



Semantics and Verification of Software

Summer Semester 2015

Lecture 5: Operational Semantics of WHILE IV
(The Compiler & Its Correctness)

Thomas Noll

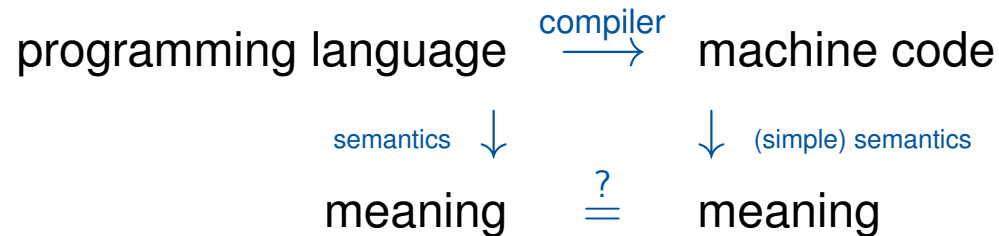
Software Modeling and Verification Group

RWTH Aachen University

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

Recap: Abstract Machine

Compiler Correctness



To do:

1. Definition of **abstract machine**
2. Definition of (operational) **semantics of machine instructions**
3. Definition of **translation** WHILE \rightarrow machine code (“compiler”)
4. **Proof:** semantics of generated machine code = semantics of original source code

Recap: Abstract Machine

The Abstract Machine

Definition (Abstract machine)

The **abstract machine (AM)** is given by

- **programs** $P \in \text{Code}$ and **instructions** p :

$$P ::= p^*$$

$$p ::= \text{PUSH}(z) \mid \text{PUSH}(t) \mid \text{ADD} \mid \text{SUB} \mid \text{MULT} \mid \\ \text{EQ} \mid \text{GT} \mid \text{NOT} \mid \text{AND} \mid \text{OR} \mid \\ \text{LOAD}(x) \mid \text{STO}(x) \mid \text{JMP}(k) \mid \text{JMPF}(k)$$

(where $z, k \in \mathbb{Z}$, $t \in \mathbb{B}$, and $x \in \text{Var}$)

- **configurations** of the form $\langle pc, e, \sigma \rangle \in \text{Cnf}$ where

– $pc \in \mathbb{Z}$ is the **program counter** (i.e., address of next instruction to be executed)

– $e \in \text{Stk} := (\mathbb{Z} \cup \mathbb{B})^*$ is the **evaluation stack** (top right)

– $\sigma \in \Sigma := (\text{Var} \rightarrow \mathbb{Z})$ is the **(storage) state**

(thus $\text{Cnf} = \mathbb{Z} \times \text{Stk} \times \Sigma$)

- **initial configurations** of the form $\langle 0, \varepsilon, \sigma \rangle$
- **final configurations** of the form $\langle |P|, e, \sigma \rangle$

Recap: Abstract Machine

Extension of Code and Stack

Lemma

If $P \vdash \langle pc, e, \sigma \rangle \triangleright^* \langle pc', e', \sigma' \rangle$, then

$$P_1; P; P_2 \vdash \langle |P_1| + pc, e_0 : e, \sigma \rangle \triangleright^* \langle |P_1| + pc', e_0 : e', \sigma' \rangle$$

for all $P_1, P_2 \in \text{Code}$ and $e_0 \in \text{Stk}$.

Interpretation: both the code and the stack component can be extended without actually changing the behaviour of the machine

Proof.

by induction on the length of the computation (on the board)



Recap: Abstract Machine

Another Property: Determinism

Lemma

The semantics of AM is **deterministic**: for all $\gamma, \gamma', \gamma'' \in \text{Cnf}$,

$$\gamma \triangleright \gamma' \text{ and } \gamma \triangleright \gamma'' \text{ imply } \gamma' = \gamma''.$$

Proof (Idea).

- Instruction to be executed is unambiguously given by program counter
- Topmost stack entries and storage state then yield unique successor configuration □

Thus the following function is well defined:

Definition (Semantics of AM)

The **semantics of an AM program** is given by $\mathfrak{M}[\cdot] : \text{Code} \rightarrow (\Sigma \dashrightarrow \Sigma)$ as follows:

$$\mathfrak{M}[P]\sigma := \begin{cases} \sigma' & \text{if } P \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |P|, e, \sigma' \rangle \text{ for some } e \in \text{Stk} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recap: Syntax of WHILE Programs

Definition (Syntax of WHILE (Definition 1.2))

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

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

Translation of Arithmetic Expressions

Definition 5.1 (Translation of arithmetic expressions)

The translation function

$$\mathfrak{T}_a[\cdot] : AExp \rightarrow Code$$

is given by

$$\begin{aligned}\mathfrak{T}_a[z] &:= PUSH(z) \\ \mathfrak{T}_a[x] &:= LOAD(x) \\ \mathfrak{T}_a[a_1 + a_2] &:= \mathfrak{T}_a[a_1]; \mathfrak{T}_a[a_2]; ADD \\ \mathfrak{T}_a[a_1 - a_2] &:= \mathfrak{T}_a[a_1]; \mathfrak{T}_a[a_2]; SUB \\ \mathfrak{T}_a[a_1 * a_2] &:= \mathfrak{T}_a[a_1]; \mathfrak{T}_a[a_2]; MULT\end{aligned}$$

Example 5.2

$$\begin{aligned}\mathfrak{T}_a[x + 1] &= \mathfrak{T}_a[x]; \mathfrak{T}_a[1]; ADD \\ &= LOAD(x); PUSH(1); ADD\end{aligned}$$

Translation of Boolean Expressions

Definition 5.3 (Translation of Boolean expressions)

The translation function

$$\mathcal{T}_b[\cdot] : BExp \rightarrow Code$$

is given by

$$\begin{aligned}\mathcal{T}_b[\text{true}] &:= \text{PUSH}(\text{true}) \\ \mathcal{T}_b[\text{false}] &:= \text{PUSH}(\text{false}) \\ \mathcal{T}_b[a_1 = a_2] &:= \mathcal{T}_a[a_1]; \mathcal{T}_a[a_2]; \text{EQ} \\ \mathcal{T}_b[a_1 > a_2] &:= \mathcal{T}_a[a_1]; \mathcal{T}_a[a_2]; \text{GT} \\ \mathcal{T}_b[\neg b] &:= \mathcal{T}_b[b]; \text{NOT} \\ \mathcal{T}_b[b_1 \wedge b_2] &:= \mathcal{T}_b[b_1]; \mathcal{T}_b[b_2]; \text{AND} \\ \mathcal{T}_b[b_1 \vee b_2] &:= \mathcal{T}_b[b_1]; \mathcal{T}_b[b_2]; \text{OR}\end{aligned}$$

Translation of Statements

Definition 5.4 (Translation of statements)

The translation function $\mathcal{T}_c[\cdot] : \text{Cmd} \rightarrow \text{Code}$ is given by

$$\begin{aligned}\mathcal{T}_c[\text{skip}] &:= \varepsilon \\ \mathcal{T}_c[x := a] &:= \mathcal{T}_a[a]; \text{STO}(x) \\ \mathcal{T}_c[c_1; c_2] &:= \mathcal{T}_c[c_1]; \mathcal{T}_c[c_2] \\ \mathcal{T}_c[\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}] &:= \mathcal{T}_b[b]; \text{JMPF}(|\mathcal{T}_c[c_1]| + 2); \\ &\quad \mathcal{T}_c[c_1]; \text{JMP}(|\mathcal{T}_c[c_2]| + 1); \\ &\quad \mathcal{T}_c[c_2] \\ \mathcal{T}_c[\text{while } b \text{ do } c \text{ end}] &:= \mathcal{T}_b[b]; \text{JMPF}(|\mathcal{T}_c[c]| + 2); \\ &\quad \mathcal{T}_c[c]; \text{JMP}(-(|\mathcal{T}_b[b]| + |\mathcal{T}_c[c]| + 1))\end{aligned}$$

Translation of Statements

Example 5.5 (Factorial program)

$$\begin{aligned} & \mathcal{T}_c \llbracket y:=1; \text{while } \neg(x=1) \text{ do } y:=y*x; x:=x-1 \text{ end} \rrbracket \\ &= \mathcal{T}_c \llbracket y:=1 \rrbracket ; \mathcal{T}_c \llbracket \underbrace{\neg(x=1)}_b \text{ do } \underbrace{y:=y*x; x:=x-1}_c \text{ end} \rrbracket \\ &= \text{PUSH}(1); \text{STO}(y); \\ & \quad \mathcal{T}_b \llbracket b \rrbracket ; \text{JMPF}(|\mathcal{T}_c \llbracket c \rrbracket| + 2); \\ & \quad \mathcal{T}_c \llbracket c \rrbracket ; \text{JMP}(-(|\mathcal{T}_b \llbracket b \rrbracket| + |\mathcal{T}_c \llbracket c \rrbracket| + 1)) \\ &= \text{PUSH}(1); \text{STO}(y); \\ & \quad \text{LOAD}(x); \text{PUSH}(1); \text{EQ}; \text{NOT}; \text{JMPF}(8 + 2); \\ & \quad \text{LOAD}(y); \text{LOAD}(x); \text{MULT}; \text{STO}(y); \\ & \quad \text{LOAD}(x); \text{PUSH}(1); \text{SUB}; \text{STO}(x); \text{JMP}(-(4 + 8 + 1)) \\ &= \text{PUSH}(1); \text{STO}(y); \\ & \quad \text{LOAD}(x); \text{PUSH}(1); \text{EQ}; \text{NOT}; \text{JMPF}(10); \\ & \quad \text{LOAD}(y); \text{LOAD}(x); \text{MULT}; \text{STO}(y); \\ & \quad \text{LOAD}(x); \text{PUSH}(1); \text{SUB}; \text{STO}(x); \text{JMP}(-13) \end{aligned}$$

The Compiler

Execution of Factorial Program

Example 5.6 (Factorial program)

Let $P :=$ 0:PUSH(1); 1:STO(y);
2:LOAD(x); 3:PUSH(1); 4:EQ; 5:NOT; 6:JMPF(10);
7:LOAD(y); 8:LOAD(x); 9:MULT; 10:STO(y);
11:LOAD(x); 12:PUSH(1); 13:SUB; 14:STO(x); 15:JMP(-13)

and $\sigma \in \Sigma$ with $\sigma(x) = 2$.

$\langle 0, \varepsilon, \sigma \rangle$	$\triangleright \langle 11, \varepsilon, \sigma[y \mapsto 2] \rangle$
$\triangleright \langle 1, 1, \sigma \rangle$	$\triangleright \langle 12, 2, \sigma[y \mapsto 2] \rangle$
$\triangleright \langle 2, \varepsilon, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 13, 2 : 1, \sigma[y \mapsto 2] \rangle$
$\triangleright \langle 3, 2, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 14, 1, \sigma[y \mapsto 2] \rangle$
$\triangleright \langle 4, 2 : 1, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 15, \varepsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 5, \text{false}, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 2, \varepsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 6, \text{true}, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 3, 1, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 7, \varepsilon, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 4, 1 : 1, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 8, 1, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 5, \text{true}, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 9, 1 : 2, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 6, \text{false}, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\triangleright \langle 10, 2, \sigma[y \mapsto 1] \rangle$	$\triangleright \langle 16, \varepsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$

Proof of Compiler Correctness

Correctness of $\mathcal{T}_a[\cdot]$ I

Definition (Recap: Evaluation of arithmetic expressions (Definition 2.2))

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:

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

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

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

Proof of Compiler Correctness

Correctness of $\mathcal{T}_a[\cdot]$ II

Lemma 5.7 (Correctness of $\mathcal{T}_a[\cdot]$)

For every $a \in AExp$, $\sigma \in \Sigma$ and $z \in \mathbb{Z}$,

$$\langle a, \sigma \rangle \rightarrow z \text{ implies } \mathcal{T}_a[a] \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |\mathcal{T}_a[a]|, z, \sigma \rangle.$$

Proof.

by induction on the syntactic structure of a (on the board) □

Proof of Compiler Correctness

Correctness of $\mathcal{T}_b[\cdot]$ I

Definition (Recap: Semantics of Boolean expressions (Definition 2.7))

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

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

Proof of Compiler Correctness

Correctness of $\mathcal{T}_b[\cdot]$ II

Lemma 5.8 (Correctness of $\mathcal{T}_b[\cdot]$)

For every $b \in BExp$, $\sigma \in \Sigma$ and $t \in \mathbb{B}$,

$$\langle b, \sigma \rangle \rightarrow t \text{ implies } \mathcal{T}_b[b] \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |\mathcal{T}_b[b]|, t, \sigma \rangle.$$

Proof.

by induction on the syntactic structure of b (omitted) □

Proof of Compiler Correctness

Correctness of $\mathcal{T}_c[\cdot]$ I

Definition (Recap: Operational functional (Definition 4.1))

The **functional of the operational semantics**,

$$\mathcal{D}[\cdot] : \text{Cmd} \rightarrow (\Sigma \dashrightarrow \Sigma),$$

assigns to every statement $c \in \text{Cmd}$ a **partial state transformation** $\mathcal{D}[c] : \Sigma \dashrightarrow \Sigma$, which is defined as follows:

$$\mathcal{D}[c]\sigma := \begin{cases} \sigma' & \text{if } \langle c, \sigma \rangle \rightarrow \sigma' \text{ for some } \sigma' \in \Sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition (Recap: Semantics of machine code (Definition 4.11))

The **semantics of an AM program** is given by $\mathfrak{M}[\cdot] : \text{Code} \rightarrow (\Sigma \dashrightarrow \Sigma)$ as follows:

$$\mathfrak{M}[P]\sigma := \begin{cases} \sigma' & \text{if } P \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |P|, e, \sigma' \rangle \text{ for some } e \in \text{Stk} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Proof of Compiler Correctness

Correctness of $\mathcal{T}_c[\cdot]$ II

Theorem 5.9 (Correctness of $\mathcal{T}_c[\cdot]$)

For every $c \in \text{Cmd}$,

$$\mathcal{D}[\![c]\!] = \mathcal{M}[\![\mathcal{T}_c[c]]\!].$$

Proof carried out in two parts. First step: from source to machine code

Lemma 5.10

For every $c \in \text{Cmd}$ and $\sigma, \sigma' \in \Sigma$,

$$\langle c, \sigma \rangle \rightarrow \sigma' \text{ implies } \mathcal{T}_c[\![c]\!] \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle \![\mathcal{T}_c[c]]\!, \varepsilon, \sigma' \rangle.$$

Proof.

by induction on the derivation tree of $\langle c, \sigma \rangle \rightarrow \sigma'$ (on the board) □

Proof of Compiler Correctness

Correctness of $\mathcal{T}_c[\cdot]$ III

Second step: from machine to source code

Lemma 5.11

For every $c \in \text{Cmd}$, $\sigma, \sigma' \in \Sigma$, and $e \in \text{Stk}$,

$\mathcal{T}_c[[c]] \vdash \langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |\mathcal{T}_c[[c]]|, e, \sigma' \rangle$ implies $\langle c, \sigma \rangle \rightarrow \sigma'$ and $e = \varepsilon$.

Proof.

by induction on the length of the computation sequence

$$\langle 0, \varepsilon, \sigma \rangle \triangleright^* \langle |\mathcal{T}_c[[c]]|, e, \sigma' \rangle$$

(see exercises)

