

# Semantics and Verification of Software

## Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)

Thomas Noll

Lehrstuhl für Informatik 2  
(Software Modeling and Verification)

RWTH Aachen University

`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/svsw08/`

Winter semester 2008/09

- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions
- 6 Execution of Statements

**WHILE**: simple imperative programming language without procedures or advanced data structures

Syntactic categories:

Category	Domain	Meta variable
Numbers	$\mathbb{Z} = \{0, 1, -1, \dots\}$	$z$
Truth values	$\mathbb{B} = \{\text{true}, \text{false}\}$	$t$
Variables	$Var = \{x, y, \dots\}$	$x$
Arithmetic expressions	$AExp$	$a$
Boolean expressions	$BExp$	$b$
Commands (statements)	$Cmd$	$c$

## Definition (Syntax of WHILE)

The **syntax of WHILE Programs** is defined by the following context-free grammar:

$$a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp$$
$$b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp$$
$$c ::= \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \in Cmd$$

**Remarks:** we assume that

- the syntax of numbers, truth values and variables is given (i.e., no “lexical analysis”)
- the syntax of ambiguous constructs is uniquely determined (by brackets, priorities, or indentation)

- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions
- 6 Execution of Statements

# Operational Semantics of WHILE

- Idea: define meaning of programs by specifying its behaviour being executed on an (abstract) machine
- Here: evaluation/execution relation for program fragments (expressions, statements)
- Approach based on Structural Operational Semantics (SOS)

*G.D. Plotkin: A structural approach to operational semantics, DAIMI FN-19, Computer Science Department, Aarhus University, 1981*

- Employs derivation rules of the form

$$\text{Name} \frac{\text{Premise(s)}}{\text{Conclusion}}$$

- meaning: if every premise is fulfilled, then conclusion can be drawn
  - a rule with no premises is called an axiom
- Derivation rules can be composed to form derivation trees with axioms as leafs (formal definition later)

- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions**
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions
- 6 Execution of Statements

- **Meaning of expression** = value (in the usual sense)
- Depends on the **values of the variables** in the expression

## Definition 2.1 (Program state)

A **(program) state** is an element of the set

$$\Sigma := \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\},$$

called the **state space**.

Thus  $\sigma(x)$  denotes the value of  $x \in Var$  in state  $\sigma \in \Sigma$ .



# Evaluation of Arithmetic Expressions I

**Remember:**  $a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp$

## Definition 2.2 (Evaluation relation for arithmetic expressions)

If  $a \in AExp$  and  $\sigma \in \Sigma$ , then  $\langle a, \sigma \rangle$  is called a **configuration**.

Expression  $a$  **evaluates to**  $z \in \mathbb{Z}$  in state  $\sigma$  (notation:  $\langle a, \sigma \rangle \rightarrow z$ ) if this relationship is derivable by means of the following rules:

Axioms:	$\frac{}{\langle z, \sigma \rangle \rightarrow z}$	$\frac{}{\langle x, \sigma \rangle \rightarrow \sigma(x)}$	
Rules:	$\frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 + a_2, \sigma \rangle \rightarrow z}$	where $z := z_1 + z_2$	
	$\frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 - a_2, \sigma \rangle \rightarrow z}$	where $z := z_1 - z_2$	
	$\frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 * a_2, \sigma \rangle \rightarrow z}$	where $z := z_1 * z_2$	

## Example 2.3

$a = (x+3)*(y-2)$ ,  $\sigma(x) = 3$ ,  $\sigma(y) = 9$ :

$$\frac{\frac{\overline{\langle x, \sigma \rangle \rightarrow 3} \quad \overline{\langle 3, \sigma \rangle \rightarrow 3}}{\overline{\langle x+3, \sigma \rangle \rightarrow 6}} \quad \frac{\overline{\langle y, \sigma \rangle \rightarrow 9} \quad \overline{\langle 2, \sigma \rangle \rightarrow 2}}{\overline{\langle y-2, \sigma \rangle \rightarrow 7}}}{\overline{\langle (x+3)*(y-2), \sigma \rangle \rightarrow 42}}$$

$$\frac{\overline{\langle a_1, \sigma \rangle \rightarrow z_1} \quad \overline{\langle a_2, \sigma \rangle \rightarrow z_2}}{\overline{\langle a_1 * a_2, \sigma \rangle \rightarrow z}} \quad \text{where } z := z_1 * z_2 \quad \frac{\overline{\langle a_1, \sigma \rangle \rightarrow z_1} \quad \overline{\langle a_2, \sigma \rangle \rightarrow z_2}}{\overline{\langle a_1 + a_2, \sigma \rangle \rightarrow z}}$$

**Here:** structure of derivation tree = structure of program fragment  
(generally not the case)

First formal result: value of an expression only depends on valuation of variables which occur in the expression

## Definition 2.4 (Free variables)

The set of **free variables** of an expression is given by the function

$$FV : AExp \rightarrow 2^{Var}$$

where

$$\begin{array}{ll} FV(z) := \emptyset & FV(a_1 + a_2) := FV(a_1) \cup FV(a_2) \\ FV(x) := \{x\} & FV(a_1 - a_2) := FV(a_1) \cup FV(a_2) \\ & FV(a_1 * a_2) := FV(a_1) \cup FV(a_2) \end{array}$$

Result will be shown by **structural induction** on the expression

- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions
- 6 Execution of Statements

## Proof principle

**Given:** an inductive set, i.e., a set  $S$

- which contains certain atomic elements and
- which is closed under certain operations

**To show:** property  $P(s)$  applies to every  $s \in S$

**Proof:** we verify:

**Induction base:**  $P(s)$  holds for every atomic element  $s$

**Induction hypothesis:** assume that  $P(s_1), P(s_2)$  etc.

**Induction step:** then also  $P(f(s_1, \dots, s_n))$  holds for every operation  $f$  of arity  $n$

## Application: natural numbers (“complete induction”)

Definition:  $\mathbb{N}$  is the least set which

- contains 0 and
- contains  $n + 1$  whenever  $n \in \mathbb{N}$

Induction base:  $P(0)$  holds

Induction hypothesis:  $P(n)$  holds

Induction step:  $P(n + 1)$  holds

## Example 2.5

We prove that  $P(n) : \sum_{i=1}^n i = \frac{n(n+1)}{2}$  holds for every  $n \in \mathbb{N}$ .

$P(0)$  holds:  $\sum_{i=1}^0 i = 0 = \frac{0(0+1)}{2} \quad \checkmark$

Assume  $P(n)$ :  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

Show  $P(n+1)$ :  $\sum_{i=1}^{n+1} i = \sum_{i=1}^n i + (n+1)$

$$\begin{aligned} &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} \\ &= \frac{(n+2)(n+1)}{2} \\ &= \frac{(n+1)((n+1)+1)}{2} \quad \checkmark \end{aligned}$$

## Application: arithmetic expressions (Def. 1.2)

**Definition:**  $AExp$  is the least set which

- contains all integers  $z \in \mathbb{Z}$  and all variables  $x \in Var$  and
- contains  $a_1 + a_2$ ,  $a_1 - a_2$  and  $a_1 * a_2$  whenever  $a_1, a_2 \in AExp$

**Induction base:**  $P(z)$  and  $P(x)$  holds (for every  $z \in \mathbb{Z}$  and  $x \in Var$ )

**Induction hypothesis:**  $P(a_1)$  and  $P(a_2)$  holds

**Induction step:**  $P(a_1 + a_2)$ ,  $P(a_1 - a_2)$  and  $P(a_1 * a_2)$  holds



## Lemma 2.6

*Let  $a \in AExp$  and  $\sigma, \sigma' \in \Sigma$  such that  $\sigma(x) = \sigma'(x)$  for every  $x \in FV(a)$ . Then, for every  $z \in \mathbb{Z}$ ,*

$$\langle a, \sigma \rangle \rightarrow z \iff \langle a, \sigma' \rangle \rightarrow z.$$

## Proof.

by **structural induction** on  $a$  (on the board)



- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions**
- 6 Execution of Statements

# Evaluation of Boolean Expressions I

**Remember:**  $b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp$

## Definition 2.7 (Evaluation relation for Boolean expressions)

For  $b \in BExp$  and  $\sigma \in \Sigma$ , and  $t \in \mathbb{B}$ , the **evaluation relation**  $\langle b, \sigma \rangle \rightarrow t$  is defined by the following rules:

$$\begin{array}{c} \overline{\langle t, \sigma \rangle \rightarrow t} \\[10pt] \frac{\langle a_1, \sigma \rangle \rightarrow z \quad \langle a_2, \sigma \rangle \rightarrow z}{\langle a_1 = a_2, \sigma \rangle \rightarrow \text{true}} \qquad \frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 = a_2, \sigma \rangle \rightarrow \text{false}} \text{ if } z_1 \neq z_2 \\[10pt] \frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 > a_2, \sigma \rangle \rightarrow \text{true}} \text{ if } z_1 > z_2 \qquad \frac{\langle a_1, \sigma \rangle \rightarrow z_1 \quad \langle a_2, \sigma \rangle \rightarrow z_2}{\langle a_1 > a_2, \sigma \rangle \rightarrow \text{false}} \text{ if } z_1 \leq z_2 \\[10pt] \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \neg b, \sigma \rangle \rightarrow \text{true}} \qquad \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \neg b, \sigma \rangle \rightarrow \text{false}} \\[10pt] \frac{\langle b_1, \sigma \rangle \rightarrow \text{true} \quad \langle b_2, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{true}} \qquad \frac{\langle b_1, \sigma \rangle \rightarrow \text{true} \quad \langle b_2, \sigma \rangle \rightarrow \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{false}} \\[10pt] \frac{\langle b_1, \sigma \rangle \rightarrow \text{false} \quad \langle b_2, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{false}} \qquad \frac{\langle b_1, \sigma \rangle \rightarrow \text{false} \quad \langle b_2, \sigma \rangle \rightarrow \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{false}} \end{array}$$

( $\vee$  analogously)

## Remarks:

- Binary Boolean operators  $\wedge$  and  $\vee$  are interpreted as **strict**, i.e., always evaluate both arguments.

Important in situations like

```
while p <> nil and p^.key < val do ...!
```

(see 1st exercise sheet for non-strict evaluation)

- $FV : BExp \rightarrow 2^{Var}$  can be defined in analogy to Def. 2.4.
- Lemma 2.6 holds analogously for Boolean expressions, i.e., the value of  $b \in BExp$  does not depend on variables in  $Var \setminus FV(b)$ .

- 1 Repetition: Syntax of WHILE
- 2 Operational Semantics of WHILE
- 3 Evaluation of Arithmetic Expressions
- 4 Excursus: Proof by Structural Induction
- 5 Evaluation of Boolean Expressions
- 6 Execution of Statements

Effect of statement = transformation of program state

**Example:**

$$\langle x := 2+3, \sigma \rangle \rightarrow \sigma[x \mapsto 5]$$

where for every  $\sigma \in \Sigma$ ,  $x, y \in Var$ , and  $z \in \mathbb{Z}$ :

$$\sigma[x \mapsto z](y) := \begin{cases} z & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

# Execution of Statements

## Remember:

$c ::= \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \in \text{Cmd}$

## Definition 2.8 (Execution relation for statements)

For  $c \in \text{Cmd}$  and  $\sigma, \sigma' \in \Sigma$ , the **execution relation**  $\langle c, \sigma \rangle \rightarrow \sigma'$  is defined by the following rules:

$$\begin{array}{lcl} \text{(skip)} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} & \text{(asn)} \frac{\langle a, \sigma \rangle \rightarrow z}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \mapsto z]} \\ \text{(seq)} \frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''} & \text{(if-t)} \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'} \\ \text{(if-f)} \frac{\langle b, \sigma \rangle \rightarrow \text{false} \quad \langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'} & \text{(wh-f)} \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma} \\ \text{(wh-t)} \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \end{array}$$