

Semantics and Verification of Software

Lecture 13: Provably Correct Implementation II (The Compiler & Its Correctness)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw13/>

Summer Semester 2013



Sommerfest der Informatik 2013

- 14^h Firmenkontaktmesse
mit Del Negro Eiswagen
- 15^h Veranstaltungsbeginn
mit One Minute Madness der Fitness
- 17^h Absolventenfeier
- 19^h Eröffnung des Buffets
- 20^h Party mit Liveband
Cocktailbar, Bierzelt und Fintunes

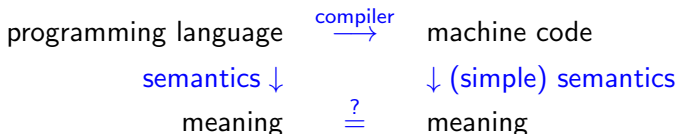
21. Juni
Informatikzentrum

Dieses Jahr mit
Cocktailbar & Eisstand!

Den ganzen Tag!



- 1 Repetition: The Abstract Machine
- 2 The Compiler
- 3 Proof of Compiler Correctness



To do:

- 1 Definition of **abstract machine**
- 2 Definition (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

Definition (Abstract machine)

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

- **configurations** of the form $\langle d, e, \sigma \rangle \in Cnf$ where
 - $d \in Code$ is the **sequence** of instructions (code) to be executed
 - $e \in Stk := (\mathbb{Z} \cup \mathbb{B})^*$ is the **evaluation stack** (top left)
 - $\sigma \in \Sigma := (Var \rightarrow \mathbb{Z})$ is the **(storage) state**

(thus $Cnf = Code \times Stk \times \Sigma$)

- **initial configurations** of the form $\langle d, \varepsilon, \sigma \rangle$
- **final configurations** of the form $\langle \varepsilon, e, \sigma \rangle$
- **code sequences** d and **instructions** i :
$$d ::= \varepsilon \mid i : d$$
$$i ::= \text{PUSH}(z) \mid \text{ADD} \mid \text{MULT} \mid \text{SUB} \mid$$
$$\text{TRUE} \mid \text{FALSE} \mid \text{EQ} \mid \text{GT} \mid \text{AND} \mid \text{OR} \mid \text{NEG} \mid$$
$$\text{LOAD}(x) \mid \text{STORE}(x) \mid \text{NOOP} \mid \text{BRANCH}(d, d) \mid \text{LOOP}(d, d)$$

(where $z \in \mathbb{Z}$ and $x \in Var$)

Lemma

If $\langle d_1, e_1, \sigma \rangle \triangleright^* \langle d', e', \sigma' \rangle$, then

$$\langle d_1 : d_2, e_1 : e_2, \sigma \rangle \triangleright^* \langle d' : d_2, e' : e_2, \sigma' \rangle$$

for every $d_2 \in \text{Code}$ and $e_2 \in \text{Stk}$.

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

Proof.

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



- 1 Repetition: The Abstract Machine
- 2 The Compiler
- 3 Proof of Compiler Correctness

Repetition: Syntax of WHILE Programs

Definition (Syntax of WHILE (Definition 1.2))

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

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

Definition 13.1 (Translation of arithmetic expressions)

The translation function

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

is given by

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

Example 13.2

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

Definition 13.3 (Translation of Boolean expressions)

The translation function

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

is given by

$$\begin{aligned}\mathfrak{T}_b[\text{true}] &:= \text{TRUE} \\ \mathfrak{T}_b[\text{false}] &:= \text{FALSE} \\ \mathfrak{T}_b[a_1 = a_2] &:= \mathfrak{T}_a[a_2] : \mathfrak{T}_a[a_1] : \text{EQ} \\ \mathfrak{T}_b[a_1 > a_2] &:= \mathfrak{T}_a[a_2] : \mathfrak{T}_a[a_1] : \text{GT} \\ \mathfrak{T}_b[\neg b] &:= \mathfrak{T}_b[b] : \text{NEG} \\ \mathfrak{T}_b[b_1 \wedge a_2] &:= \mathfrak{T}_b[b_2] : \mathfrak{T}_b[b_1] : \text{AND} \\ \mathfrak{T}_b[b_1 \vee a_2] &:= \mathfrak{T}_b[b_2] : \mathfrak{T}_b[b_1] : \text{OR}\end{aligned}$$

Definition 13.4 (Translation of statements)

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

$$\begin{aligned}\mathcal{T}_c[\text{skip}] &:= \text{NOOP} \\ \mathcal{T}_c[x := a] &:= \mathcal{T}_a[a] : \text{STORE}(x) \\ \mathcal{T}_c[c_1; c_2] &:= \mathcal{T}_c[c_1] : \mathcal{T}_c[c_2] \\ \mathcal{T}_c[\text{if } b \text{ then } c_1 \text{ else } c_2] &:= \mathcal{T}_b[b] : \text{BRANCH}(\mathcal{T}_c[c_1], \mathcal{T}_c[c_2]) \\ \mathcal{T}_c[\text{while } b \text{ do } c] &:= \text{LOOP}(\mathcal{T}_b[b], \mathcal{T}_c[c])\end{aligned}$$

Example 13.5 (Factorial program)

$$\begin{aligned}\mathcal{T}_c[y:=1; \text{while } \neg(x=1) \text{ do } (y:=y*x; x:=x-1)] \\ &= \mathcal{T}_c[y:=1] : \mathcal{T}_c[\text{while } \neg(x=1) \text{ do } (y:=y*x; x:=x-1)] \\ &= \mathcal{T}_a[1] : \text{STORE}(y) : \text{LOOP}(\mathcal{T}_b[\neg(x=1)], \mathcal{T}_c[y:=y*x; x:=x-1]) \\ &= \text{PUSH}(1) : \text{STORE}(y) : \text{LOOP}(\mathcal{T}_b[x=1] : \text{NEG}, \mathcal{T}_c[y:=y*x] : \mathcal{T}_c[x:=x-1]) \\ &\vdots \\ &= \text{PUSH}(1) : \text{STORE}(y) : \text{LOOP}(\text{PUSH}(1) : \text{LOAD}(x) : \text{EQ} : \text{NEG}, \\ &\quad \text{LOAD}(x) : \text{LOAD}(y) : \text{MULT} : \text{STORE}(y) : \\ &\quad \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x))\end{aligned}$$

Execution of Factorial Program

Example 13.6 (Factorial program)

Let $\sigma \in \Sigma$ with $\sigma(x) = 2$, $d_1 := \text{PUSH}(1) : \text{LOAD}(x) : \text{EQ} : \text{NEG}$, and $d_2 := \text{LOAD}(x) : \text{LOAD}(y) : \text{MULT} : \text{STORE}(y) : \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x)$.

$\langle \text{PUSH}(1) : \text{STORE}(y) : \text{LOOP}(d_1, d_2)$	, ϵ, σ	$\rangle$
$\triangleright \langle \text{STORE}(y) : \text{LOOP}(d_1, d_2)$	, $1, \sigma$	$\rangle$
$\triangleright \langle \text{LOOP}(d_1, d_2)$	, $\epsilon, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle d_1 : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\epsilon, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{LOAD}(x) : \text{EQ} : \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $1, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{EQ} : \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $2 : 1, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\text{false}, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\text{true}, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle d_2 : \text{LOOP}(d_1, d_2)$	, $\epsilon, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{LOAD}(y) : \text{MULT} : \text{STORE}(y) : \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $2, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{MULT} : \text{STORE}(y) : \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $1 : 2, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{STORE}(y) : \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $2, \sigma[y \mapsto 1]$	$\rangle$
$\triangleright \langle \text{PUSH}(1) : \text{LOAD}(x) : \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $\epsilon, \sigma[y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{LOAD}(x) : \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $1, \sigma[y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{SUB} : \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $2 : 1, \sigma[y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{STORE}(x) : \text{LOOP}(d_1, d_2)$	, $1, \sigma[y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{LOOP}(d_1, d_2)$	, $\epsilon, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle d_1 : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\epsilon, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{LOAD}(x) : \text{EQ} : \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $1, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{EQ} : \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $1 : 1, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{NEG} : \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\text{true}, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{BRANCH}(d_2 : \text{LOOP}(d_1, d_2), \text{NOOP})$	, $\text{false}, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \text{NOOP}$	, $\epsilon, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$
$\triangleright \langle \epsilon$	, $\epsilon, \sigma[x \mapsto 1, y \mapsto 2]$	$\rangle$

- 1 Repetition: The Abstract Machine
- 2 The Compiler
- 3 Proof of Compiler Correctness

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

Definition (Repetition: Semantics of arithm. expr. (Def. 4.4))

The (denotational) semantic functional for arithmetic expressions,

$$\mathcal{A}[\cdot] : AExp \rightarrow (\Sigma \rightarrow \mathbb{Z}),$$

is given by:

$$\begin{aligned}\mathcal{A}[z]\sigma &:= z & \mathcal{A}[a_1 + a_2]\sigma &:= \mathcal{A}[a_1]\sigma + \mathcal{A}[a_2]\sigma \\ \mathcal{A}[x]\sigma &:= \sigma(x) & \mathcal{A}[a_1 - a_2]\sigma &:= \mathcal{A}[a_1]\sigma - \mathcal{A}[a_2]\sigma \\ & & \mathcal{A}[a_1 * a_2]\sigma &:= \mathcal{A}[a_1]\sigma * \mathcal{A}[a_2]\sigma\end{aligned}$$

Lemma 13.7 (Correctness of $\mathcal{T}_a[\cdot]$)

For every $a \in AExp$ and $\sigma \in \Sigma$,

$$\langle \mathcal{T}_a[a], \varepsilon, \sigma \rangle \triangleright^* \langle \varepsilon, \mathcal{A}[a]\sigma, \sigma \rangle.$$

Proof.

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

Definition (Repetition: Semantics of Boolean expr. (Def. 4.5))

The (denotational) semantic functional for Boolean expressions,

$$\mathfrak{B}[\cdot] : BExp \rightarrow (\Sigma \rightarrow \mathbb{B}),$$

is given by:

$$\begin{aligned}\mathfrak{B}[t]\sigma &:= t \\ \mathfrak{B}[a_1 = a_2]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{A}[a_1]\sigma = \mathfrak{A}[a_2]\sigma \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[a_1 > a_2]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{A}[a_1]\sigma > \mathfrak{A}[a_2]\sigma \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[\neg b]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{B}[b]\sigma = \text{false} \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[b_1 \wedge b_2]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{B}[b_1]\sigma = \mathfrak{B}[b_2]\sigma = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[b_1 \vee b_2]\sigma &:= \begin{cases} \text{false} & \text{if } \mathfrak{B}[b_1]\sigma = \mathfrak{B}[b_2]\sigma = \text{false} \\ \text{true} & \text{otherwise} \end{cases}\end{aligned}$$

Lemma 13.8 (Correctness of $\mathfrak{T}_b[\![\cdot]\!]$)

For every $b \in BExp$ and $\sigma \in \Sigma$,

$$\langle \mathfrak{T}_b[\![b]\!], \varepsilon, \sigma \rangle \triangleright^* \langle \varepsilon, \mathfrak{B}[\![b]\!]\sigma, \sigma \rangle$$

Proof.

by induction on the syntactic structure of b (omitted)



Definition (Repetition: Operational functional (Def. 4.1))

The **functional of the operational semantics**,

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

assigns to every statement $c \in \text{Cmd}$ a partial state transformation

$\mathfrak{O}[c] : \Sigma \dashrightarrow \Sigma$, which is defined as follows:

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

Definition (Repetition: Semantics of machine code (Def. 12.9))

The **semantics of an instruction sequence** is given by the mapping

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

defined by

$$\mathfrak{M}[d]\sigma := \begin{cases} \sigma' & \text{if } \langle d, \varepsilon, \sigma \rangle \triangleright^* \langle \varepsilon, e, \sigma' \rangle \\ \text{undefined} & \text{otherwise} \end{cases}$$

Theorem 13.9 (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 13.10

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

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

Proof.

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



Second step: from machine to source code

Lemma 13.11

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

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

Proof.

by induction on the length of the computation sequence

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

(see exercises)

