Semantics of Programming Languages

Operational Semantics

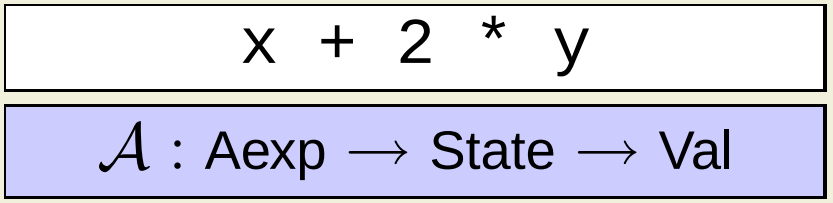
Prof. Peter Müller

Software Component Technology

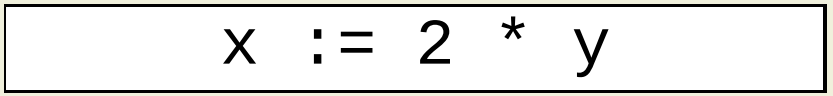
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Operational Semantics of Statements

- Evaluation of an expression in a state yields a value



- Execution of a statement modifies the state



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

Big-Step and Small-Step Semantics

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

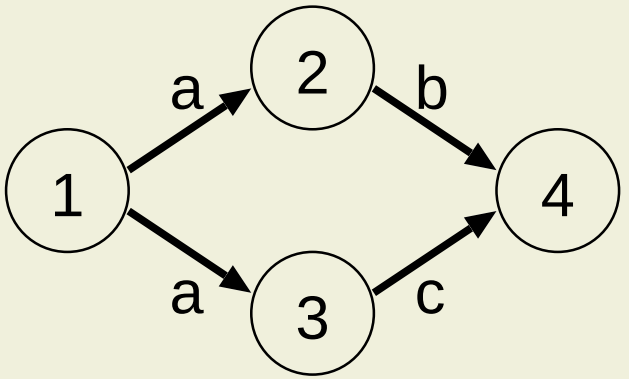
2. Operational Semantics

Transition Systems

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

- Example: Finite automaton

$$\Gamma = \{\langle w, S\rangle \mid w \in \{a,b,c\}^*, S \in \{1,2,3,4\}\}$$
$$T = \{\langle \epsilon, S\rangle \mid S \in \{1,2,3,4\}\}$$
$$\triangleright = \{(\langle aw, 1\rangle \to \langle w, 2\rangle), (\langle aw, 1\rangle \to \langle w, 3\rangle),$$
$$(\langle bw, 2\rangle \to \langle w, 4\rangle), (\langle cw, 3\rangle \to \langle w, 4\rangle)\}$$



Transitions in Natural Semantics

- Two types of configurations for operational semantics
  1. $\langle s, \sigma \rangle$, which represents that the statement s is to be executed in state σ
  2. σ, which represents a terminal state
- The transition relation $\rightarrow$ describes how executions take place
  - Typical transition: $\langle s, \sigma \rangle \rightarrow \sigma'$
  - Example: $\langle \texttt{skip}, \sigma \rangle \rightarrow \sigma$

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

Rules

- Transition relation is specified by rules

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

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

- Meaning of the rule

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

- Terminology
  - $\varphi_1, \ldots, \varphi_n$ are called **premises**
  - ψ is called **conclusion**
  - A rule without premises is called **axiom**

Notation

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

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

Natural Semantics of IMP

- `skip` does not modify the state

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

- $x\texttt{:=}e$ assigns the value of e to variable e

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

- Sequential composition $s_1\texttt{;}s_2$
  - First, s_1 is executed in state σ, leading to σ'
  - Then s_2 is executed in state σ'

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

Natural Semantics of IMP (cont'd)

- Conditional statement $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}$
  - If b holds, s_1 is executed
  - If b does not hold, s_2 is executed

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

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

Natural Semantics of IMP (cont'd)

- Loop statement `while` b `do` s `end`
  - If b holds, s is executed once, leading to state σ'
  - Then the whole while-statement is executed again σ'

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

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

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

Rule Instantiations

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

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

- Assignment rule **instance**

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

Derivations: Example

- What is the final state if statement

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

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

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

Derivation Trees

- Rule instances can be combined to derive a transition $\langle s, \sigma \rangle \rightarrow \sigma'$
- The result is a **derivation tree**
  - The root is the transition $\langle s, \sigma \rangle \rightarrow \sigma'$
  - The leaves are axiom instances
  - The internal nodes are conclusions of rule instances and have the corresponding premises as immediate children
- The conditions of all instantiated rules must be satisfied
- There can be several derivations for one transition (non-deterministic semantics)

Termination

- The execution of a statement s in state σ
  - **terminates** iff there is a state σ' such that $\langle s, \sigma \rangle \rightarrow \sigma'$
  - **loops** iff there is no state σ' such that $\langle s, \sigma \rangle \rightarrow \sigma'$

- A statement s
  - **always terminates** if the execution in a state σ terminates for all choices of σ
  - **always loops** if the execution in a state σ loops for all choices of σ

2. Operational Semantics

2.1 Big-Step Semantics

2.1.1 Natural Semantics of IMP

2.1.2 Properties of the Semantics

2.1.3 Extensions of IMP

2.2 Small-Step Semantics

2.3 Equivalence

2.4 Applications of Operational Semantics

Semantic Equivalence

- Definition

  Two statements s_1 and s_2 are **semantically equivalent** (denoted by $s_1 \equiv s_2$) if the following property holds for all states σ, σ':

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

- Example

  `while` b `do` s `end` $\equiv$
  `if` b `then` s`;` `while` b `do` s `end`

Unfolding Loops in C, C++, and Java

```
int i = 0;
while(i < 2 ) {


  while(i < 1)
    if(i == 0) break;

  i = i + 1;
}

printf("i = %d", i);
```

```
i = 2
```

```
int i = 0;
while(i < 2 ) {
  if(i < 1) {
    if(i == 0) break;
    while(i < 1)
      if(i == 0) break;
  }
  i = i + 1;
}

printf("i = %d", i);
```

```
i = 0
```

- Equivalence does not hold in these languages

Unfolding Loops in IMP

- We prove the equivalence based on the natural semantics

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

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

Proof: Case "$\Rightarrow$"

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

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

we know

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

Proof: Case "$\Rightarrow$" (cont'd)

- We can construct the derivation tree

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

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

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

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

Proof: Case "$\Rightarrow$" (cont'd)

- For the case

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

  we know

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

- We can construct the derivation tree

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

- We have a derivation tree for $(**)$, which completes Case "$\Rightarrow$"

Induction on Derivations

Induction on the shape of derivation trees

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

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

Using Induction on Derivations

Lemma: The natural semantics of IMP is deterministic

- We prove

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

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

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

Induction Base

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

Induction Step: Seq. Composition

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

Induction Step: `if`

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

Induction Step: `while`

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

2. Operational Semantics

2.1 Big-Step Semantics

2.1.1 Natural Semantics of IMP

2.1.2 Properties of the Semantics

2.1.3 Extensions of IMP

2.2 Small-Step Semantics

2.3 Equivalence

2.4 Applications of Operational Semantics

Local Variable Declarations

- Statement `var` x`:=`e `in` s `end` declares a new variable that is visible in the statement sequence of the declaration, s (block)
- Semantics
  - Expression e is evaluated in the initial state
  - Statement s is executed in a state in which x has the value of e
  - After the execution of s, the initial value of x is restored
- Rule

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

Procedure Declarations and Calls

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

Procedures: Example

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

Vector Notation

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

Natural Semantics of Procedure Calls

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

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

Abortion

- Statement `abort` stops the execution of the complete program
- Abortion is modeled in the operational semantics by ensuring that the configurations $\langle \texttt{abort}, \sigma \rangle$ are **stuck**
- There is no additional rule for `abort` in the natural semantics

Abortion: Observations

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

- Natural semantics cannot distinguish between **looping** and **abnormal termination**
  - Natural semantics is only concerned with programs that terminate normally
  - Abortion could be modeled by “normal termination” in a special error configuration

Non-determinism

- For the statement $s_1 \,[]\, s_2$ either s_1 or s_2 is non-deterministically chosen to be executed
- The statement

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

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

$$\frac{\langle s_1, \sigma\rangle \rightarrow \sigma'}{\langle s_1 \,[]\, s_2, \sigma\rangle \rightarrow \sigma'} \qquad \frac{\langle s_2, \sigma\rangle \rightarrow \sigma'}{\langle s_1 \,[]\, s_2, \sigma\rangle \rightarrow \sigma'}$$

Non-determinism: Observations

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

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

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

Parallelism

- For the statement s_1 `par` s_2 both statements s_1 and s_2 are executed, but execution can be **interleaved**
- The statement

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

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

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

Parallelism: Observations

- Attempt to define rules

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

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

- Rules do not allow interleaving execution
- In a natural semantics the execution of the immediate constituents is an **atomic entity** so we cannot express interleaving of computations

Problems of Natural Semantics

- Properties of looping programs cannot be expressed
- No distinction between abortion and looping
- Non-determinism suppresses looping (if possible)
- Parallelism cannot be modeled
- Definition of equivalence is too coarse
  - All sorting programs are equivalent
  - All looping programs are equivalent