



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

Recap: Syntax of WHILE

Syntactic Categories

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

Syntactic categories:

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

Recap: Syntax of WHILE

Syntax of WHILE Programs

Definition (Syntax of WHILE)

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

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

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)

Operational Semantics of WHILE

Operational Semantics of WHILE

- 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

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

- 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)

Evaluation of Arithmetic Expressions

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

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

called the **state space**.

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

Evaluation of Arithmetic Expressions

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

Axioms:

$$\frac{}{\langle z, \sigma \rangle \rightarrow z} \quad \frac{}{\langle x, \sigma \rangle \rightarrow \sigma(x)}$$

Rules:

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

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

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

Evaluation of Arithmetic Expressions

Evaluation of Arithmetic Expressions II

Example 2.3

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

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

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

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

Evaluation of Arithmetic Expressions

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

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

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

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 $P(f(s_1, \dots, s_n))$ 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: \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: $P(0), P(1), \dots, P(n) \Rightarrow P(n + 1)$
- corresponds to well-founded induction over natural numbers

Excursus: Proof by Structural Induction

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

Excursus: Proof by Structural Induction

Excursus: Proof by Structural Induction IV

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

Excursus: Proof by Structural Induction

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 $z \in \mathbb{Z}$,

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

Proof.

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



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** $\langle b, \sigma \rangle \rightarrow t$ is defined by:

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

(\vee analogously)

Evaluation of Boolean Expressions

Evaluation of Boolean Expressions II

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 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 $b \in BExp$ does not depend on variables in $Var \setminus FV(b)$.

Evaluation of Boolean Expressions

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** $\langle b, \sigma \rangle \rightarrow t$ is defined by the following rules:

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

Remarks: yields same result as strict evaluation

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

Evaluation of Boolean Expressions

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:

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