Semantics and Verification of Software

Lecture 17: Provably Correct Implementation II (Compiler & Its Correctness)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw11/

Winter Semester 2011/12

Outline

Compiler Correctness

$$\begin{array}{rcl} \text{programming language} & \xrightarrow{\text{compiler}} & \text{machine code} \\ \text{semantics} \downarrow & & \downarrow \text{(simple) semantics} \\ \text{meaning} & \stackrel{?}{=} & \text{meaning} \end{array}$$

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

The Abstract Machine

Definition (Abstract machine)

The abstract machine (AM) is given by

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

  (thus $\mathit{Cnf} = \mathit{Code} \times \mathit{Stk} \times \Sigma$)
- 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:

$$
\begin{aligned}
d &::= \varepsilon \mid i : d \\
i &::= \texttt{PUSH}(z) \mid \texttt{ADD} \mid \texttt{MULT} \mid \texttt{SUB} \mid \\
&\quad\ \texttt{TRUE} \mid \texttt{FALSE} \mid \texttt{EQ} \mid \texttt{GT} \mid \texttt{AND} \mid \texttt{OR} \mid \texttt{NEG} \mid \\
&\quad\ \texttt{LOAD}(x) \mid \texttt{STORE}(x) \mid \texttt{NOOP} \mid \texttt{BRANCH}(d,d) \mid \texttt{LOOP}(d,d)
\end{aligned}
$$

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

Extension of Code and Stack

Lemma

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

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

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

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

Proof.

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

Outline

Another Property: Determinism

Lemma 17.1

The semantics of AM is deterministic: for all $\gamma, \gamma', \gamma'' \in \mathit{Cnf}$,
$\gamma \rhd \gamma'$ *and* $\gamma \rhd \gamma''$ *imply* $\gamma' = \gamma''$.

Proof.

The successor configuration is determined by the first instruction in the code component, which is unique. □

Thus the following function is well defined:

Definition 17.2 (Semantics of AM)

The semantics of an instruction sequence is given by the mapping

$$\mathfrak{M}[\![.]\!] : \mathit{Code} \to (\Sigma \dashrightarrow \Sigma),$$

defined by

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

Outline

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:

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

Translation of Arithmetic Expressions

Definition 17.3 (Translation of arithmetic expressions)

The translation function

$$\mathfrak{T}_a[\![.]\!] : \mathit{AExp} \to \mathit{Code}$$

is given by

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

Example 17.4

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

Translation of Boolean Expressions

Definition 17.5 (Translation of Boolean expressions)

The translation function

$$\mathfrak{T}_b[\![.]\!] : \mathit{BExp} \to \mathit{Code}$$

is given by

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

Translation of Statements

Definition 17.6 (Translation of statements)

The translation function $\mathfrak{T}_c[\![.]\!] : \mathit{Cmd} \to \mathit{Code}$ is given by

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

Example 17.7 (Factorial program)

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

Execution of Factorial Program

Example 17.8 (Factorial program)

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

$$
\begin{array}{ll}
 & \langle \texttt{PUSH(1):STORE(y):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{STORE(y):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{LOOP}(d_1,d_2) \\
\triangleright & \langle d_1\texttt{:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{LOAD(x):EQ:NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{EQ:NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle d_2\texttt{:LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{LOAD(y):MULT:STORE(y):PUSH(1):LOAD(x):SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{MULT:STORE(y):PUSH(1):LOAD(x):SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{STORE(y):PUSH(1):LOAD(x):SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{PUSH(1):LOAD(x):SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{LOAD(x):SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{SUB:STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{STORE(x):LOOP}(d_1,d_2) \\
\triangleright & \langle \texttt{LOOP}(d_1,d_2) \\
\triangleright & \langle d_1\texttt{:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{LOAD(x):EQ:NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{EQ:NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{NEG:BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{BRANCH}(d_2\texttt{:LOOP}(d_1,d_2),\texttt{NOOP}) \\
\triangleright & \langle \texttt{NOOP} \\
\triangleright & \langle \varepsilon
\end{array}
\quad
\begin{array}{r}
, \quad \varepsilon, \sigma \rangle \\
, \quad 1, \sigma \rangle \\
, \quad \varepsilon, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad 1, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, 2:1, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \text{false}, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \text{true}, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad 2, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, 1:2, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad 2, \sigma[\mathrm{y} \mapsto 1] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{y} \mapsto 2] \rangle \\
, \quad 1, \sigma[\mathrm{y} \mapsto 2] \rangle \\
, 2:1, \sigma[\mathrm{y} \mapsto 2] \rangle \\
, \quad 1, \sigma[\mathrm{y} \mapsto 2] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \quad 1, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, 1:1, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \text{true}, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \text{false}, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle \\
, \quad \varepsilon, \sigma[\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2] \rangle
\end{array}
$$

Outline

Correctness of $\mathfrak{T}_a[\![.]\!]$

Definition (Repetition: Semantics of arithm. expr. (Def. 5.1))

The (denotational) semantic functional for arithmetic expressions,

$$\mathfrak{A}[\![.]\!] : AExp \to (\Sigma \to \mathbb{Z}),$$

is given by:

$$\mathfrak{A}[\![z]\!]\sigma := z \qquad \mathfrak{A}[\![a_1 + a_2]\!]\sigma := \mathfrak{A}[\![a_1]\!]\sigma + \mathfrak{A}[\![a_2]\!]\sigma$$
$$\mathfrak{A}[\![x]\!]\sigma := \sigma(x) \qquad \mathfrak{A}[\![a_1 - a_2]\!]\sigma := \mathfrak{A}[\![a_1]\!]\sigma - \mathfrak{A}[\![a_2]\!]\sigma$$
$$\mathfrak{A}[\![a_1 * a_2]\!]\sigma := \mathfrak{A}[\![a_1]\!]\sigma * \mathfrak{A}[\![a_2]\!]\sigma$$

Lemma 17.9 (Correctness of $\mathfrak{T}_a[\![.]\!]$)

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

$$\langle \mathfrak{T}_a[\![a]\!], \varepsilon, \sigma \rangle \rhd^* \langle \varepsilon, \mathfrak{A}[\![a]\!]\sigma, \sigma \rangle.$$

Proof.

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

Correctness of $\mathfrak{T}_b[\![.]\!]$ I

Definition (Repetition: Semantics of Boolean expr. (Def. 5.2))

The (denotational) semantic functional for Boolean expressions,

$$\mathfrak{B}[\![.]\!] : BExp \to (\Sigma \to \mathbb{B}),$$

is given by:

$$\mathfrak{B}[\![t]\!]\sigma := t$$

$$\mathfrak{B}[\![a_1 = a_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma = \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![a_1 > a_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![\neg b]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b]\!]\sigma = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![b_1 \wedge b_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![b_1 \vee b_2]\!]\sigma := \begin{cases} \text{false} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{false} \\ \text{true} & \text{otherwise} \end{cases}$$

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

Lemma 17.10 (Correctness of $\mathfrak{T}_b[\![.]\!]$)

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

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

Proof.

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

Correctness of $\mathfrak{T}_c[\![.]\!]$ I

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

The functional of the operational semantics,

$$\mathfrak{O}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma),$$

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

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

Definition (Repetition: Semantics of machine code (Def. 17.2))

The semantics of an instruction sequence is given by the mapping

$$\mathfrak{M}[\![.]\!] : \mathit{Code} \to (\Sigma \dashrightarrow \Sigma),$$

defined by

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

Correctness of $\mathfrak{T}_c[\![.]\!]$ II

Theorem 17.11 (Correctness of $\mathfrak{T}_c[\![.]\!]$)

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

$$\mathfrak{O}[\![c]\!] = \mathfrak{M}[\![\mathfrak{T}_c[\![c]\!]]\!].$$

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

Lemma 17.12

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

$$\langle c, \sigma\rangle \to \sigma' \text{ implies } \langle \mathfrak{T}_c[\![c]\!], \varepsilon, \sigma\rangle \rhd^* \langle \varepsilon, \varepsilon, \sigma'\rangle.$$

Proof.

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

Correctness of $\mathfrak{T}_c[\![.]\!]$ III

Second step: from machine to source code

Lemma 17.13

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

$$\langle \mathfrak{T}_c[\![c]\!], \varepsilon, \sigma \rangle \rhd^* \langle \varepsilon, e, \sigma' \rangle \text{ implies } \langle c, \sigma \rangle \to \sigma' \text{ and } e = \varepsilon.$$

Proof.

by induction on the length of the computation sequence $\langle \mathfrak{T}_c[\![c]\!], \varepsilon, \sigma \rangle \rhd^* \langle \varepsilon, e, \sigma' \rangle$ (see Exercise 10.3) □