



Semantics and Verification of Software

Summer Semester 2019

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

Thomas Noll

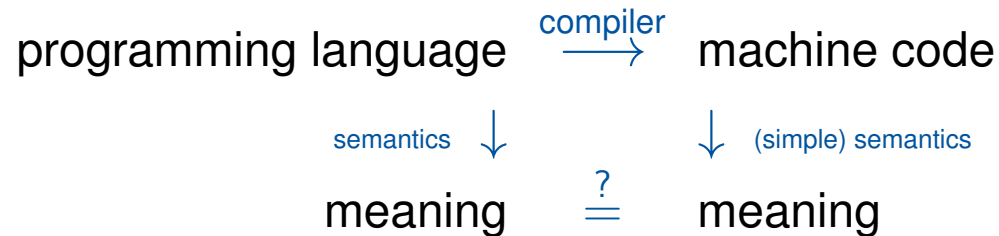
Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-19/sv-sw/>

Recap: Compiler Correctness

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: Compiler Correctness

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 to the 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: Compiler Correctness

Semantics of AM Instructions

Definition (Transition relation of AM)

For $P = p_0; \dots; p_{n-1} \in \text{Code}$ and $0 \leq pc < n$, the **transition relation** $\triangleright \subseteq \text{Cnf} \times \text{Cnf}$ is given by

$P \vdash \langle pc, e, \sigma \rangle \triangleright \langle pc + 1, e : z, \sigma \rangle$	if $p_{pc} = \text{PUSH}(z)$
$P \vdash \langle pc, e, \sigma \rangle \triangleright \langle pc + 1, e : t, \sigma \rangle$	if $p_{pc} = \text{PUSH}(t)$
$P \vdash \langle pc, e : z_1 : z_2, \sigma \rangle \triangleright \langle pc + 1, e : (z_1 + z_2), \sigma \rangle$	if $p_{pc} = \text{ADD}$
$P \vdash \langle pc, e : z_1 : z_2, \sigma \rangle \triangleright \langle pc + 1, e : (z_1 - z_2), \sigma \rangle$	if $p_{pc} = \text{SUB}$
$P \vdash \langle pc, e : z_1 : z_2, \sigma \rangle \triangleright \langle pc + 1, e : (z_1 \cdot z_2), \sigma \rangle$	if $p_{pc} = \text{MULT}$
$P \vdash \langle pc, e : z_1 : z_2, \sigma \rangle \triangleright \langle pc + 1, e : (z_1 = z_2), \sigma \rangle$	if $p_{pc} = \text{EQ}$
$P \vdash \langle pc, e : z_1 : z_2, \sigma \rangle \triangleright \langle pc + 1, e : (z_1 > z_2), \sigma \rangle$	if $p_{pc} = \text{GT}$
$P \vdash \langle pc, e : t, \sigma \rangle \triangleright \langle pc + 1, e : (\neg t), \sigma \rangle$	if $p_{pc} = \text{NOT}$
$P \vdash \langle pc, e : t_1 : t_2, \sigma \rangle \triangleright \langle pc + 1, e : (t_1 \wedge t_2), \sigma \rangle$	if $p_{pc} = \text{AND}$
$P \vdash \langle pc, e : t_1 : t_2, \sigma \rangle \triangleright \langle pc + 1, e : (t_1 \vee t_2), \sigma \rangle$	if $p_{pc} = \text{OR}$
$P \vdash \langle pc, e, \sigma \rangle \triangleright \langle pc + 1, e : \sigma(x), \sigma \rangle$	if $p_{pc} = \text{LOAD}(x)$
$P \vdash \langle pc, e : z, \sigma \rangle \triangleright \langle pc + 1, e, \sigma[x \mapsto z] \rangle$	if $p_{pc} = \text{STO}(x)$
$P \vdash \langle pc, e, \sigma \rangle \triangleright \langle pc + k, e, \sigma \rangle$	if $p_{pc} = \text{JMP}(k)$
$P \vdash \langle pc, e : \text{true}, \sigma \rangle \triangleright \langle pc + 1, e, \sigma \rangle$	if $p_{pc} = \text{JMPF}(k)$
$P \vdash \langle pc, e : \text{false}, \sigma \rangle \triangleright \langle pc + k, e, \sigma \rangle$	if $p_{pc} = \text{JMPF}(k)$

Recap: Compiler Correctness

Determinism of Execution

Lemma

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

$$P \vdash \gamma \triangleright \gamma' \text{ and } P \vdash \gamma \triangleright \gamma'' \text{ implies } \gamma' = \gamma''.$$

Proof.

- 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 Programs)

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 (Translation of factorial program)

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

Execution of Factorial Program

Example 5.6 (Execution of factorial program)

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

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

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

Correctness of Expression Translation

Correctness of $\mathfrak{T}_a[\cdot]$ 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:

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

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

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

Correctness of Expression Translation

A New Inductive Principle

The correctness proof requires to embed AM computations into extended machine configurations.

Application: Finite AM computations (Def. 4.7)

Definition: a finite computation $\gamma_0, \gamma_1, \dots, \gamma_k$ has **length k**

Induction base: property holds for all computations of length **0**

Induction hypothesis: property holds for all computations of length $\leq k$

Induction step: property holds for all computations of length $k + 1$

Correctness of Expression Translation

Application: Extension of Code and Stack

Lemma 5.7

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) □

Correctness of Expression Translation

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

Lemma 5.8 (Correctness of $\mathfrak{T}_a[\cdot]$)

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

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

Note: implication sufficient to ensure soundness and completeness as expression evaluation **total** and semantics of machine code **deterministic** (Lemma 4.9)

Proof.

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



Correctness of Expression Translation

Correctness of $\mathfrak{T}_b[\cdot]$ 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:

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

(\vee analogously)

Correctness of Expression Translation

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

Lemma 5.9 (Correctness of $\mathfrak{T}_b[\cdot]$)

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

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

Again: implication sufficient to ensure soundness and completeness as expression evaluation **total** and semantics of machine code **deterministic** (Lemma 4.9)

Proof.

by induction on the syntactic structure of b (omitted)



Correctness of Command Translation

Correctness of $\mathfrak{S}_c[\cdot]$ I

Definition (Recap: Operational functional (Definition 4.1))

The **functional of the operational semantics**,

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

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

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

Definition (Recap: Semantics of machine code (Definition 4.10))

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

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

Correctness of Command Translation

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

Theorem 5.10 (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 steps:

Completeness (Lemma 5.11): from source to machine code

Soundness (Lemma 5.12): from machine to source code



Correctness of Command Translation

Completeness of $\mathfrak{T}_c[\cdot]$

Lemma 5.11 (Completeness of $\mathfrak{T}_c[\cdot]$)

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

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

Proof.

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

Reminder:

$$\mathfrak{T}_c[\text{skip}] := \varepsilon$$

$$\mathfrak{T}_c[x := a] := \mathfrak{T}_a[a]; \text{STO}(x)$$

$$\mathfrak{T}_c[c_1; c_2] := \mathfrak{T}_c[c_1]; \mathfrak{T}_c[c_2]$$

$$\mathfrak{T}_c[\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}] := \mathfrak{T}_b[b]; \text{JMPF}(|\mathfrak{T}_c[c_1]| + 2);$$

$$\mathfrak{T}_c[c_1]; \text{JMP}(|\mathfrak{T}_c[c_2]| + 1); \mathfrak{T}_c[c_2]$$

$$\mathfrak{T}_c[\text{while } b \text{ do } c \text{ end}] := \mathfrak{T}_b[b]; \text{JMPF}(|\mathfrak{T}_c[c]| + 2);$$

$$\mathfrak{T}_c[c]; \text{JMP}(-(|\mathfrak{T}_b[b]| + |\mathfrak{T}_c[c]| + 1))$$

Correctness of Command Translation

Soundness of $\mathfrak{T}_c[\cdot]$

Lemma 5.12 (Soundness of $\mathfrak{T}_c[\cdot]$)

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

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

Proof.

by induction on the length of the computation sequence

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

(see exercises)

