

# **Semantics and Verification of Software**

Summer Semester 2015

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

Thomas Noll Software Modeling and Verification Group RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ss-15/sv-sw/





# Schedule

- Lecture Tue 14:15–15:45 AH 2 (starting 14 April)
- Lecture Thu 11:45–13:15 AH 2 (starting 9 April)

Semantics and Verification of Software

Lecture 2: Operational Semantics of WHILE I

Summer Semester 2015

(Evaluation of Expressions)

• Exercise class Wed 15:00–16:30 AH 6 (starting 22 April)



Software Modeling

#### **Outline of Lecture 2**

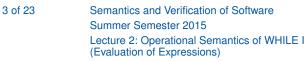
Recap: Syntax of WHILE

**Operational Semantics of WHILE** 

**Evaluation of Arithmetic Expressions** 

**Excursus: Proof by Structural Induction** 

**Evaluation of Boolean Expressions** 







#### **Syntactic Categories**

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

# Syntactic categories:

Category	Domain	Meta variable
Numbers	$\mathbb{Z} = \{0, 1, -1, \ldots\}$	Ζ
Truth values	$\mathbb{B} = \{$ true, false $\}$	t
Variables	$Var = \{x, y, \ldots\}$	X
Arithmetic expressions	AExp (next slide)	а
Boolean expressions	<i>BExp</i> (next slide)	b
Commands (statements)	Cmd (next slide)	С





# Syntax of WHILE Programs

# Definition (Syntax of WHILE)

The syntax of WHILE Programs is defined by the following context-free grammar:  $a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp$   $b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \land b_2 \mid b_1 \lor b_2 \in BExp$  $c ::= skip \mid x := a \mid c_1; c_2 \mid if b then c_1 else c_2 end \mid while b do c end \in Cmd$ 

#### Remarks: we assume that

- the syntax of numbers, truth values and variables is predefined (i.e., no "lexical analysis")
- the syntactic interpretation of ambiguous constructs (expressions) is uniquely determined (by brackets or priorities)





#### **Outline of Lecture 2**

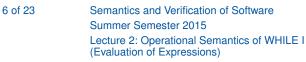
Recap: Syntax of WHILE

**Operational Semantics of WHILE** 

**Evaluation of Arithmetic Expressions** 

**Excursus: Proof by Structural Induction** 

**Evaluation of Boolean Expressions** 







 Idea: define meaning of programs by specifying its behavior being executed on an (abstract) machine





- Idea: define meaning of programs by specifying its behavior being executed on an (abstract) machine
- Here: evaluation/execution relation for program fragments (expressions, statements)





- Idea: define meaning of programs by specifying its behavior 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



- Idea: define meaning of programs by specifying its behavior 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

(Name) Premise(s) [side conditions]

- meaning: if every premise [and all side conditions] are fulfilled, then conclusion can be drawn
- a rule with no premises is called an axiom





- Idea: define meaning of programs by specifying its behavior 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

(Name)  $\frac{Premise(s)}{Conclusion}$  [side conditions]

- meaning: if every premise [and all side conditions] are 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)





#### **Outline of Lecture 2**

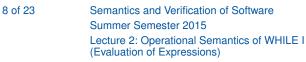
Recap: Syntax of WHILE

**Operational Semantics of WHILE** 

**Evaluation of Arithmetic Expressions** 

**Excursus: Proof by Structural Induction** 

**Evaluation of Boolean Expressions** 







# **Program States**

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





# **Program States**

- Meaning of expression = its 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 \to \mathbb{Z} \},\$$

called the state space.

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





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





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

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:  $\langle a, \sigma \rangle \to z$ ) if this relationship is derivable by means of the following rules:

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

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

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

10 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)





#### **Evaluation of Arithmetic Expressions II**

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





#### **Evaluation of Arithmetic Expressions II**

#### Example 2.3

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

 $\langle (x+3)*(y-2), \sigma \rangle \rightarrow$ 





# **Evaluation of Arithmetic Expressions II**

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

$$\begin{array}{c|c} \langle \mathbf{x} + \mathbf{3}, \sigma \rangle \to & \langle \mathbf{y} - \mathbf{2}, \sigma \rangle \to \\ & \langle (\mathbf{x} + \mathbf{3}) \ast (\mathbf{y} - \mathbf{2}), \sigma \rangle \to \\ \hline & \frac{\langle a_1, \sigma \rangle \to z_1 \ \langle a_2, \sigma \rangle \to z_2}{\langle a_1 \ast a_2, \sigma \rangle \to z} \quad \text{where } z := z_1 \cdot z_2 \end{array}$$





### **Evaluation of Arithmetic Expressions II**

### Example 2.3

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

$$\frac{\overline{\langle x, \sigma \rangle \rightarrow} \quad \overline{\langle 3, \sigma \rangle \rightarrow}}{\langle x+3, \sigma \rangle \rightarrow} \quad \overline{\langle y-2, \sigma \rangle \rightarrow}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \rightarrow}$$

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

11 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)





#### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to}}{\langle x+3, \sigma \rangle \to} \quad \overline{\langle y-2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\overline{\langle x, \sigma \rangle \to \sigma(x)}$$







#### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to} \overline{\langle 3, \sigma \rangle \to 3}$$

$$\overline{\langle y-2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\overline{\langle z, \sigma \rangle \to z}$$





### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y-2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

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





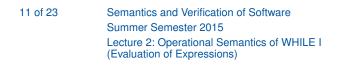
### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y, \sigma \rangle \to} \quad \overline{\langle 2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\frac{\langle a_1, \sigma \rangle \to z_1 \quad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 - a_2, \sigma \rangle \to z} \quad \text{where } z := z_1 - z_2$$







#### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \frac{\overline{\langle y, \sigma \rangle \to 9} \quad \overline{\langle 2, \sigma \rangle \to}}{\langle y-2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\overline{\langle x, \sigma \rangle \to \sigma(x)}$$





#### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \frac{\overline{\langle y, \sigma \rangle \to 9} \quad \overline{\langle 2, \sigma \rangle \to 2}}{\langle y-2, \sigma \rangle \to}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\overline{\langle z, \sigma \rangle \to z}$$





### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y, \sigma \rangle \to 9} \quad \overline{\langle 2, \sigma \rangle \to 2}$$

$$\overline{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y-2, \sigma \rangle \to 7}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to}$$

$$\frac{\langle a_1, \sigma \rangle \to z_1 \quad \langle a_2, \sigma \rangle \to z_2}{\langle a_1-a_2, \sigma \rangle \to z} \quad \text{where } z := z_1 - z_2$$





### **Evaluation of Arithmetic Expressions II**

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y, \sigma \rangle \to 9} \quad \overline{\langle 2, \sigma \rangle \to 2}$$

$$\overline{\langle y-2, \sigma \rangle \to 7}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to 42}$$

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





#### **Evaluation of Arithmetic Expressions II**

#### Example 2.3

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

$$\frac{\overline{\langle x, \sigma \rangle \to 3} \quad \overline{\langle 3, \sigma \rangle \to 3}}{\langle x+3, \sigma \rangle \to 6} \quad \overline{\langle y, \sigma \rangle \to 9} \quad \overline{\langle 2, \sigma \rangle \to 2}$$

$$\overline{\langle (x+3)*(y-2), \sigma \rangle \to 42}$$

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

11 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)





#### **Free Variables I**

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





#### **Free Variables I**

First formal result: value of an expression only depends on valuation of variables which occur (freely) 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

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





#### **Free Variables I**

First formal result: value of an expression only depends on valuation of variables which occur (freely) 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

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

Result will be shown by structural induction on the expression

12 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)





#### **Outline of Lecture 2**

Recap: Syntax of WHILE

**Operational Semantics of WHILE** 

**Evaluation of Arithmetic Expressions** 

**Excursus: Proof by Structural Induction** 

**Evaluation of Boolean Expressions** 

13 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)





#### **Excursus: Proof by Structural Induction**

#### **Excursus: Proof by Structural Induction I**

# Proof principle

Given: an inductive set, i.e., a set S whose elements are either

- atomic or
- obtained from atomic elements by (finite) application of 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, \ldots, s_n)) holds for every operation f of arity n
```





### **Excursus: Proof by Structural Induction**

#### **Excursus: Proof by Structural Induction I**

# Proof principle

```
Given: an inductive set, i.e., a set S whose elements are either
```

- atomic or
- obtained from atomic elements by (finite) application of 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, ..., s_n)) holds for every operation f of arity n
```

Remark: structural induction is a special case of well-founded induction





#### **Excursus: Proof by Structural Induction**

# **Excursus: Proof by Structural Induction II**

Application: natural numbers ("mathematical induction")

Definition:  $\ensuremath{\mathbb{N}}$  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



#### 15 of 23 Semantics and Verification of Software Summer Semester 2015 Lecture 2: Operational Semantics of WHILE I (Evaluation of Expressions)

# **Excursus: Proof by Structural Induction II**

Application: natural numbers ("mathematical 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

# Generalization: complete (strong, course-of-values) induction

- induction step:  $P(0), P(1), \ldots, P(n) \Rightarrow P(n+1)$
- corresponds to well-founded induction over natural numbers





# **Excursus: Proof by Structural Induction III**

Example 2.5 (Mathematical induction)

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







# **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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





# **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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





## **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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:  $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2} \checkmark$ Assume P(n):  $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Show P(n+1):  $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$ 





## **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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:  $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2} \checkmark$ Assume P(n):  $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Show P(n+1):  $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$  $= \frac{n(n+1)}{2} + (n+1)$ 







### **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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:  $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2} \checkmark$ Assume P(n):  $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Show P(n+1):  $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$   $= \frac{n(n+1)}{2} + (n+1)$  $= \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$ 







### **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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:  $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2} \checkmark$ Assume P(n):  $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Show P(n+1):  $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$   $= \frac{n(n+1)}{2} + (n+1)$   $= \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$  $= \frac{(n+2)(n+1)}{2}$ 





### **Excursus: Proof by Structural Induction III**

#### Example 2.5 (Mathematical induction)

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:  $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2} \checkmark$ Assume  $P(n) : \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Show  $P(n+1) : \sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$   $= \frac{n(n+1)}{2} + (n+1)$   $= \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$  $= \frac{(n+2)(n+1)}{2} \checkmark$ 





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





#### **Free Variables II**

#### 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  $z \in \mathbb{Z}$ ,

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





#### **Free Variables II**

#### 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  $z \in \mathbb{Z}$ ,

$$\langle \boldsymbol{a}, \sigma \rangle \to \boldsymbol{z} \iff \langle \boldsymbol{a}, \sigma' \rangle \to \boldsymbol{z}.$$

#### Proof.

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





# **Outline of Lecture 2**

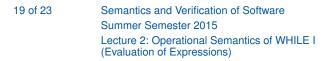
Recap: Syntax of WHILE

**Operational Semantics of WHILE** 

**Evaluation of Arithmetic Expressions** 

**Excursus: Proof by Structural Induction** 

**Evaluation of Boolean Expressions** 



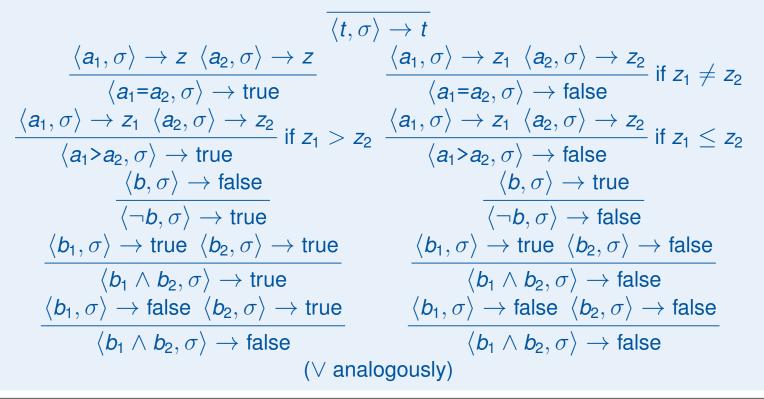




# **Evaluation of Boolean Expressions I**

Definition 2.7 ((Strict) evaluation relation for Boolean expressions)

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







## **Evaluation of Boolean Expressions**

# **Evaluation of Boolean Expressions II**

### **Remarks:**

Binary Boolean operators 
 A and 
 V are interpreted as strict, i.e., always evaluate both arguments.

Important in situations like

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

(see following slides for alternatives)



## **Evaluation of Boolean Expressions**

# **Evaluation of Boolean Expressions II**

## **Remarks:**

Binary Boolean operators 
 A and 
 V are interpreted as strict, i.e., always evaluate both arguments.

Important in situations like

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

(see following slides for alternatives)

•  $FV : BExp \rightarrow 2^{Var}$  can be defined in analogy to Def. 2.4.





# **Evaluation of Boolean Expressions**

# **Evaluation of Boolean Expressions II**

## **Remarks:**

Binary Boolean operators 
 A and 
 V are interpreted as strict, i.e., always evaluate both arguments.

Important in situations like

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

(see following slides for alternatives)

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





# **Evaluation of Boolean Expressions III**

### Definition 2.8 (Sequential evaluation of Boolean expressions)

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

$$\frac{\langle b_1, \sigma \rangle \to \text{false}}{\langle b_1 \land b_2, \sigma \rangle \to \text{false}} \quad \frac{\langle b_1, \sigma \rangle \to \text{true } \langle b_2, \sigma \rangle \to t}{\langle b_1 \land b_2, \sigma \rangle \to t}$$
$$\frac{\langle b_1, \sigma \rangle \to \text{true }}{\langle b_1 \lor b_2, \sigma \rangle \to \text{true }} \quad \frac{\langle b_1, \sigma \rangle \to \text{false } \langle b_2, \sigma \rangle \to t}{\langle b_1 \lor b_2, \sigma \rangle \to t}$$





# **Evaluation of Boolean Expressions III**

### Definition 2.8 (Sequential evaluation of Boolean expressions)

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

$$\frac{\langle b_1, \sigma \rangle \to \text{false}}{\langle b_1 \land b_2, \sigma \rangle \to \text{false}} \quad \frac{\langle b_1, \sigma \rangle \to \text{true } \langle b_2, \sigma \rangle \to t}{\langle b_1 \land b_2, \sigma \rangle \to t} \\
\frac{\langle b_1, \sigma \rangle \to \text{true }}{\langle b_1 \lor b_2, \sigma \rangle \to \text{true }} \quad \frac{\langle b_1, \sigma \rangle \to \text{false } \langle b_2, \sigma \rangle \to t}{\langle b_1 \lor b_2, \sigma \rangle \to t}$$

**Remarks:** yields same result as strict evaluation

- (Boolean) expressions have no side effects (assignments, exceptions, ...)
- evaluation always terminates





# **Evaluation of Boolean Expressions IV**

### Definition 2.9 (Parallel evaluation of Boolean expressions)

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

$$\begin{array}{c} \frac{\langle b_1, \sigma \rangle \rightarrow \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{false}} & \frac{\langle b_2, \sigma \rangle \rightarrow \text{false}}{\langle b_1 \wedge b_2, \sigma \rangle \rightarrow \text{false}} \\ \frac{\langle b_1, \sigma \rangle \rightarrow \text{true}}{\langle b_1, \sigma \rangle \rightarrow \text{true}} & \langle b_2, \sigma \rangle \rightarrow \text{true}} \\ \frac{\langle b_1, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow \text{true}} & \frac{\langle b_2, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow \text{true}} \\ \frac{\langle b_1, \sigma \rangle \rightarrow \text{true}}{\langle b_1, \sigma \rangle \rightarrow \text{true}} & \frac{\langle b_2, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow \text{true}} \\ \frac{\langle b_1, \sigma \rangle \rightarrow \text{false}}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow \text{false}} & \frac{\langle b_2, \sigma \rangle \rightarrow \text{true}}{\langle b_1 \vee b_2, \sigma \rangle \rightarrow \text{true}} \end{array}$$



