Formal Techniques for Software Engineering: A Simple Imperative Programming language

Rocco De Nicola

IMT Institute for Advanced Studies, Lucca
rocco.denicola@imtlucca.it

April 2013

Lesson 3

The **While** language

- **While** is a very simple imperative programming language
- Specifically devised for illustrating the different approaches to program semantics

For **While** shall study:

- BNF syntax
- Operational Semantics
  - Natural semantics (or *big-step* semantics)
  - Structural Operational Semantics (or *small-step* semantics)
- Denotational semantics

The **While** language: syntax

Meta-variables and syntactic categories

- n will range over numerals, **Num**
- $\mathtt{x}$ will range over variables, **Var**
- a will range over arithmetic expressions, **Aexp**
- b will range over boolean expressions, **Bexp**
- S will range over statements, **Stm**

Syntax

$$a \ ::= \ n \ \mid \ \mathtt{x} \ \mid \ a_1 + a_2 \ \mid \ a_1 \star a_2 \ \mid \ a_1 - a_2$$

$$b \ ::= \ \mathtt{true} \ \mid \ \mathtt{false} \ \mid \ a_1 = a_2 \ \mid \ a_1 \leq a_2 \ \mid \ \neg b \ \mid \ b_1 \wedge b_2$$

$$S \ ::= \ \mathtt{x} := a \ \mid \ \mathtt{skip} \ \mid \ S_1 \,;\, S_2 \ \mid \ \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2 \ \mid \ \mathtt{while}\ b\ \mathtt{do}\ S$$

Abstract syntax vs. concrete syntax

Abstract syntax

$$a \;::=\; n \mid \mathtt{x} \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

$$b \;::=\; \mathtt{true} \mid \mathtt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$

$$S \;::=\; \mathtt{x} := a \mid \mathtt{skip} \mid S_1 \,;\, S_2 \mid \mathtt{if}\; b\; \mathtt{then}\; S_1\; \mathtt{else}\; S_2 \mid \mathtt{while}\; b\; \mathtt{do}\; S$$

- The structure of numerals and variables is assumed as given
- A syntactic term could be generated by more than one syntax tree
- Concrete syntax permits deriving unique trees but is definitely more cumbersome than abstract syntax
- to resolve ambiguities, we use:
  - brackets (. . .)
  - operator precedences ($+$ binds more than $\star$, . . . ,)

Abstract syntax vs. concrete syntax

Abstract syntax

$$a \;::=\; n \mid \mathtt{x} \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$$

$$b \;::=\; \mathtt{true} \mid \mathtt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$$

$$S \;::=\; \mathtt{x} := a \mid \mathtt{skip} \mid S_1 \,;\, S_2 \mid \mathtt{if}\; b \;\mathtt{then}\; S_1 \;\mathtt{else}\; S_2 \mid \mathtt{while}\; b \;\mathtt{do}\; S$$

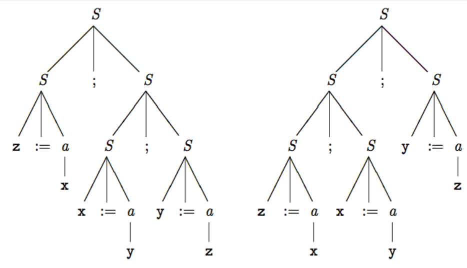
- The structure of numerals and variables is assumed as given
- A syntactic term could be generated by more than one syntax tree
- Concrete syntax permits deriving unique trees but is definitely more cumbersome than abstract syntax
- to resolve ambiguities, we use:
  - brackets (...)
  - operator precedences ($+$ binds more than $\star$, ...,)

Abstract syntax vs. concrete syntax

$$
\begin{array}{lll}
a & ::= & n \mid \texttt{x} \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & \texttt{x} := a \mid \texttt{skip} \mid S_1 \,;\, S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \\
 & \mid & \texttt{while}\ b\ \texttt{do}\ S
\end{array}
$$

More than one syntax tree could correspond to a program:

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



Semantics of **While**

The semantics of **While** is given by defining semantic relations for each of the syntactic categories

For each syntactic category the semantics of its terms is defined **compositionally**, i.e. there is a semantic clause:

- for each basic elements of the syntactic category
- for each construct for building composite elements

The semantics of composite elements is defined in terms of the semantics of the direct components.

- The operational and denotational approaches specify semantic relations for the statements of **While**
- The semantic functions for numerals and arithmetic or boolean expressions are specified once and for all.

Natural Numbers

In the rest of the lectures the structure of numerals will be left unspecified
$\Longrightarrow$ the semantic function for numerals is unspecified too

An example of numerals definition

Numerals in the *binary* system:

$$n \quad ::= \quad 0 \quad | \quad 1 \quad | \quad n\,0 \quad | \quad n\,1$$

Semantics Function for numerals

Semantics of numerals is defined by function

$$\mathcal{N} \; : \; \mathbf{Num} \rightarrow \mathbf{Z}$$

Natural Numbers

In the rest of the lectures the structure of numerals will be left unspecified
$\Longrightarrow$ the semantic function for numerals is unspecified too

An example of numerals definition

Numerals in the *binary* system:

$$n \quad ::= \quad 0 \quad | \quad 1 \quad | \quad n\,0 \quad | \quad n\,1$$

Semantics Function for numerals

Semantics of numerals is defined by function

$$\mathcal{N} \; : \; \mathbf{Num} \rightarrow \mathbf{Z}$$

Natural Numbers

In the rest of the lectures the structure of numerals will be left unspecified
$\Longrightarrow$ the semantic function for numerals is unspecified too

An example of numerals definition

Numerals in the *binary* system:

$$n \quad ::= \quad \mathtt{0} \quad | \quad \mathtt{1} \quad | \quad n\,\mathtt{0} \quad | \quad n\,\mathtt{1}$$

Semantics Function for numerals

Semantics of numerals is defined by function

$$\mathcal{N} \; : \; \mathbf{Num} \rightarrow \mathbf{Z}$$

Natural Numbers

In the rest of the lectures the structure of numerals will be left unspecified
$\Longrightarrow$ the semantic function for numerals is unspecified too

An example of numerals definition

Numerals in the *binary* system:

$$n \quad ::= \quad \mathtt{0} \quad | \quad \mathtt{1} \quad | \quad n\,\mathtt{0} \quad | \quad n\,\mathtt{1}$$

Semantics Function for numerals

Semantics of numerals is defined by function

$$\mathcal{N} \;:\; \mathbf{Num} \rightarrow \mathbf{Z}$$

Semantics of Binary Numerals

$\mathcal{N} : \mathbf{Num} \to \mathbf{Z}$

$$
\begin{aligned}
\mathcal{N}[\![0]\!] &= \mathbf{0} \\
\mathcal{N}[\![1]\!] &= \mathbf{1} \\
\mathcal{N}[\![n\ 0]\!] &= \mathbf{2} \cdot \mathcal{N}[\![n]\!] \\
\mathcal{N}[\![n\ 1]\!] &= \mathbf{2} \cdot \mathcal{N}[\![n]\!] + \mathbf{1}
\end{aligned}
$$

- $\mathbf{0}$ and $\mathbf{1}$ are elements of $\mathbf{Z}$
- $+$ and $\cdot$ are arithmetic operations in $\mathbf{Z}$.

Expression variables and state

The semantics of an expression that contains variables depends on the values of such variables.

The State Function

The Function $\mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}$ associates to each variable its current value

Notation

Function $\mathbf{State}$ is written as collection of pairs of the form $\mathtt{x} \mapsto n$:

$$[\mathtt{x} \mapsto \mathtt{5}, \mathtt{y} \mapsto \mathtt{7}, \mathtt{z} \mapsto \mathtt{0}]$$

State Update

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

Arithmetic expressions

The semantics of arithmetic expressions is defined by function

$$\mathcal{A} : \mathbf{Aexp} \rightarrow (\ \mathbf{State} \rightarrow \mathbf{Z}\ )$$

that takes its parameters *one at a time*:

- When an arithmetic expression a is provided, function $\mathcal{A}[\![\ a\ ]\!]$ is obtained; i.e. a function $\mathbf{State} \rightarrow \mathbf{Z}$
- The value of a is obtained when a state s is provided.

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \cdot \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

Boolean expressions

Semantics of boolean expressions is defined by function

$$\mathcal{B} : \mathbf{Bexp} \rightarrow (\mathbf{State} \rightarrow \{\mathbf{tt}, \mathbf{ff}\})$$

$$\begin{aligned}
\mathcal{B}[\![\texttt{false}]\!]s &= \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\, b]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s &= \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{aligned}$$

Operational Semantics

The Operational semantics describes an abstraction of how a program is executed on a machine and ignores

- the use of registers
- the actual address of variables
- machine architectures
- implementation strategies

Small-step vs. big-step semantics

- Small-step describes how *individual steps* of computations take place and hides many execution details
- Big-step describes how the *overall* results of executions are obtained and hides even more execution details

Small-step vs. big-step semantics

Small-steps Semantics

Structural Operational Semantics (SOS) describes how *individual steps* of computations take place:

$$
\begin{array}{lrl}
 & \langle z{:=}x;\ x{:=}y;\ y{:=}z, & [x{\mapsto}\mathbf{5},\ y{\mapsto}\mathbf{7},\ z{\mapsto}\mathbf{0}]\rangle \\
\Rightarrow & \langle x{:=}y;\ y{:=}z, & [x{\mapsto}\mathbf{5},\ y{\mapsto}\mathbf{7},\ z{\mapsto}\mathbf{5}]\rangle \\
\Rightarrow & \langle y{:=}z, & [x{\mapsto}\mathbf{7},\ y{\mapsto}\mathbf{7},\ z{\mapsto}\mathbf{5}]\rangle \\
\Rightarrow & & [x{\mapsto}\mathbf{7},\ y{\mapsto}\mathbf{5},\ z{\mapsto}\mathbf{5}]
\end{array}
$$

Big-step Semantics

Natural Semantics describes how the *overall* results of executions are obtained:

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

Natural Semantics of **While**

Transition relation

The semantics defines the relationship between the *initial* and the *final* state of an execution. For each statement S

$$\langle S, s \rangle \rightarrow s'$$

specifies the relationship between the initial state s and the final state s'

Note

- The intuitive meaning of a step is that the execution of S from s will terminate and the resulting state will be s'
- The rôle of a **While** statement is to change the state
  - Expressions only **inspect** the state
  - statements inspect and **modify** the state

Natural Semantics: transition relation

$[\text{ass}_{\text{ns}}]$	$\langle x := a,\ s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$
$[\text{skip}_{\text{ns}}]$	$\langle \texttt{skip},\ s\rangle \to s$
$[\text{comp}_{\text{ns}}]$	$\dfrac{\langle S_1,\ s\rangle \to s',\ \langle S_2,\ s'\rangle \to s''}{\langle S_1;S_2,\ s\rangle \to s''}$
$[\text{if}^{\text{tt}}_{\text{ns}}]$	$\dfrac{\langle S_1,\ s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \to s'}$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$
$[\text{if}^{\text{ff}}_{\text{ns}}]$	$\dfrac{\langle S_2,\ s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \to s'}$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$
$[\text{while}^{\text{tt}}_{\text{ns}}]$	$\dfrac{\langle S,\ s\rangle \to s',\ \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''}$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$
$[\text{while}^{\text{ff}}_{\text{ns}}]$	$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$

Natural Semantics: examples

Two exercises

Determine the progress of the two programs below

- $(\mathtt{z} := \mathtt{x}\,;\ \mathtt{x} := \mathtt{y})\,;\ \mathtt{y} := \mathtt{z}$ with $[\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]$

- Factorial program $(\mathtt{x}!)$
  $\mathtt{y} := 1\,;\ \mathtt{while}\ \neg(\mathtt{x} = 1)\ \mathtt{do}\ (\mathtt{y} := \mathtt{y} \star \mathtt{x}\,;\ \mathtt{x} := \mathtt{x} - 1)$ with $[\mathtt{x} \mapsto 3]$

according to the natural semantics for **While**.

Natural Semantics: semantical equivalence

Example

Consider

$$\texttt{while}\ b\ \texttt{do}\ S \quad \text{and} \quad \texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip}$$

- They are syntactically different
- Are they semantically equivalent?

Semantical equivalence

Two statements S_1 and S_2 are **semantically equivalent** if for all states s and s'

$$\langle S_1\,,s\rangle \rightarrow s' \qquad \text{if and only if} \qquad \langle S_2\,,s\rangle \rightarrow s'$$

Natural Semantics: semantical equivalence

Example

Consider

$$\texttt{while}\ b\ \texttt{do}\ S \quad \text{and} \quad \texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip}$$

- They are syntactically different
- Are they semantically equivalent?
- Do we have: if $\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s''$ then $\langle \texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip}, s\rangle \to s''$

and viceversa?

Semantical equivalence

Two statements S_1 and S_2 are **semantically equivalent** if for all states s and s'

$$\langle S_1, s\rangle \to s' \qquad \text{if and only if} \qquad \langle S_2, s\rangle \to s'$$

Natural Semantics: deterministic semantics

A natural semantics is deterministic if for all S, s, s' and s''

$$\langle S, s\rangle \rightarrow s' \text{ and } \langle S, s\rangle \rightarrow s'' \quad \text{imply} \quad s' = s''$$

Theorem

The natural semantics of **While** is deterministic

Natural Semantics: deterministic semantics

A natural semantics is deterministic if for all S, s, s' and s''

$$\langle S, s\rangle \to s' \text{ and } \langle S, s\rangle \to s'' \quad \text{imply} \quad s' = s''$$

Theorem

The natural semantics of **While** is deterministic

Natural Semantics: deterministic semantics

A natural semantics is deterministic if for all S, s, s' and s''

$$\langle S , s\rangle \rightarrow s' \text{ and } \langle S , s\rangle \rightarrow s'' \quad \text{imply} \quad s' = s''$$

Theorem

The natural semantics of **While** is deterministic

Proof technique: induction on the depth of the derivation trees

1. Prove that the property holds for all the simple derivation trees by showing that it holds for the **axioms** of the transition rules
2. Prove that the property holds for all composite derivation trees: For each rule assume that the property holds for its premises (*induction hypothesis*) and prove that it also holds for the conclusion of the rule provided that the conditions of the rule are satisfied

Natural Semantics: deterministic semantics

A natural semantics is deterministic if for all S, s, s' and s''

$$\langle S, s\rangle \to s' \text{ and } \langle S, s\rangle \to s'' \qquad \text{imply} \qquad s' = s''$$

Theorem

The natural semantics of **While** is deterministic

Proof (sketch). Some cases:

- $[ass_{ns}]$
- $[comp_{ns}]$

Natural Semantics: deterministic semantics

A natural semantics is deterministic if for all S, s, s' and s''

$$\langle S, s\rangle \to s' \text{ and } \langle S, s\rangle \to s'' \quad \text{imply} \quad s' = s''$$

Theorem

The natural semantics of **While** is deterministic

Remark

We cannot use structural induction on the statement S when proving the theorem because the natural semantics of $\texttt{while}\ b\ \texttt{do}\ S$ is defined in terms of itself (rule [while_{tt}]).Not well founded

Natural Semantics: statements meaning

Meanings as functions

The **meaning** of **While** statements according to the natural semantics is the (partial) function

$$\mathcal{S}_{\text{ns}}\colon \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_{\text{ns}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\, s\rangle \to s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

Why is it a partial function?

Because, e.g., the statement

```
while true do skip
```

Natural Semantics: statements meaning

Meanings as functions

The **meaning** of **While** statements according to the natural semantics is the (partial) function

$$\mathcal{S}_{\mathrm{ns}}\colon \mathbf{Stm} \rightarrow (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_{\mathrm{ns}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\, s\rangle \rightarrow s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

Why is it a partial function?

Because, e.g., the statement

```
while true do skip
```

Structural Operational Semantics (SOS)

The emphasis of SOS semantics is on the individual steps of the execution

- the execution of assignments
- the execution of tests

Transition relation

$\langle S, s \rangle \Rightarrow \gamma$ expresses the *first step* of the execution of S from state s and there are two possible outcomes:

- $\gamma = \langle S', s' \rangle$, i.e. an *intermediate* configuration
- $\gamma = s'$, i.e. it is a *final* state

A *derivation sequence* of a statement is either

- a *finite* sequence of transitions, or
- an *infinite* sequence of transitions

Structural Operational Semantics: transition relation

$[\text{ass}_{\text{sos}}]$	$\langle x := a,\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$
$[\text{skip}_{\text{sos}}]$	$\langle \texttt{skip},\ s\rangle \Rightarrow s$
$[\text{comp}^1_{\text{sos}}]$	$\dfrac{\langle S_1,\ s\rangle \Rightarrow \langle S'_1,\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S'_1;S_2,\ s'\rangle}$
$[\text{comp}^2_{\text{sos}}]$	$\dfrac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$
$[\text{if}^{\text{tt}}_{\text{sos}}]$	$\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \mathbf{tt}$
$[\text{if}^{\text{ff}}_{\text{sos}}]$	$\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle$ if $\mathcal{B}[\![b]\!]s = \mathbf{ff}$
$[\text{while}_{\text{sos}}]$	$\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \Rightarrow$ $\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip},\ s\rangle$

Structural Operational Semantics: examples

Two exercises

Determine the progress of the two programs below

- $(\mathtt{z} := \mathtt{x}\,;\ \mathtt{x} := \mathtt{y})\,;\ \mathtt{y} := \mathtt{z}$ with $[\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]$
- Factorial program ($\mathtt{x}$!)
  $\mathtt{y} := 1\,;\ \mathtt{while}\ \neg(\mathtt{x} = 1)\ \mathtt{do}\ (\mathtt{y} := \mathtt{y} \star \mathtt{x}\,;\ \mathtt{x} := \mathtt{x} - 1)$ with $[\mathtt{x} \mapsto 3]$

according to the SOS semantics for **While**.

Structural Operational Semantics: proof by induction

Proof technique: induction on the length of the derivation sequences

For SOS, it is useful to conduct proofs by induction on the lengths of the considered finite derivation sequences

1. Prove that the property holds for all derivation sequences of length 0
2. Prove that the property holds for all finite derivation sequences: Assume that the property holds for all derivation sequences of length at most k (*induction hypothesis*) and show that it holds for derivation sequences of length $k + 1$

Lemma

If $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$, then $\exists\ s', k_1, k_2$ such that

$$k = k_1 + k_2 \text{ and } \langle\ S_1\ ,\ s\ \rangle \Rightarrow^{k_1}\ s' \quad \text{and} \quad \langle\ S_2\ ,\ s'\rangle \Rightarrow^{k_2}\ s''$$

Proof (sketch). By induction on the length of the derivation sequence $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$ (i.e., by induction on the number k).

Structural Operational Semantics: proof by induction

Proof technique: induction on the length of the derivation sequences

For SOS, it is useful to conduct proofs by induction on the lengths of the considered finite derivation sequences

1. Prove that the property holds for all derivation sequences of length 0
2. Prove that the property holds for all finite derivation sequences: Assume that the property holds for all derivation sequences of length at most k (*induction hypothesis*) and show that it holds for derivation sequences of length $k + 1$

Lemma

If $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$, then $\exists\ s', k_1, k_2$ such that

$$k = k_1 + k_2 \text{ and } \langle\ S_1\ ,\ s\ \rangle \Rightarrow^{k_1}\ s' \quad \text{and} \quad \langle\ S_2\ ,\ s'\rangle \Rightarrow^{k_2}\ s''$$

Proof (sketch). By induction on the length of the derivation sequence $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$ (i.e., by induction on the number k).

Structural Operational Semantics: proof by induction

Proof technique: induction on the length of the derivation sequences

For SOS, it is useful to conduct proofs by induction on the lengths of the considered finite derivation sequences

1. Prove that the property holds for all derivation sequences of length 0
2. Prove that the property holds for all finite derivation sequences: Assume that the property holds for all derivation sequences of length at most k (*induction hypothesis*) and show that it holds for derivation sequences of length $k+1$

Lemma

If $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$, then $\exists\ s', k_1, k_2$ such that

$$k = k_1 + k_2 \text{ and } \langle\ S_1\ ,\ s\ \rangle \Rightarrow^{k_1}\ s' \quad \text{and} \quad \langle\ S_2\ ,\ s'\rangle \Rightarrow^{k_2}\ s''$$

Proof (sketch). By induction on the length of the derivation sequence $\langle\ S_1\ ;\ S_2\ ,\ s\ \rangle \Rightarrow^k\ s''$ (i.e., by induction on the number k).

Structural Operational Semantics: deterministic semantics

Definition

A structural operational semantics is deterministic if for all S, s, γ and γ' we have:

$$\langle S, s\rangle \Rightarrow \gamma \text{ and } \langle S, s\rangle \Rightarrow \gamma' \qquad \text{imply} \qquad \gamma = \gamma'$$

Theorem

The structural operational semantics of **While** is deterministic

Structural Operational Semantics: deterministic semantics

Definition

A structural operational semantics is deterministic if for all S, s, γ and γ' we have:

$$\langle S, s\rangle \Rightarrow \gamma \text{ and } \langle S, s\rangle \Rightarrow \gamma' \quad \text{imply} \quad \gamma = \gamma'$$

Theorem

The structural operational semantics of **While** is deterministic

SOS: Semantic Equivalence

Semantical equivalence

S_1 and S_2 are **semantically equivalent** if for all states s

- $\langle S_1, s\rangle \Rightarrow^* \gamma$ *iff* $\langle S_2, s\rangle \Rightarrow^* \gamma$ where γ is either stuck or terminal
- there is an infinite derivation sequence from $\langle S_1, s\rangle$ *iff* there is an infinite derivation sequence from $\langle S_2, s\rangle$

Example

$\texttt{while}\ b\ \texttt{do}\ S$ $\qquad$ $\texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip}$

- They are syntactically different
- Are they semantically equivalent?

SOS: Semantic Equivalence

Semantical equivalence

S_1 and S_2 are **semantically equivalent** if for all states s

- $\langle S_1, s\rangle \Rightarrow^* \gamma$ *iff* $\langle S_2, s\rangle \Rightarrow^* \gamma$ where γ is either stuck or terminal
- there is an infinite derivation sequence from $\langle S_1, s\rangle$
  iff there is an infinite derivation sequence from $\langle S_2, s\rangle$

Example

$\texttt{while}\ b\ \texttt{do}\ S \qquad \texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip}$

- They are syntactically different
- Are they semantically equivalent?

SOS: statements meaning

Meanings as functions

The **meaning** of **While** statements according to the SOS semantics is the (partial) function

$$\mathcal{S}_{\text{sos}}\colon \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\, s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

Operational Semantics: equivalence result

Theorem (Equivalence between nat. sem. and SOS)

For every statement S, we have

$$\mathcal{S}_{\mathrm{ns}}[\![\; S \;]\!] \;=\; \mathcal{S}_{\mathrm{sos}}[\![\; S \;]\!]$$

This result expresses two properties:

- If the execution of S from some state terminates in one of the semantics, then it also terminates in the other and the resulting states will be equal
- If the execution of S from some state loops in one of the semantics, then it will also loop in the other

Operational Semantics: equivalence result (proof)

Lemma 1

For every statement S and states s and s', we have

$$\langle S, s \rangle \rightarrow s' \quad \text{implies} \quad \langle S, s \rangle \Rightarrow^* s'$$

Proof (sketch). The proof proceeds by induction on the shape of the derivation tree for $\langle S, s \rangle \rightarrow s'$

Lemma 2

For every statement S and states s and s', and number k, we have

$$\langle S, s \rangle \Rightarrow^k s' \quad \text{implies} \quad \langle S, s \rangle \rightarrow s'$$

Proof (sketch). The proof proceeds by induction on k

The equivalence of the two semantics follows directly from Lemma 1 and 2.

Operational Semantics: equivalence result (proof)

Lemma 1

For every statement S and states s and s', we have

$$\langle S, s \rangle \to s' \quad \text{implies} \quad \langle S, s \rangle \Rightarrow^* s'$$

Proof (sketch). The proof proceeds by induction on the shape of the derivation tree for $\langle S, s \rangle \to s'$

Lemma 2

For every statement S and states s and s', and number k, we have

$$\langle S, s \rangle \Rightarrow^k s' \quad \text{implies} \quad \langle S, s \rangle \to s'$$

Proof (sketch). The proof proceeds by induction on k

The equivalence of the two semantics follows directly from Lemma 1 and 2.

Operational Semantics: equivalence result (proof)

Lemma 1

For every statement S and states s and s', we have

$$\langle S, s \rangle \rightarrow s' \quad \text{implies} \quad \langle S, s \rangle \Rightarrow^* s'$$

Proof (sketch). The proof proceeds by induction on the shape of the derivation tree for $\langle S, s \rangle \rightarrow s'$

Lemma 2

For every statement S and states s and s', and number k, we have

$$\langle S, s \rangle \Rightarrow^k s' \quad \text{implies} \quad \langle S, s \rangle \rightarrow s'$$

Proof (sketch). The proof proceeds by induction on k

The equivalence of the two semantics follows directly from Lemma 1 and 2.

Operational Semantics: equivalence result (proof)

Lemma 1

For every statement S and states s and s', we have

$$\langle S, s\rangle \to s' \quad \text{implies} \quad \langle S, s\rangle \Rightarrow^* s'$$

Proof (sketch). The proof proceeds by induction on the shape of the derivation tree for $\langle S, s\rangle \to s'$

Lemma 2

For every statement S and states s and s', and number k, we have

$$\langle S, s\rangle \Rightarrow^k s' \quad \text{implies} \quad \langle S, s\rangle \to s'$$

Proof (sketch). The proof proceeds by induction on k

The equivalence of the two semantics follows directly from Lemma 1 and 2.

Operational Semantics: equivalence result (proof)

Lemma 1

For every statement S and states s and s', we have

$$\langle S, s\rangle \rightarrow s' \quad \text{implies} \quad \langle S, s\rangle \Rightarrow^* s'$$

Proof (sketch). The proof proceeds by induction on the shape of the derivation tree for $\langle S, s\rangle \rightarrow s'$

Lemma 2

For every statement S and states s and s', and number k, we have

$$\langle S, s\rangle \Rightarrow^k s' \quad \text{implies} \quad \langle S, s\rangle \rightarrow s'$$

Proof (sketch). The proof proceeds by induction on k

The equivalence of the two semantics follows directly from Lemma 1 and 2.

Operational Semantics: Natural vs. SOS

Now we know that the two semantics are equivalent; we can ask

which one is better?

- It is is largely a matter of taste
- For some language constructs, it could be easy to specify the semantics in one style but difficult or even impossible in the other
- There are situations where equivalent semantics can be specified in the two styles but where one of the semantics is to be preferred because of a particular application

Operational Semantics: Natural vs. SOS

Now we know that the two semantics are equivalent; we can ask

which one is better?

- It is is largely a matter of taste
- For some language constructs, it could be easy to specify the semantics in one style but difficult or even impossible in the other
- There are situations where equivalent semantics can be specified in the two styles but where one of the semantics is to be preferred because of a particular application

Operational Semantics: Natural vs. SOS

Now we know that the two semantics are equivalent; we can ask

which one is better?

- It is is largely a matter of taste
- For some language constructs, it could be easy to specify the semantics in one style but difficult or even impossible in the other
- There are situations where equivalent semantics can be specified in the two styles but where one of the semantics is to be preferred because of a particular application

Operational Semantics: Natural vs. SOS

- Consider **While** + `abort`

`abort`

The statement `abort` stops the execution of the complete program

$\Rightarrow$ no rule is added to the two semantics

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, we cannot distinguish between *looping* (`while true do skip`) and *abnormal termination* (`abort`)
- In a SOS, *looping* is reflected by infinite derivation sequences and *abnormal termination* by finite ones ending in a stuck configuration

Workaround

Model abnormal termination by normal termination, but in a special error configuration.

Operational Semantics: Natural vs. SOS

- Consider **While** + `abort`

`abort`

The statement `abort` stops the execution of the complete program

$\Rightarrow$ no rule is added to the two semantics

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, we cannot distinguish between *looping* (`while true do skip`) and *abnormal termination* (`abort`)
- In a SOS, *looping* is reflected by infinite derivation sequences and *abnormal termination* by finite ones ending in a stuck configuration

Workaround

Model abnormal termination by normal termination, but in a special error configuration.

Operational Semantics: Natural vs. SOS

- Consider **While** + `abort`

`abort`

The statement `abort` stops the execution of the complete program

$\Rightarrow$ no rule is added to the two semantics

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, we cannot distinguish between *looping* (`while true do skip`) and *abnormal termination* (`abort`)
- In a SOS, *looping* is reflected by infinite derivation sequences and *abnormal termination* by finite ones ending in a stuck configuration

Workaround

Model abnormal termination by normal termination, but in a special error configuration.

Operational Semantics: Natural vs. SOS

- Consider **While** + `or`

Non-determinism

The statement S_1 `or` S_2 can non-deterministically choose to execute either S_1 or S_2

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, *non-determinism suppresses looping*, if possible (e.g., `(while true do skip) or (x := 2; x := x + 2)`)
- In a SOS, *non-determinism does not suppress looping*

Operational Semantics: Natural vs. SOS

- Consider **While** + `or`

Non-determinism

The statement S_1 `or` S_2 can non-deterministically choose to execute either S_1 or S_2

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, *non-determinism suppresses looping*, if possible (e.g., (`while true do skip`) `or` ($\mathtt{x} := 2\,;\ \mathtt{x} := \mathtt{x} + 2$))
- In a SOS, *non-determinism does not suppress looping*

Operational Semantics: Natural vs. SOS

- Consider **While** + `par`

Parallelism

The statement S_1 `par` S_2 can *interleave* the execution of S_1 and S_2

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, we cannot express interleaving of computations (indeed, the execution of the immediate constituents is an *atomic* entity)
- In a SOS, we can easily express interleaving (indeed, we concentrate on the small steps of the computation)

Operational Semantics: Natural vs. SOS

- Consider **While** + `par`

Parallelism

The statement S_1 `par` S_2 can *interleave* the execution of S_1 and S_2

Natural Semantics versus Structural Operational Semantics

- In the natural semantics, we cannot express interleaving of computations (indeed, the execution of the immediate constituents is an *atomic* entity)
- In a SOS, we can easily express interleaving (indeed, we concentrate on the small steps of the computation)