Semantics and Verification of Software

Lecture 2: Operational Semantics of WHILE

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw/

Summer semester 2007

Outline

Syntactic Categories

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

Syntactic categories:

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

Syntax of WHILE Programs

Definition (Syntax of WHILE)

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

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

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)

Outline

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

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

  - 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)

Outline

Program States

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

Definition 2.2 (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 $\sigma \in \Sigma$.

Evaluation of Arithmetic Expressions I

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

Definition 2.3 (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:

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

Rules:

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

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

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

Evaluation of Arithmetic Expressions II

Example 2.4

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

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

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

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

Free Variables

First formal result: value of an expression does not depend on the valuation of variables which do not occur in the expression

Definition 2.5 (Free variables)

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

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

where

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

Result will be shown by structural induction on the expression

Outline

Excursus: Proof by Structural Induction I

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 $P(f(s_1, \ldots, s_n))$ holds for every operation f of arity n

Excursus: Proof by Structural Induction II

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

Excursus: Proof by Structural Induction III

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 $a_1\texttt{+}a_2$, $a_1\texttt{-}a_2$ and $a_1\texttt{*}a_2$ 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: $P(a_1\texttt{+}a_2)$, $P(a_1\texttt{-}a_2)$ and $P(a_1\texttt{*}a_2)$ 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* $z \in \mathbb{Z}$,

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

Proof.

by structural induction on a (on the board) □

Outline

Evaluation of Boolean Expressions I

Remember: $b ::= t \mid a_1\texttt{=}a_2 \mid a_1\texttt{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in \mathit{BExp}$

Definition 2.7 (Evaluation relation for Boolean expressions)

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

$$\frac{}{\langle t, \sigma\rangle \to t}$$

$$\frac{\langle a_1, \sigma\rangle \to z \quad \langle a_2, \sigma\rangle \to z}{\langle a_1\texttt{=}a_2, \sigma\rangle \to \mathsf{true}} \qquad \frac{\langle a_1, \sigma\rangle \to z_1 \quad \langle a_2, \sigma\rangle \to z_2}{\langle a_1\texttt{=}a_2, \sigma\rangle \to \mathsf{false}} \text{ if } z_1 \neq z_2$$

$$\frac{\langle a_1, \sigma\rangle \to z_1 \quad \langle a_2, \sigma\rangle \to z_2}{\langle a_1\texttt{>}a_2, \sigma\rangle \to \mathsf{true}} \text{ if } z_1 > z_2 \qquad \frac{\langle a_1, \sigma\rangle \to z_1 \quad \langle a_2, \sigma\rangle \to z_2}{\langle a_1\texttt{>}a_2, \sigma\rangle \to \mathsf{false}} \text{ if } z_1 \leq z_2$$

$$\frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle \neg b, \sigma\rangle \to \mathsf{true}} \qquad \frac{\langle b, \sigma\rangle \to \mathsf{true}}{\langle \neg b, \sigma\rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma\rangle \to \mathsf{true} \quad \langle b_2, \sigma\rangle \to \mathsf{true}}{\langle b_1 \wedge b_2, \sigma\rangle \to \mathsf{true}} \qquad \frac{\langle b_1, \sigma\rangle \to \mathsf{true} \quad \langle b_2, \sigma\rangle \to \mathsf{false}}{\langle b_1 \wedge b_2, \sigma\rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma\rangle \to \mathsf{false} \quad \langle b_2, \sigma\rangle \to \mathsf{true}}{\langle b_1 \wedge b_2, \sigma\rangle \to \mathsf{false}} \qquad \frac{\langle b_1, \sigma\rangle \to \mathsf{false} \quad \langle b_2, \sigma\rangle \to \mathsf{false}}{\langle b_1 \wedge b_2, \sigma\rangle \to \mathsf{false}}$$

($\vee$ analogously)

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 Assignment 1 for non–strict evaluation)
- $FV : BExp \to 2^{Var}$ can be defined in analogy to Def. 2.5.
- Lemma 2.6 holds analogously for Boolean expressions, i.e., the value of $b \in BExp$ does not depend on variables in $Var \setminus FV(b)$.

Outline

Effect of statement = transformation of program state

Example:

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

where for every $\sigma \in \Sigma$, $x, y \in \mathit{Var}$, 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:

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

Definition 2.8 (Execution relation for statements)

For $c \in Cmd$ and $\sigma, \sigma' \in \Sigma$, the execution relation $\langle c, \sigma \rangle \to \sigma'$ is defined by the following rules:

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \to \sigma} \text{(skip)} \qquad \frac{\langle a, \sigma \rangle \to z}{\langle x \texttt{ := } a, \sigma \rangle \to \sigma[x \mapsto z]} \text{(asgn)}$$

$$\frac{\langle c_1, \sigma \rangle \to \sigma' \quad \langle c_2, \sigma' \rangle \to \sigma''}{\langle c_1\texttt{;}c_2, \sigma \rangle \to \sigma''} \text{(seq)} \qquad \frac{\langle b, \sigma \rangle \to \mathsf{true} \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'} \text{(if–t)}$$

$$\frac{\langle b, \sigma \rangle \to \mathsf{false} \quad \langle c_2, \sigma \rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'} \text{(if–f)} \qquad \frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \sigma} \text{(wh–f)}$$

$$\frac{\langle b, \sigma \rangle \to \mathsf{true} \quad \langle c, \sigma \rangle \to \sigma' \quad \langle \texttt{while } b \texttt{ do } c, \sigma' \rangle \to \sigma''}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \sigma''} \text{(wh–t)}$$

Non–Terminating Statements

Corollary 2.9

The execution relation for statements is not total, i.e., there exist $c \in Cmd$ and $\sigma \in \Sigma$ such that $\langle c, \sigma \rangle \to \sigma'$ for no $\sigma' \in \Sigma$.

Proof.

Counterexample: $c =$ `while true do skip`
(by contradiction; on the board) □