (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 1]$       and
  - $\langle x:=1 \parallel (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 4]$
- There is a derivation tree for
  - $\langle \text{while true do skip end} \parallel (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 4]$

## 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  
 $\langle \text{while true do skip end} \square (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 4]$
- Because big-step semantics cannot accurately describe non-terminating computations, if only one non-deterministic branch terminates successfully, we will only “see” that result

## 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  
 $\langle \text{while true do skip end} \square (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 4]$
- Because big-step semantics cannot accurately describe non-terminating computations, if only one non-deterministic branch terminates successfully, we will only “see” that result
  - In big-step semantics, non-determinism suppresses non-termination, if possible
  - $\sigma[x \mapsto 4]$  is the only possible final state in the above example
  - In fact,  $\text{while true do skip end} \square (x:=2; x:=x+2)$  is semantically equivalent to  $(x:=2; x:=x+2)$  in the big-step semantics
  - In a sense, the big-step semantics only shows the behavior of “correct” choice(s). But this means that we miss some possible behaviors

## 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  
 $\langle \text{while true do skip end} \square (x:=2; x:=x+2), \sigma \rangle \rightarrow \sigma[x \mapsto 4]$
- Because big-step semantics cannot accurately describe non-terminating computations, if only one non-deterministic branch terminates successfully, we will only “see” that result
  - In big-step semantics, non-determinism suppresses non-termination, if possible
  - $\sigma[x \mapsto 4]$  is the only possible final state in the above example

## Termination Revisited

- Recall: for an IMP statement  $s$  we defined termination in the context of big-step semantics as follows:
- The execution of a statement  $s$  in state  $\sigma$ 
  - terminates successfully iff there is a state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
  - fails to terminate iff there is no state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$

## Termination Revisited

- Recall: for an IMP statement  $s$  we defined termination in the context of big-step semantics as follows:
- The execution of a statement  $s$  in state  $\sigma$ 
  - terminates successfully iff there is a state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
  - fails to terminate iff there is no state  $\sigma'$  such that  $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
- According to these definitions
  - `while true do skip end` `skip` terminates successfully
  - `while true do skip end` fails to terminate
  - `abort` fails to terminate

but we cannot give a more precise definition with our big-step semantics

## Parallelism: Observations

- Attempt to define rules:

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow \underline{\sigma'} \quad \langle \underline{s'}, \underline{\sigma'} \rangle \rightarrow \underline{\sigma''}}{\langle \underline{s} \text{ par } \underline{s'}, \underline{\sigma} \rangle \rightarrow \underline{\sigma''}} \text{ (PAR1}_{NS}\text{)}$$

$$\frac{\langle \underline{s'}, \underline{\sigma} \rangle \rightarrow \underline{\sigma'} \quad \langle \underline{s}, \underline{\sigma'} \rangle \rightarrow \underline{\sigma''}}{\langle \underline{s} \text{ par } \underline{s'}, \underline{\sigma} \rangle \rightarrow \underline{\sigma''}} \text{ (PAR2}_{NS}\text{)}$$

- But, these rules do not allow interleaving execution!
- In big-step semantics rules, the executions expressed by premises are reasoned about as atomic steps; therefore, we cannot express interleaving of computations

## Parallelism

- Idea: for the statement  $s \text{ par } s'$  both statements  $s$  and  $s'$  are executed, but execution can be interleaved
- 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  $x := x + 2$

## Problems with Natural Semantics

- Properties of non-terminating programs cannot be expressed
- No distinction between aborting (abnormal termination) and non-termination
- Non-determinism suppresses non-termination (when possible)
- Parallelism (interleaving) cannot be modeled
- Definition of semantic equivalence is coarse (but no reasonable alternative notions are available)
  - All sorting programs are equivalent
  - All non-terminating programs are equivalent

## Reminder: Big-Step and Small-Step Semantics

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

## Structural Operational Semantics (SOS)

- Small-step semantics focuses attention on the individual steps of an execution
  - Execution of assignments
  - Execution of if-conditions, while-iterations, etc.
- Describing small steps of the execution allows one to express the order of execution of individual steps
  - Can be used to express interleaving computations
  - Evaluation order for expressions (not shown in the course)
- Always describing the next small step allows one to express properties of non-terminating programs

## 3. Operational Semantics

### 3.1 Big-Step Semantics

### 3.2 Small-Step Semantics

#### 3.2.1 Structural Operational Semantics of IMP

#### 3.2.2 Proving Properties of the Semantics

#### 3.2.3 Extensions of IMP

### 3.3 Equivalence

## Transitions in SOS

- The configurations are the same as for natural semantics ( $\langle s, \sigma \rangle$  or  $\sigma$ )
  - We use  $\gamma$  as meta-variable for (terminal or non-terminal) configurations
- The transition relation  $\rightarrow_1$  can have two forms
- $\langle s, \sigma \rangle \rightarrow_1 \langle s', \sigma' \rangle$ : the execution of  $s$  from  $\sigma$  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  $\sigma$  has terminated and the final state is  $\sigma'$
- A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  describes the first step of the execution of  $s$  in state  $\sigma$



## 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 a non-terminal configuration  $\langle s, \sigma \rangle$  is *stuck* if there does not exist a configuration  $\gamma$  such that  $\langle s, \sigma \rangle \rightarrow_1 \gamma$ 
  - Note: terminal configurations (final states)  $\sigma$  are never stuck.
- We will again define the transition relation  $\rightarrow_1$  using a derivation system, and write  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$  to mean that there exists a finite derivation tree ending in  $\langle s, \sigma \rangle \rightarrow_1 \gamma$ 
  - $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma \Leftrightarrow \exists T. \text{root}(T) = \langle s, \sigma \rangle \rightarrow_1 \gamma$

## SOS of IMP: Sequential Composition

- Sequential composition  $s; s'$
- First step of executing  $s; s'$  is the first step of executing  $s$
- Either:  $s$  executes completely in one step

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'}{\langle \underline{s}; \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma}' \rangle} (\text{SEQ1}_{\text{SOS}})$$

- Or:  $s$  is not executed completely after one step

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s}; \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}''; \underline{s}', \underline{\sigma}' \rangle} (\text{SEQ2}_{\text{SOS}})$$

## SOS of IMP

- skip does not modify the state

$$\frac{}{\langle \text{skip}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}} (\text{SKIP}_{\text{SOS}})$$

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

$$\frac{}{\langle \underline{x} := \underline{e}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}[\underline{x} \mapsto \mathcal{A}[[\underline{e}]]\underline{\sigma}]} (\text{ASS}_{\text{SOS}})$$

- skip and assignment require only one step to reach a final state
- Rules are analogous to natural semantics; recall:

$$\frac{}{\langle \text{skip}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}} (\text{SKIP}_{\text{NS}})$$

$$\frac{}{\langle \underline{x} := \underline{e}, \underline{\sigma} \rangle \rightarrow \underline{\sigma}[\underline{x} \mapsto \mathcal{A}[[\underline{e}]]\underline{\sigma}]} (\text{ASS}_{\text{NS}})$$

## 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  $b$ , and thereby, which branch to select

$$\frac{}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}, \underline{\sigma} \rangle} (\text{IFT}_{\text{SOS}}) \quad \text{if } \mathcal{B}[[\underline{b}]]\underline{\sigma} = tt$$

$$\frac{}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma} \rangle} (\text{IFF}_{\text{SOS}}) \quad \text{if } \mathcal{B}[[\underline{b}]]\underline{\sigma} = ff$$

- These are the standard rules that we will use for our small-step `if` semantics (unless otherwise specified)

## Alternative Rules 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  $b$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'} \text{ (IFT1}_{SOS}) \quad \text{if } \mathcal{B}[\underline{b}]\underline{\sigma} = tt$$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \text{if } \underline{b} \text{ then } \underline{s} \text{ else } \underline{s}' \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle} \text{ (IFT2}_{SOS}) \quad \text{if } \mathcal{B}[\underline{b}]\underline{\sigma} = tt$$

and two analogous rules for the case  $\mathcal{B}[\underline{b}]\underline{\sigma} = ff$

- The choice between the two rules of the previous slide, and the (four) alternative rules here results in equivalent semantics for IMP
- But the choice can make a difference for languages with parallel execution (different granularity for interleaving)

## Alternative Rules for While 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

$$\frac{}{\langle \text{while } \underline{b} \text{ do } \underline{s} \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}; \text{while } \underline{b} \text{ do } \underline{s} \text{ end}, \underline{\sigma} \rangle} \text{ (WHT}_{SOS})^* \quad \text{*if } \mathcal{B}[\underline{b}]\underline{\sigma} = tt$$

$$\frac{}{\langle \text{while } \underline{b} \text{ do } \underline{s} \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}} \text{ (WHF}_{SOS}) \quad \text{if } \mathcal{B}[\underline{b}]\underline{\sigma} = ff$$

- Or combine with the alternative semantics of the conditional statement
- Alternatives are equivalent for IMP (without extensions)

## SOS of IMP: While Statement

- The first step is to unroll the loop

$$\frac{}{\langle \text{while } \underline{b} \text{ do } \underline{s} \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \text{if } \underline{b} \text{ then } \underline{s}; \text{while } \underline{b} \text{ do } \underline{s} \text{ end else skip end}, \underline{\sigma} \rangle} \text{ (WHILE}_{SOS})$$

- 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 big-step semantics
- This is the standard rule, that we will use for our small-step `while` semantics (unless otherwise specified)

## Multi-Step Executions

- Our binary relation  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  is defined by the derivation rules shown
  - A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  belongs to our transition relation iff  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$

## Multi-Step Executions

- Our binary relation  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  is defined by the derivation rules shown
  - A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  belongs to our transition relation iff  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$
- We now use this to define a relation,  $k$ -step execution, written  $\gamma \rightarrow_1^k \gamma'$ 
  - Intuitive meaning: there is an execution from  $\gamma$  to  $\gamma'$  in exactly  $k$  steps
  - $k$  is a natural number (no negative-length executions)

## Multi-Step Executions

- Our binary relation  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  is defined by the derivation rules shown
  - A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  belongs to our transition relation iff  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$
- We now use this to define a relation,  $k$ -step execution, written  $\gamma \rightarrow_1^k \gamma'$ 
  - Intuitive meaning: there is an execution from  $\gamma$  to  $\gamma'$  in exactly  $k$  steps
  - $k$  is a natural number (no negative-length executions)
- We define the relation  $\gamma \rightarrow_1^k \gamma'$  (inductively over  $k$ ) as follows:
  - $\gamma \rightarrow_1^0 \gamma'$  if and only if  $\gamma = \gamma'$
  - For  $k > 0$ ,  $\gamma \rightarrow_1^k \gamma'$  if and only if there exists  $\gamma''$  such that both  $\vdash \gamma \rightarrow_1 \gamma''$  and  $\gamma'' \rightarrow_1^{k-1} \gamma'$
  - Note:  $\gamma \rightarrow_1^1 \gamma'$  if and only if  $\vdash \gamma \rightarrow_1 \gamma'$  (apply definition for  $k = 1$ )
  - Note:  $\gamma \rightarrow_1^{k_1+k_2} \gamma'$  if and only if  $\exists \gamma''. \gamma \rightarrow_1^{k_1} \gamma'' \wedge \gamma'' \rightarrow_1^{k_2} \gamma'$  (proof?)

## Multi-Step Executions

- Our binary relation  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  is defined by the derivation rules shown
  - A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  belongs to our transition relation iff  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$
- We now use this to define a relation,  $k$ -step execution, written  $\gamma \rightarrow_1^k \gamma'$ 
  - Intuitive meaning: there is an execution from  $\gamma$  to  $\gamma'$  in exactly  $k$  steps
  - $k$  is a natural number (no negative-length executions)
- We define the relation  $\gamma \rightarrow_1^k \gamma'$  (inductively over  $k$ ) as follows:
  - $\gamma \rightarrow_1^0 \gamma'$  if and only if  $\gamma = \gamma'$
  - For  $k > 0$ ,  $\gamma \rightarrow_1^k \gamma'$  if and only if there exists  $\gamma''$  such that both  $\vdash \gamma \rightarrow_1 \gamma''$  and  $\gamma'' \rightarrow_1^{k-1} \gamma'$

## Multi-Step Executions

- Our binary relation  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  is defined by the derivation rules shown
  - A transition  $\langle s, \sigma \rangle \rightarrow_1 \gamma$  belongs to our transition relation iff  $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma$
- We now use this to define a relation,  $k$ -step execution, written  $\gamma \rightarrow_1^k \gamma'$ 
  - Intuitive meaning: there is an execution from  $\gamma$  to  $\gamma'$  in exactly  $k$  steps
  - $k$  is a natural number (no negative-length executions)
- We define the relation  $\gamma \rightarrow_1^k \gamma'$  (inductively over  $k$ ) as follows:
  - $\gamma \rightarrow_1^0 \gamma'$  if and only if  $\gamma = \gamma'$
  - For  $k > 0$ ,  $\gamma \rightarrow_1^k \gamma'$  if and only if there exists  $\gamma''$  such that both  $\vdash \gamma \rightarrow_1 \gamma''$  and  $\gamma'' \rightarrow_1^{k-1} \gamma'$
  - Note:  $\gamma \rightarrow_1^1 \gamma'$  if and only if  $\vdash \gamma \rightarrow_1 \gamma'$  (apply definition for  $k = 1$ )
  - Note:  $\gamma \rightarrow_1^{k_1+k_2} \gamma'$  if and only if  $\exists \gamma''. \gamma \rightarrow_1^{k_1} \gamma'' \wedge \gamma'' \rightarrow_1^{k_2} \gamma'$  (proof?)
- We also define  $\gamma \rightarrow_1^* \gamma'$  to mean  $\exists k. \gamma \rightarrow_1^k \gamma'$ 
  - $\gamma \rightarrow_1^* \gamma'$  means there is an execution from  $\gamma$  to  $\gamma'$  in some finite number of steps

## Derivation Sequences

- A derivation sequence is a (non-empty, finite or infinite) sequence of configurations  $\gamma_0, \gamma_1, \gamma_2, \dots$ , for which:
  - $\gamma_i \rightarrow_1^1 \gamma_{i+1}$  for each  $0 \leq i$  such that  $i+1$  is in the range of the sequence
  - if the derivation sequence is finite then the last configuration in the sequence is either a terminal configuration or a stuck configuration
- Intuitively, a derivation sequence shows a sequence of transitions which cannot be extended with further transitions

## Derivation Sequences

- A derivation sequence is a (non-empty, finite or infinite) sequence of configurations  $\gamma_0, \gamma_1, \gamma_2, \dots$ , for which:
  - $\gamma_i \rightarrow_1^1 \gamma_{i+1}$  for each  $0 \leq i$  such that  $i+1$  is in the range of the sequence
  - if the derivation sequence is finite then the last configuration in the sequence is either a terminal configuration or a stuck configuration
- Intuitively, a derivation sequence shows a sequence of transitions which cannot be extended with further transitions
- Note: if  $\gamma_0, \gamma_1, \gamma_2, \dots$  is a derivation sequence, then, for all  $i$  in the range of the sequence,  $\gamma \rightarrow_1^i \gamma_i$  (proof?)

## Derivation Sequences

- A derivation sequence is a (non-empty, finite or infinite) sequence of configurations  $\gamma_0, \gamma_1, \gamma_2, \dots$ , for which:
  - $\gamma_i \rightarrow_1^1 \gamma_{i+1}$  for each  $0 \leq i$  such that  $i+1$  is in the range of the sequence
  - if the derivation sequence is finite then the last configuration in the sequence is either a terminal configuration or a stuck configuration
- Intuitively, a derivation sequence shows a sequence of transitions which cannot be extended with further transitions
- Note: if  $\gamma_0, \gamma_1, \gamma_2, \dots$  is a derivation sequence, then, for all  $i$  in the range of the sequence,  $\gamma \rightarrow_1^i \gamma_i$  (proof?)
- The length of a derivation sequence  $\gamma_0, \gamma_1, \dots$  is the number of transitions  $\gamma_i \rightarrow_1^1 \gamma_{i+1}$  (with  $i$  and  $i+1$  in the range of the sequence)
  - A finite derivation sequence  $\gamma_0, \gamma_1, \dots, \gamma_k$  has length  $k$
  - Derivation sequences of length  $k$  correspond to  $k$ -step executions  $\gamma_0 \rightarrow_1^k \gamma_k$  in which the final configuration  $\gamma_k$  is either stuck or terminal

## Derivation Sequences: Example

- What is the final state if statement

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

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

## Derivation Sequences: Example

- What is the final state if statement

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

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

$$\begin{aligned}
 & \langle (z:=x; x:=y); y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0] \rangle \\
 & \rightarrow_1^1 \langle x:=y; y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5] \rangle \\
 & \rightarrow_1^1 \langle y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5][x \mapsto 7] \rangle \\
 & \rightarrow_1^1 \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5][x \mapsto 7][y \mapsto 5] \\
 & = \sigma_{zero}[x \mapsto 7][y \mapsto 5][z \mapsto 5]
 \end{aligned}$$

## Derivation Trees

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

$$\begin{aligned}
 & \langle (z:=x; x:=y); y:=z, \sigma \rangle \rightarrow_1^1 \langle x:=y; y:=z, \sigma[z \mapsto 5] \rangle \\
 & \text{where } \sigma = \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0]
 \end{aligned}$$

the derivation tree is

$$\frac{\frac{\overline{\langle z:=x, \sigma \rangle \rightarrow_1 \sigma[z \mapsto 5]}}{\langle z:=x; x:=y, \sigma \rangle \rightarrow_1 \langle x:=y, \sigma[z \mapsto 5] \rangle} (\text{SEQ1}_{SOS})}{\langle (z:=x; x:=y); y:=z, \sigma \rangle \rightarrow_1 \langle x:=y; y:=z, \sigma[z \mapsto 5] \rangle} (\text{SEQ2}_{SOS})$$

- Note: if our statement were instead  $z:=x; (x:=y; y:=z)$ , the corresponding first transition would have a simpler derivation tree with only two rule applications ( $\text{ASS}_{SOS}$  and  $\text{SEQ1}_{SOS}$ )

## Derivation Sequences: Example

- What is the final state if statement

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

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

$$\begin{aligned}
 & \langle (z:=x; x:=y); y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0] \rangle \\
 & \rightarrow_1^1 \langle x:=y; y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5] \rangle \\
 & \rightarrow_1^1 \langle y:=z, \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5][x \mapsto 7] \rangle \\
 & \rightarrow_1^1 \sigma_{zero}[x \mapsto 5][y \mapsto 7][z \mapsto 0][z \mapsto 5][x \mapsto 7][y \mapsto 5] \\
 & = \sigma_{zero}[x \mapsto 7][y \mapsto 5][z \mapsto 5]
 \end{aligned}$$

- The three transitions can be justified by appropriate derivation trees
- The last equality is justified by properties of state updates (proved on Exercise Sheet 9)
- The first four configurations above make up a derivation sequence (of length 3)

## Derivation Sequences and Trees

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

## Termination

- The execution of a statement  $s$  in state  $\sigma$ 
  - terminates iff there is a finite derivation sequence starting with  $\langle s, \sigma \rangle$
  - runs forever iff there is an infinite derivation sequence starting with  $\langle s, \sigma \rangle$
- The execution of a statement  $s$  in state  $\sigma$ 
  - terminates successfully iff  $\exists \sigma'. \langle s, \sigma \rangle \rightarrow_1^* \sigma'$
  - Note: in IMP, an execution terminates successfully iff it terminates; there are no stuck configurations for IMP (proof?)
- Note: these are properties of configurations and not statements alone.
  - while  $x \neq 0$  do  $x := x - 1$  end terminates successfully in some states, and runs forever in others
- We will see later that adding non-determinism to the language can result in statements that both terminate and run forever in a state  $\sigma$

## Proving Properties of Derivation Sequences

- A finite derivation sequence, has a length which is a natural number.
  - Recall: a finite derivation sequence  $\gamma_0, \gamma_1, \dots, \gamma_n$  has length  $n$
- When reasoning about finite derivation sequences, we usually use strong induction on the length of a derivation sequence
- More generally, we reason about a multi-step execution  $\gamma \rightarrow_1^k \gamma'$  by strong induction on the number of steps  $k$ 
  - Define  $P(k) \equiv$  “for all executions of length  $k$ , our property holds”.
  - Prove  $P(k)$  for arbitrary  $k$ , with the induction hypothesis  $\forall k' < k. P(k')$

## 3. Operational Semantics

### 3.1 Big-Step Semantics

### 3.2 Small-Step Semantics

#### 3.2.1 Structural Operational Semantics of IMP

#### 3.2.2 Proving Properties of the Semantics

#### 3.2.3 Extensions of IMP

### 3.3 Equivalence

## Using Induction on Multi-Step Executions

- After setting up the induction, the proof often proceeds by:
- Dealing with the case of a 0-step execution specially (if applicable)
- Dealing with all other cases by “splitting off” the first execution step
- For this first step, we often get more information by considering either:
  - the structure of the statement in the initial configuration, or
  - the derivation tree validating the first step of the execution
- The remaining steps (after the first) form an execution with fewer steps, to which our induction hypothesis applies
- Example: we will prove the Lemma

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

## Proof

We define

$$P(k) \equiv \forall s_1, s_2, \sigma, \sigma''. \langle s_1; s_2, \sigma \rangle \rightarrow_1^k \sigma'' \Rightarrow \exists \sigma', k_1, k_2. \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$$

- We prove  $\forall k. P(k)$  by strong induction on  $k$ . Thus, we need to show  $P(k)$  for arbitrary  $k$ , and get the induction hypothesis  $\forall m < k. P(m)$
- In our desired conclusion  $P(k)$ , let  $s_1, s_2, \sigma, \sigma''$  be arbitrary
- To prove the desired implication, we assume  $\langle s_1; s_2, \sigma \rangle \rightarrow_1^k \sigma''$  and need to show  $\exists \sigma', k_1, k_2. \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$
- Consider the case  $k = 0$ . This immediately gives a contradiction, since, by the definition of  $\rightarrow_1^0$  we conclude  $\langle s_1; s_2, \sigma \rangle = \sigma''$
- Consider the case  $k > 0$ . Then the execution  $\langle s_1; s_2, \sigma \rangle \rightarrow_1^k \sigma''$  can be split up as:  $\langle s_1; s_2, \sigma \rangle \rightarrow_1^1 \gamma \rightarrow_1^{k-1} \sigma''$  for some configuration  $\gamma$

## Proof: Case 1

- From the form of the rule SEQ1<sub>SOS</sub> we obtain, for some state  $\sigma_1$ :  $\vdash \langle s_1, \sigma \rangle \rightarrow_1 \sigma_1$  and  $\gamma = \langle s_2, \sigma_1 \rangle$ .
- Thus, we have both  $\langle s_1, \sigma \rangle \rightarrow_1^1 \sigma_1$  and  $\langle s_2, \sigma_1 \rangle \rightarrow_1^{k-1} \sigma''$
- Choosing  $\sigma' = \sigma_1$  and  $k_1 = 1$  and  $k_2 = k - 1$ , we obtain the required result:  $\exists \sigma', k_1, k_2. \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$

## Proof (cont'd)

- $\langle s_1; s_2, \sigma \rangle \rightarrow_1^1 \gamma \rightarrow_1^{k-1} \sigma''$
- Consider the derivation tree justifying the step  $\langle s_1; s_2, \sigma \rangle \rightarrow_1^1 \gamma$
- There are two possible last rules in this derivation:
- Case 1 (rule SEQ1<sub>SOS</sub> - recall:)

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'}{\langle \underline{s}; \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma}' \rangle} \text{ (SEQ1}_{SOS}\text{)}$$

- Case 2: (rule SEQ2<sub>SOS</sub> - recall:)

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s}; \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}''; \underline{s}', \underline{\sigma}' \rangle} \text{ (SEQ2}_{SOS}\text{)}$$

## Proof: Case 2

- From the form of the rule SEQ2<sub>SOS</sub> we obtain, for some  $s'_1$  and  $\sigma_1$ :  $\vdash \langle s_1, \sigma \rangle \rightarrow_1 \langle s'_1, \sigma_1 \rangle$  and  $\gamma = \langle s'_1; s_2, \sigma_1 \rangle$
- Thus, we have  $\langle s_1, \sigma \rangle \rightarrow_1^1 \langle s'_1, \sigma_1 \rangle$
- From  $\gamma \rightarrow_1^{k-1} \sigma''$  we also get  $\langle s'_1; s_2, \sigma_1 \rangle \rightarrow_1^{k-1} \sigma''$
- By applying the induction hypothesis (since  $k - 1 < k$ ), we get  $\exists \sigma_2, l_1, l_2 : \langle s'_1, \sigma_1 \rangle \rightarrow_1^{l_1} \sigma_2 \wedge \langle s_2, \sigma_2 \rangle \rightarrow_1^{l_2} \sigma'' \wedge l_1 + l_2 = k - 1$
- From  $\langle s_1, \sigma \rangle \rightarrow_1^1 \langle s'_1, \sigma_1 \rangle$  and  $\langle s'_1, \sigma_1 \rangle \rightarrow_1^{l_1} \sigma_2$  we get  $\langle s_1, \sigma \rangle \rightarrow_1^{l_1+1} \sigma_2$
- This, by taking  $\sigma' = \sigma_2$ ,  $k_1 = (l_1 + 1)$  and  $k_2 = l_2$ , we obtain the required result:  $\exists \sigma', k_1, k_2. \langle s_1, \sigma \rangle \rightarrow_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \rightarrow_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$

## Semantic Equivalence

Under the small-step semantics, two statements  $s_1$  and  $s_2$  are semantically equivalent if for all states  $\sigma$ , both:

- for all stuck or terminal configurations  $\gamma$ , we have  $\langle s_1, \sigma \rangle \rightarrow_1^* \gamma$  if and only if  $\langle s_2, \sigma \rangle \rightarrow_1^* \gamma$ , and
- there is an infinite derivation sequence starting in  $\langle s_1, \sigma \rangle$  if and only if there is one starting in  $\langle s_2, \sigma \rangle$

- Note: in the first case, the lengths of the two derivation sequences may be different
- Note: the intermediate configurations making up the derivation sequences may also be different

## 3. Operational Semantics

### 3.1 Big-Step Semantics

### 3.2 Small-Step Semantics

#### 3.2.1 Structural Operational Semantics of IMP

#### 3.2.2 Proving Properties of the Semantics

#### 3.2.3 Extensions of IMP

### 3.3 Equivalence

## Determinism

Lemma: The small-step semantics of IMP is deterministic.

That is, for all  $s, \sigma, \gamma$ , and  $\gamma'$  we have that  
 $\vdash \langle s, \sigma \rangle \rightarrow_1 \gamma \wedge \vdash \langle s, \sigma \rangle \rightarrow_1 \gamma' \Rightarrow \gamma = \gamma'$

- 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 a configuration  $\langle s, \sigma \rangle$

## Local Variable Declarations

- Local variable declaration `var x:=e in s end`
- The steps are
  1. Assign  $e$  to  $x$
  2. Execute  $s$  (possibly in several steps)
  3. Restore the initial value of  $x$   
(necessary if  $x$  exists in the enclosing scope)
- The first small step could be easily defined:

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

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



## Artificial End Marker

- We extend the syntactic category Stm with a restore statement

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

- Note that the restore statement contains a *value*, not an expression
- The restore statement is used internally by the semantics but must not occur in source programs.
- Now we can use the restore statement to mark the end of the scope of a local variable and remember its original value:

$$\frac{}{\langle \text{var } \underline{x} := \underline{e} \text{ in } \underline{s} \text{ end}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}; \text{restore } (\underline{x}, \underline{\sigma}(\underline{x})), \underline{\sigma}[\underline{x} \mapsto \mathcal{A}[\underline{e}]]\underline{\sigma} \rangle} (\text{LOC}_{\text{SOS}})$$

$$\frac{}{\langle \text{restore } (\underline{x}, \underline{v}), \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}[\underline{x} \mapsto \underline{v}]} (\text{RET}_{\text{SOS}})$$

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

## Abort: Observations

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

$$\begin{aligned} &\langle \text{while true do skip end}, \sigma \rangle \rightarrow_1^1 \\ &\langle \text{if true then skip; while true do skip end end}, \sigma \rangle \rightarrow_1^1 \\ &\langle \text{skip; while true do skip end}, \sigma \rangle \rightarrow_1^1 \\ &\langle \text{while true do skip end}, \sigma \rangle \rightarrow_1^1 \dots \end{aligned}$$

- In our small-step semantics,
  - running forever is reflected by infinite derivation sequences
  - abnormal termination is reflected by finite derivation sequences ending in a stuck configuration

## Abort Statement

- Statement abort stops the execution of the complete program
- Aborting 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  $\sigma$
  - $\langle \text{skip}, \sigma \rangle \rightarrow_1^1 \sigma$  is the only derivation sequence for skip starting in  $\sigma$
  - skip terminates successfully in all states, whereas abort terminates in all states, but not successfully

## Non-determinism

- For the statement  $s \sqcap s'$  either  $s$  or  $s'$  is non-deterministically chosen to be executed
- The statement

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

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

- Rules

$$\frac{}{\langle \underline{s} \sqcap \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}, \underline{\sigma} \rangle} (\text{ND1}_{\text{SOS}})$$

$$\frac{}{\langle \underline{s} \sqcap \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma} \rangle} (\text{ND2}_{\text{SOS}})$$

## Non-determinism: Observations

- There are two derivation sequences
  - $\langle x:=1 \parallel (x:=2; x:=x+2), \sigma \rangle \rightarrow_1^* \sigma[x \mapsto 1]$
  - $\langle x:=1 \parallel (x:=2; x:=x+2), \sigma \rangle \rightarrow_1^* \sigma[x \mapsto 4]$
- There are also two derivation sequences for  $\langle \text{while true do skip end} \parallel (x:=2; x:=x+2), \sigma \rangle$ 
  - a finite derivation sequence leading to  $\sigma[x \mapsto 4]$
  - an infinite derivation sequence
- A small-step semantics can always show the effect of choosing either branch of a non-deterministic choice
- In particular, in the small-step semantics, non-determinism does not suppress non-termination; we do not lose behaviors

## Example: Interleaving

- The statement

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

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

## Parallelism

- For the statement  $s \text{ par } s'$  both statements  $s$  and  $s'$  are executed, but execution can be interleaved

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'' \text{ par } \underline{s}', \underline{\sigma}' \rangle} (\text{PAR1}_{SOS})$$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma}' \rangle} (\text{PAR2}_{SOS})$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s} \text{ par } \underline{s}'', \underline{\sigma}' \rangle} (\text{PAR3}_{SOS})$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \underline{\sigma}'}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}, \underline{\sigma}' \rangle} (\text{PAR4}_{SOS})$$

## Example: Derivation Sequences

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

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

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

## Comparison: Summary

### Natural Semantics

- Local variable declarations and procedures can be modeled easily
- No distinction between aborting and running forever
- Non-determinism suppresses non-termination (when possible)
- Interleaving parallelism cannot be modeled

### Structural Operational Semantics

- Local variable declarations (and procedures) require an explicit encoding of the original state
- Distinction between aborting and running forever
- Non-determinism does not suppress non-termination
- Interleaving parallelism can be modeled

## Semantic Functions

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

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

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

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

## 3. Operational Semantics

### 3.1 Big-Step Semantics

### 3.2 Small-Step Semantics

### 3.3 Equivalence

## Equivalence Theorem

Theorem: For every statement  $s$  of IMP,  $\mathcal{S}_{NS}[[s]]$  is defined iff  $\mathcal{S}_{SOS}[[s]]$  is defined, and (when defined)  $\mathcal{S}_{NS}[[s]] = \mathcal{S}_{SOS}[[s]]$

- If the execution of  $s$  from some state terminates successfully in one of the semantics then it also terminate successfully in the other, and the resulting final states will be equal
- Note: this also implies that a statement in a state fails to terminate in the big step semantics if and only if it either gets stuck or runs forever in the small-step semantics

## Equivalence Lemma 1

Lemma: For every statement  $s$  of IMP and states  $\sigma$  and  $\sigma'$  we have  $\vdash \langle s, \sigma \rangle \rightarrow \sigma' \Rightarrow \langle s, \sigma \rangle \rightarrow_1^* \sigma'$

- If the execution of  $s$  from  $\sigma$  terminates successfully in the big-step semantics then it will terminate successfully in the same final state in the small-step semantics
- The proof runs by induction on the shape of the derivation tree for  $\langle s, \sigma \rangle \rightarrow \sigma'$  (see exercises)

## Equivalence Theorem: Proof

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

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

- The combination of the two Equivalence Lemmas shows (for all  $s, \sigma, \sigma'$ ) that:
  - If one of  $\mathcal{S}_{NS}[[s]]\sigma$  and  $\mathcal{S}_{SOS}[[s]]\sigma$  is defined then the other is also defined
  - $\mathcal{S}_{NS}[[s]]\sigma = \sigma' \Leftrightarrow \mathcal{S}_{SOS}[[s]]\sigma = \sigma'$
- This is sufficient to prove  $\mathcal{S}_{NS}[[s]] = \mathcal{S}_{SOS}[[s]]$  (i.e., the two are equal as semantic functions)

## Equivalence Lemma 2

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

- If the execution of  $s$  from  $\sigma$  terminates successfully in the small-step semantics then it will terminate successfully in the same final state in the big-step semantics
- The proof runs by induction on the number of steps  $k$  (see exercises)

## Equivalence: Summary

- The big-step (natural) and small-step (structural operational) semantics are equivalent for IMP (when considering complete executions)
  - Proof of Lemma 1 runs by induction on the shape of the derivation tree
  - Proof of Lemma 2 runs by induction on the number of execution steps
- For extended languages, different formalizations of the equivalence theorem could be necessary
  - For non-deterministic languages, above functions are not well-defined (but we could consider the set of all possible final states)
- For properties other than successful termination, different formalisations of the equivalence theorem and/or semantics could be necessary
  - Comparing abnormal termination in the two semantics (would require extending big-step semantics, e.g. with special error states)