Formal Methods and Functional Programming

Operational Semantics

Peter Müller

Chair of Programming Methodology
ETH Zurich

Operational Semantics of Statements

- Recall: evaluation of an expression in a state yields a value

```
x + 2 * y
```

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

- Execution of a statement modifies the state

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

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

Big-Step and Small-Step Semantics

- Big-step semantics describe how the overall results of the executions are obtained
  - 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

3. Operational Semantics

Transition Systems

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

Transition Systems

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

Transition Systems

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

Transition Systems

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

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

Inference Rules

- We specify the transition relation by rules of the form

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

*(*optional: side-condition*)

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

- Meaning of the rule

If $\varphi_1, \ldots, \varphi_n$ are transitions
(and *side-condition* is true)
then ψ is a transition

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

Big-Step Semantics of IMP

- `skip` does not modify the state

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

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

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

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

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to \underline{\sigma}' \quad \langle \underline{s}', \underline{\sigma}' \rangle \to \underline{\sigma}''}{\langle \underline{s}\,;\underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}''} \; (\textsc{Seq}_{NS})$$

Big-Step Semantics of IMP (cont'd)

- Conditional statement $\texttt{if}\ b\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{end}$
  - If b holds, s is executed
  - If b does not hold, s' is executed

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

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

Big-Step 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'

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

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

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

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

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

- Assignment rule instance

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

Derivation Trees

- Rule instances can be combined to derive a transition $\langle s, \sigma \rangle \to \sigma'$
- The result is a derivation tree T
  - The root of T is $\langle s, \sigma \rangle \to \sigma'$, written as $root(T) = \langle s, \sigma \rangle \to \sigma'$
  - 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 $\langle s, \sigma \rangle \to \sigma'$, written as $\vdash \langle s, \sigma \rangle \to \sigma'$, if and only if there exists a finite derivation tree ending in $\langle s, \sigma \rangle \to \sigma'$
  - $\vdash \langle s, \sigma \rangle \to \sigma' \Leftrightarrow \exists T. root(T) = \langle s, \sigma \rangle \to \sigma'$

Derivations: Example

- What is the result of executing statement

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

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

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

- 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 $x \not\equiv y$), which will be proved in the exercises

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 σ
  - terminates successfully iff there exists a state σ' such that $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$
  - fails to terminate iff there is no state σ' such that $\vdash \langle s, \sigma \rangle \rightarrow \sigma'$

- For example, `while true do skip end` fails to terminate

3. Operational Semantics

Semantic Equivalence

- Definition

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

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

- 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

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

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

- 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 ("⇒" direction only)

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

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

  Thus, we know that (for some σ''):

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

Proof: Case "⇒" (cont'd)

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

$$\dfrac{\begin{array}{cc} T_1 & T_2 \\ \langle s, \sigma\rangle \to \sigma'' & \langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma''\rangle \to \sigma' \end{array}}{\langle s\texttt{;while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma'}\ (\textsc{Seq}_{NS})$$

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

$$\dfrac{\dfrac{\begin{array}{cc} T_1 & T_2 \\ \langle s, \sigma\rangle \to \sigma'' & \langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma''\rangle \to \sigma' \end{array}}{\langle s\texttt{;while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma'}\ (\textsc{Seq}_{NS})}{\langle \texttt{if } b \texttt{ then } s\texttt{;while } b \texttt{ do } s \texttt{ end else skip end}, \sigma\rangle \to \sigma'}\ (\textsc{IfT}_{NS})$$

Proof: Case "⇒" (cont'd)

- (**Case: WhF$_{NS}$ is the last rule applied in** T) i.e., T is of the form:

$$\frac{}{\langle \texttt{while } b \texttt{ do } s \texttt{ end}, \sigma\rangle \to \sigma}\ (\textsc{WhF}_{NS})$$

  Therefore, we know that

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

- We can construct the derivation tree

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

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

Deterministic Semantics

Lemma: The big-step semantics of IMP is deterministic

- We prove

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

Proof Attempt: Structural Induction

- Reminder: abstract syntax of statements

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

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

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

Proof Attempt: Structural Induction (2)

- Define $P(s) \equiv$
  $\forall \sigma, \sigma', \sigma''.(\vdash \langle s, \sigma \rangle \to \sigma' \quad \wedge \quad \vdash \langle s, \sigma \rangle \to \sigma'' \quad \Rightarrow \quad \sigma' = \sigma'')$

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

- **(Case: $s \equiv x := e$)**
  - Analogously to the previous case.

Proof Attempt: Structural Induction (3)

- **(Case:** $s \equiv \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$**)**
  - Let σ,σ',σ'' be arbitrary
  - We assume there is a derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma'$
  - There are two last rules that could have been applied in this derivation
  - In the case for WhT, we conclude that (for some σ_1) we have $\vdash \langle s', \sigma\rangle \to \sigma_1$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$

Proof Attempt: Structural Induction (3)

- **(Case: $s \equiv \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$)**
  - Let σ,σ',σ'' be arbitrary
  - We assume there is a derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma'$
  - There are two last rules that could have been applied in this derivation
  - In the case for WHT, we conclude that (for some σ_1) we have $\vdash \langle s', \sigma\rangle \to \sigma_1$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$
  - Analogously, we derive from $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma''$ that $\vdash \langle s', \sigma\rangle \to \sigma_2$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_2\rangle \to \sigma''$
  - s' is a proper sub-statement of s. Therefore, we can apply the induction hypothesis ($P(s')$), which allows us to conclude from $\vdash \langle s', \sigma\rangle \to \sigma_1$ and $\vdash \langle s', \sigma\rangle \to \sigma_2$ that $\sigma_1 = \sigma_2$
  - It remains to show that $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma''$ imply $\sigma' = \sigma''$

Proof Attempt: Structural Induction (3)

- **(Case: $s \equiv \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$)**
  - Let σ,σ',σ'' be arbitrary
  - We assume there is a derivation tree for $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma'$
  - There are two last rules that could have been applied in this derivation
  - In the case for WHT, we conclude that (for some σ_1) we have $\vdash \langle s', \sigma\rangle \to \sigma_1$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$
  - Analogously, we derive from $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma\rangle \to \sigma''$ that $\vdash \langle s', \sigma\rangle \to \sigma_2$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_2\rangle \to \sigma''$
  - s' is a proper sub-statement of s. Therefore, we can apply the induction hypothesis ($P(s')$), which allows us to conclude from $\vdash \langle s', \sigma\rangle \to \sigma_1$ and $\vdash \langle s', \sigma\rangle \to \sigma_2$ that $\sigma_1 = \sigma_2$
  - It remains to show that $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma'$ and $\vdash \langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_1\rangle \to \sigma''$ imply $\sigma' = \sigma''$

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

- So we cannot apply the induction hypothesis
- The proof is stuck; 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 $T' \sqsubset T$

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:

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

New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

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

$$\forall s, \sigma, \sigma', \sigma''.(\exists T.root(T) = \langle s, \sigma \rangle \to \sigma') \wedge \vdash \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma''$$

New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

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

$$\forall s, \sigma, \sigma', \sigma''.(\exists T.root(T) = \langle s, \sigma \rangle \to \sigma') \wedge \vdash \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma''$$

- This is logically equivalent to:

$$\forall T. \forall s, \sigma, \sigma', \sigma''.(root(T) = \langle s, \sigma \rangle \to \sigma' \wedge \vdash \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma'')$$

  Thus, induction can be applied

New Proof Attempt: Induction on Shape of Derivation Tree

- Recall that we want to prove:

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

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

$$\forall s, \sigma, \sigma', \sigma''. (\exists T. root(T) = \langle s, \sigma \rangle \to \sigma') \wedge \vdash \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma''$$

- This is logically equivalent to:

$$\forall T. \forall s, \sigma, \sigma', \sigma''. (root(T) = \langle s, \sigma \rangle \to \sigma' \wedge \vdash \langle s, \sigma \rangle \to \sigma'' \Rightarrow \sigma' = \sigma'')$$

  Thus, induction can be applied

- We define:

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

  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)

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

- Let T be an arbitrary derivation tree, and let s,σ,σ',σ'' be arbitrary
- We assume $root(T) = \langle s, \sigma \rangle \to \sigma'$ and $\vdash \langle s, \sigma \rangle \to \sigma''$
- 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 (3)

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

  Therefore, the derivation of $\langle s, \sigma \rangle \to \sigma''$ is actually a derivation of $\langle \texttt{skip}, \sigma \rangle \to \sigma''$. The last rule applied in this derivation must also have been the $\textsc{Skip}_{NS}$ rule, from which we obtain $\sigma'' = \sigma$

- (**Case** $\textsc{Ass}_{NS}$): From the form of the rule, we know:
  - $s = x := e$ for some x and e
  - $\sigma' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$

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

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

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

- (**Case** SEQ_{NS}): From the form of the rule, we know:
  - $s = s_1 ; s_2$ for some s_1 and s_2
  - $\vdash \langle s_1, \sigma \rangle \to \sigma_0$ and $\vdash \langle s_2, \sigma_0 \rangle \to \sigma'$, for some state σ_0 (where these are sub-derivations of T)

  Analogously to the previous cases, we can conclude from $\vdash \langle s, \sigma \rangle \to \sigma''$ that the same last rule must be applied, and so $\vdash \langle s_1, \sigma \rangle \to \sigma_1$ and $\vdash \langle s_2, \sigma_1 \rangle \to \sigma''$ for some state σ_1

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

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

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

- (**Case** IFT_{NS}): From the form of the rule, we know:
  - $s = \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}$ for some b, s_1, and s_2
  - $\mathcal{B}[\![b]\!]\sigma = tt$
  - There is a derivation tree for $\langle s_1, \sigma\rangle \to \sigma'$ (and it is a subderivation of T)

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

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

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

- (**Case** IFF_{NS}): Analogously

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

- (**Case** WHT_{NS}): From the form of the rule, we know:
  - $s = \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}$ for some b and s'
  - $\mathcal{B}[\![b]\!]\sigma = tt$
  - There are sub-derivations of T for $\langle s', \sigma\rangle \to \sigma_0$ and $\langle \texttt{while}\ b\ \texttt{do}\ s'\ \texttt{end}, \sigma_0\rangle \to \sigma'$, for some state σ_0

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

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

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

3. Operational Semantics

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

Local Variable Declarations

- Idea: statement `var` x`:=`e `in` s `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:

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

Procedure Declarations and Calls

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

- Formal parameters
  - $x_1, \ldots, x_n$ are value parameters
  - $y_1, \ldots, y_m$ 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 $x_1, \ldots, x_n, y_1, \ldots, y_m$ must be distinct from each other
  - $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ are the only free variables in s (local variables declared in s are permitted since they are bound)
  - For procedure calls $p(e_1, \ldots, e_n; z_1, \ldots, z_m)$, the actual variable parameters $z_1, \ldots, z_m$ have to be distinct from each other (no aliasing)

Procedures: Example

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

Natural Semantics of Procedure Calls

- Procedure call $p(e_1, \ldots, e_n; z_1, \ldots, z_m)$ in state σ, with declaration `procedure` $p(x_1, \ldots, x_n; y_1, \ldots, y_m)$ `begin` s `end`
  - The value arguments $\overrightarrow{e_i} = e_1, \ldots, e_n$ are evaluated in the initial state σ to values $\mathcal{A}[\![e_1]\!]\sigma, \ldots, \mathcal{A}[\![e_n]\!]\sigma$
  - The body of the procedure, s, is executed in a state, in which the value parameters $\overrightarrow{x_i} = x_1, \ldots, x_n$ are initialized with the values $\mathcal{A}[\![e_1]\!]\sigma, \ldots, \mathcal{A}[\![e_n]\!]\sigma$, and the variable parameters $\overrightarrow{y_j} = y_1, \ldots, y_m$ are initialized with the values of $\overrightarrow{z_j} = z_1, \ldots, z_m$ in the initial state
  - After termination of p, execution continues in the initial state with the values of $\overrightarrow{y_j}$ assigned to the variables $\overrightarrow{z_j}$

$$\frac{\langle \underline{s}, \sigma_{zero}[\underline{\overrightarrow{x_i}} \mapsto \underline{\overrightarrow{\mathcal{A}[\![e_i]\!]\sigma}}][\underline{\overrightarrow{y_j}} \mapsto \underline{\overrightarrow{\sigma(z_j)}}]\rangle \to \underline{\sigma'}}{\langle \underline{p}(\underline{\overrightarrow{e_i}}; \underline{\overrightarrow{z_j}}), \underline{\sigma}\rangle \to \underline{\sigma}[\underline{\overrightarrow{z_j}} \mapsto \underline{\overrightarrow{\sigma'(y_j)}}]} \ (\textsc{Call}_{NS})$$

The notation $\sigma[\overrightarrow{x_i} \mapsto \overrightarrow{v_i}]$ abbreviates $\sigma[x_1 \mapsto v_1][x_2 \mapsto v_2] \ldots [x_n \mapsto v_n]$

Abort

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

Abort: Observations

- `abort` and `skip` are not semantically equivalent since there is a derivation tree for $\langle \texttt{skip}, \sigma \rangle \to \sigma$, but not for $\langle \texttt{abort}, \sigma \rangle \to \sigma'$
- `abort` and `while true do skip end` are semantically equivalent!
- 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

- Idea: for the statement $s \,[\!]\, s'$ either s or s' is non-deterministically chosen to be executed

- The statement

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

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

- Rules

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to \underline{\sigma}'}{\langle \underline{s} \,[\!]\, \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}'} \,(\mathrm{ND1}_{NS}) \qquad \frac{\langle \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}'}{\langle \underline{s} \,[\!]\, \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}'} \,(\mathrm{ND2}_{NS})$$

Non-determinism: Observations

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

Non-determinism: Observations

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

- There is a derivation tree for

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

Non-determinism: Observations

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

- There is a derivation tree for
  $$\langle \texttt{while true do skip end} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle \to \sigma[\texttt{x} \mapsto 4]$$

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

- There is a derivation tree for

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

- 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

Non-determinism: Observations

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

- There is a derivation tree for

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

- 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, $\texttt{while true do skip end} \,[]\, (\texttt{x:=2; x:=x+2})$ is semantically equivalent to $(\texttt{x:=2; x:=x+2})$ 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

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 σ
  - terminates successfully iff there is a state σ' such that $\vdash \langle s, \sigma \rangle \to \sigma'$
  - fails to terminate iff there is no state σ' such that $\vdash \langle s, \sigma \rangle \to \sigma'$

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 σ
  - terminates successfully iff there is a state σ' such that $\vdash \langle s, \sigma \rangle \to \sigma'$
  - fails to terminate iff there is no state σ' such that $\vdash \langle s, \sigma \rangle \to \sigma'$

- 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

- Idea: for the statement s `par` s' both statements s and s' 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:

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to \underline{\sigma}' \quad \langle \underline{s}', \underline{\sigma}' \rangle \to \underline{\sigma}''}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}''} \; (\textsc{Par1}_{NS})$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}' \quad \langle \underline{s}, \underline{\sigma}' \rangle \to \underline{\sigma}''}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to \underline{\sigma}''} \; (\textsc{Par2}_{NS})$$

- 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

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)

3. Operational Semantics

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

Transitions in SOS

- The configurations are the same as for natural semantics ($\langle s, \sigma \rangle$ or σ)
  - We use γ 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 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 in state 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 a non-terminal configuration $\langle s, \sigma\rangle$ is stuck if there does not exist a configuration γ such that $\langle s, \sigma\rangle \rightarrow_1 \gamma$
  - Note: terminal configurations (final states) σ 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 $\langle s, \sigma\rangle \rightarrow_1 \gamma$
  - $\vdash \langle s, \sigma\rangle \rightarrow_1 \gamma \Leftrightarrow \exists T.\mathit{root}(T) = \langle s, \sigma\rangle \rightarrow_1 \gamma$

SOS of IMP

- `skip` does not modify the state

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

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

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

- `skip` and assignment require only one step to reach a final state

- Rules are analogous to natural semantics; recall:

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

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

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

$$\frac{\langle \underline{s}, \underline{\sigma}\rangle \to_1 \underline{\sigma'}}{\langle \underline{s}\,;\underline{s'}, \underline{\sigma}\rangle \to_1 \langle \underline{s'}, \underline{\sigma'}\rangle}\ (\textsc{Seq1}_{SOS})$$

- Or: s is not executed completely after one step

$$\frac{\langle \underline{s}, \underline{\sigma}\rangle \to_1 \langle \underline{s''}, \underline{\sigma'}\rangle}{\langle \underline{s}\,;\underline{s'}, \underline{\sigma}\rangle \to_1 \langle \underline{s''}\,;\underline{s'}, \underline{\sigma'}\rangle}\ (\textsc{Seq2}_{SOS})$$

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

$$\frac{}{\langle \texttt{if } \underline{b} \texttt{ then } \underline{s} \texttt{ else } \underline{s}' \texttt{ end}, \underline{\sigma} \rangle \to_1 \langle \underline{s}, \underline{\sigma} \rangle} \ (\textsc{IfT}sos) \quad \text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = tt$$

$$\frac{}{\langle \texttt{if } \underline{b} \texttt{ then } \underline{s} \texttt{ else } \underline{s}' \texttt{ end}, \underline{\sigma} \rangle \to_1 \langle \underline{s}', \underline{\sigma} \rangle} \ (\textsc{IfF}sos) \quad \text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = ff$$

- 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

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to_1 \underline{\sigma}'}{\langle \texttt{if } \underline{b} \texttt{ then } \underline{s} \texttt{ else } \underline{s}' \texttt{ end}, \underline{\sigma} \rangle \to_1 \underline{\sigma}'} \; (\text{IFT1}_{SOS}) \quad \text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = tt$$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \texttt{if } \underline{b} \texttt{ then } \underline{s} \texttt{ else } \underline{s}' \texttt{ end}, \underline{\sigma} \rangle \to_1 \langle \underline{s}'', \underline{\sigma}' \rangle} \; (\text{IFT2}_{SOS}) \quad \text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = tt$$

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)

SOS of IMP: While Statement

- The first step is to unroll the loop

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

- 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)

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

$$\frac{}{\langle \texttt{while}\ \underline{b}\ \texttt{do}\ \underline{s}\ \texttt{end}, \underline{\sigma}\rangle \to_1 \langle \underline{s}\texttt{;while}\ \underline{b}\ \texttt{do}\ \underline{s}\ \texttt{end}, \underline{\sigma}\rangle}\ (\textsc{WhT}_{sos})^*$$

$$^*\text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = tt$$

$$\frac{}{\langle \texttt{while}\ \underline{b}\ \texttt{do}\ \underline{s}\ \texttt{end}, \underline{\sigma}\rangle \to_1 \underline{\sigma}}\ (\textsc{WhF}_{sos}) \quad \text{if } \mathcal{B}[\![\underline{b}]\!]\underline{\sigma} = ff$$

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

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 γ to γ' 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 γ to γ' 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 γ'' 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 γ to γ' 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 γ'' 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 γ to γ' 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 γ'' 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 γ to γ' in some finite number of steps

Derivation Sequences

- A derivation sequence is a (non-empty, finite or infinite) sequence of configurations $\gamma_0, \gamma_1, \gamma_2, \ldots$, 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 $\gamma_0, \gamma_1, \gamma_2, \ldots$, 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 $\gamma_0, \gamma_1, \gamma_2, \ldots$ 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 $\gamma_0, \gamma_1, \gamma_2, \ldots$, 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 $\gamma_0, \gamma_1, \gamma_2, \ldots$ 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 $\gamma_0, \gamma_1, \ldots$ 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 $\gamma_0, \gamma_1, \ldots, \gamma_k$ 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 γ_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 $\sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0]$?

Derivation Sequences: Example

- What is the final state if statement

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

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

$$
\begin{aligned}
&\langle (\mathtt{z:=x;\ x:=y});\ \mathtt{y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0]\rangle \\
&\rightarrow^1_1 \langle \mathtt{x:=y;\ y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5]\rangle \\
&\rightarrow^1_1 \langle \mathtt{y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5][\mathtt{x} \mapsto 7]\rangle \\
&\rightarrow^1_1 \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5][\mathtt{x} \mapsto 7][\mathtt{y} \mapsto 5] \\
&= \sigma_{zero}[\mathtt{x} \mapsto 7][\mathtt{y} \mapsto 5][\mathtt{z} \mapsto 5]
\end{aligned}
$$

Derivation Sequences: Example

- What is the final state if statement

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

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

$$
\begin{aligned}
&\langle (\mathtt{z:=x;\ x:=y});\ \mathtt{y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0]\rangle \\
&\rightarrow^1_1 \langle \mathtt{x:=y;\ y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5]\rangle \\
&\rightarrow^1_1 \langle \mathtt{y:=z}, \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5][\mathtt{x} \mapsto 7]\rangle \\
&\rightarrow^1_1 \sigma_{zero}[\mathtt{x} \mapsto 5][\mathtt{y} \mapsto 7][\mathtt{z} \mapsto 0][\mathtt{z} \mapsto 5][\mathtt{x} \mapsto 7][\mathtt{y} \mapsto 5] \\
&= \sigma_{zero}[\mathtt{x} \mapsto 7][\mathtt{y} \mapsto 5][\mathtt{z} \mapsto 5]
\end{aligned}
$$

- 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 Trees

- Derivation trees explain why single-step transitions take place

- For the first step

$$\langle (\texttt{z:=x;}\ \ \texttt{x:=y});\ \ \texttt{y:=z}, \sigma\rangle \rightarrow_1^1 \langle \texttt{x:=y;}\ \ \texttt{y:=z}, \sigma[\texttt{z} \mapsto 5]\rangle$$

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

  the derivation tree is

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

- Note: if our statement were instead $\texttt{z:=x;}\ \ (\texttt{x:=y;}\ \ \texttt{y:=z})$, the corresponding first transition would have a simpler derivation tree with only two rule applications ($\textsc{Ass}_{sos}$ and $\textsc{Seq1}_{sos}$)

Derivation Sequences and Trees

- Natural (big-step) semantics
  - The execution of a statement 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 σ
  - 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 σ
  - 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 # 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 σ

3. Operational Semantics

Proving Properties of Derivation Sequences

- A finite derivation sequence, has a length which is a natural number.
  - Recall: a finite derivation sequence $\gamma_0, \gamma_1, \ldots, \gamma_n$ 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 $\forall k' < k.\ P(k')$

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

$$P(k) \equiv \forall s_1, s_2, \sigma, \sigma''.\ \langle s_1 ; s_2, \sigma\rangle \to_1^k \sigma'' \Rightarrow \exists \sigma', k_1, k_2.\ \langle s_1, \sigma\rangle \to_1^{k_1} \sigma' \wedge \langle s_2, \sigma'\rangle \to_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$$

- 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 $\langle s_1 ; s_2, \sigma\rangle \to_1^k \sigma''$ and need to show $\exists \sigma', k_1, k_2.\ \langle s_1, \sigma\rangle \to_1^{k_1} \sigma' \wedge \langle s_2, \sigma'\rangle \to_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$
- Consider the case $k = 0$. This immediately gives a contradiction, since, by the definition of $\to_1^0$ we conclude $\langle s_1 ; s_2, \sigma\rangle = \sigma''$
- Consider the case $k > 0$. Then the execution $\langle s_1 ; s_2, \sigma\rangle \to_1^k \sigma''$ can be split up as: $\langle s_1 ; s_2, \sigma\rangle \to_1^1 \gamma \to_1^{k-1} \sigma''$ for some configuration γ

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_{SOS} - recall:)

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

- Case 2: (rule SEQ2_{SOS} - recall:)

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

Proof: Case 1

- From the form of the rule $\textsc{Seq1}_{SOS}$ we obtain, for some state σ_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: Case 2

- From the form of the rule $\textsc{Seq2}_{SOS}$ we obtain, for some s_1' and σ_1:
  $\vdash \langle s_1, \sigma \rangle \to_1 \langle s_1', \sigma_1 \rangle$ and $\gamma = \langle s_1'; s_2, \sigma_1 \rangle$
- Thus, we have $\langle s_1, \sigma \rangle \to_1^1 \langle s_1', \sigma_1 \rangle$
- From $\gamma \to_1^{k-1} \sigma''$ we also get $\langle s_1'; s_2, \sigma_1 \rangle \to_1^{k-1} \sigma''$
- By applying the induction hypothesis (since $k - 1 < k$), we get
  $\exists \sigma_2, l_1, l_2 : \langle s_1', \sigma_1 \rangle \to_1^{l_1} \sigma_2 \wedge \langle s_2, \sigma_2 \rangle \to_1^{l_2} \sigma'' \wedge l_1 + l_2 = k - 1$
- From

  $\langle s_1, \sigma \rangle \to_1^1 \langle s_1', \sigma_1 \rangle$ and $\langle s_1', \sigma_1 \rangle \to_1^{l_1} \sigma_2$

  we get $\langle s_1, \sigma \rangle \to_1^{l_1+1} \sigma_2$
- This, by taking $\sigma' = \sigma_2$, $k_1 = (l_1 + 1)$ and $k_2 = l_2$, we obtain the required result: $\exists \sigma', k_1, k_2.\ \langle s_1, \sigma \rangle \to_1^{k_1} \sigma' \wedge \langle s_2, \sigma' \rangle \to_1^{k_2} \sigma'' \wedge k_1 + k_2 = k$

Semantic Equivalence

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

- for all stuck or terminal configurations γ, 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

Determinism

Lemma: The small-step semantics of IMP is deterministic.
That is, for all s,σ,γ, and γ' we have that
$\vdash \langle s, \sigma \rangle \to_1 \gamma \ \wedge \ \vdash \langle s, \sigma \rangle \to_1 \gamma' \ \Rightarrow \ \gamma = \gamma'$

- The proof runs by induction on the shape of the derivation tree for the transition $\langle s, \sigma \rangle \to_1 \gamma$

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

3. Operational Semantics

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:

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

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

Artificial End Marker

- We extend the syntactic category Stm with a restore statement

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

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

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

$$\frac{}{\langle \texttt{restore}\ (\underline{x}, \underline{v}), \underline{\sigma}\rangle \to_1 \underline{\sigma}[\underline{x} \mapsto \underline{v}]}\ (\textsc{Ret}_{sos})$$

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

Abort Statement

- Statement `abort` stops the execution of the complete program
- Aborting is modeled by ensuring that the configurations $\langle \texttt{abort}, \sigma \rangle$ are stuck
- There is no additional rule for `abort` in the structural operational semantics
- `abort` and `skip` are not semantically equivalent
  - $\langle \texttt{abort}, \sigma \rangle$ is the only derivation sequence for `abort` starting in σ
  - $\langle \texttt{skip}, \sigma \rangle \rightarrow_1^1 \sigma$ is the only derivation sequence for `skip` starting in σ
  - `skip` terminates successfully in all states, whereas `abort` terminates in all states, but not successfully

Abort: Observations

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

$$\langle \texttt{while true do skip end}, \sigma \rangle \rightarrow_1^1$$
$$\langle \texttt{if true then skip;while true do skip end end}, \sigma \rangle \rightarrow_1^1$$
$$\langle \texttt{skip;while true do skip end}, \sigma \rangle \rightarrow_1^1$$
$$\langle \texttt{while true do skip end}, \sigma \rangle \rightarrow_1^1 \ldots$$

- 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

Non-determinism

- For the statement $s \,[\!]\, s'$ either s or s' is non-deterministically chosen to be executed
- The statement

  ```
  x:=1 [] (x:=2; x:=x+2)
  ```

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

$$\frac{}{\langle \underline{s} \,[\!]\, \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}, \underline{\sigma} \rangle} \; (\mathrm{ND1}_{SOS})$$

$$\frac{}{\langle \underline{s} \,[\!]\, \underline{s}', \underline{\sigma} \rangle \rightarrow_1 \langle \underline{s}', \underline{\sigma} \rangle} \; (\mathrm{ND2}_{SOS})$$

Non-determinism: Observations

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

- There are also two derivation sequences for $\langle \texttt{while true do skip end} \,\square\, (\texttt{x:=2; x:=x+2}), \sigma \rangle$
  - a finite derivation sequence leading to $\sigma[\texttt{x} \mapsto 4]$
  - an infinite derivation sequence

- A 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

Parallelism

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

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to_1 \langle \underline{s}'' \text{ par } \underline{s}', \underline{\sigma}' \rangle} \; (\text{PAR1}_{SOS})$$

$$\frac{\langle \underline{s}, \underline{\sigma} \rangle \to_1 \underline{\sigma}'}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to_1 \langle \underline{s}', \underline{\sigma}' \rangle} \; (\text{PAR2}_{SOS})$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \to_1 \langle \underline{s}'', \underline{\sigma}' \rangle}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to_1 \langle \underline{s} \text{ par } \underline{s}'', \underline{\sigma}' \rangle} \; (\text{PAR3}_{SOS})$$

$$\frac{\langle \underline{s}', \underline{\sigma} \rangle \to_1 \underline{\sigma}'}{\langle \underline{s} \text{ par } \underline{s}', \underline{\sigma} \rangle \to_1 \langle \underline{s}, \underline{\sigma}' \rangle} \; (\text{PAR4}_{SOS})$$

Example: Interleaving

- The statement

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

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

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

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

Example: Derivation Sequences

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

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

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

Comparison: Summary

Natural Semantics	Structural Operational Semantics
• Local variable declarations and procedures can be modeled easily	• Local variable declarations (and procedures) require an explicit encoding of the original state
• No distinction between aborting and running forever	• Distinction between aborting and running forever
• Non-determinism suppresses non-termination (when possible)	• Non-determinism does not suppress non-termination
• Interleaving parallelism cannot be modeled	• Interleaving parallelism can be modeled

3. Operational Semantics

Semantic Functions

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

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

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

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

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

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

Equivalence Theorem

Theorem: For every statement s of IMP,
$\mathcal{S}_{NS}[\![s]\!]$ 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 σ and σ' we have $\vdash \langle s, \sigma \rangle \to \sigma' \Rightarrow \langle s, \sigma \rangle \to_1^* \sigma'$

- If the execution of s from σ 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 $\langle s, \sigma \rangle \to \sigma'$ (see exercises)

Equivalence Lemma 2

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

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

Equivalence Theorem: Proof

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

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

- The combination of the two Equivalence Lemmas shows (for all s, σ, σ') 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: 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)