Semantics and Verification of Software

Lecture 25: Provably Correct Implementation II (Compiler 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/svsw08/

Winter semester 2008/09

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:

- Definition of abstract machine
- Definition (operational) semantics of machine instructions
- Definition of translation WHILE $\rightarrow$ machine code ("compiler")
- Proof: semantics of generated machine code = semantics of original source code

The Abstract Machine

Definition

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
  - $\sigma \in \Sigma := (\mathit{Var} \to \mathbb{Z})$ is the (storage) state

  (thus $\mathit{Cnf} = \mathit{Code} \times \mathit{Stk} \times \Sigma$)
- final configurations of the form $\langle \varepsilon, e, \sigma \rangle$
- code sequences and instructions:

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

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

Translation of Arithmetic Expressions

Definition

The translation function

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

is given by

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

Example

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

Translation of Boolean Expressions

Definition

The translation function

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

is given by

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

Translation of Statements

Definition

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

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

Example (Factorial program)

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

Outline

Execution of Factorial Program

Example 25.1 (Factorial program)

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

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

Outline

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

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

The (denotational) semantic functional for arithmetic expressions,

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

is given by:

$$\begin{array}{rclcrcl}
\mathfrak{A}[\![z]\!]\sigma & := & z & \quad & \mathfrak{A}[\![a_1\texttt{+}a_2]\!]\sigma & := & \mathfrak{A}[\![a_1]\!]\sigma + \mathfrak{A}[\![a_2]\!]\sigma \\
\mathfrak{A}[\![x]\!]\sigma & := & \sigma(x) & & \mathfrak{A}[\![a_1\texttt{-}a_2]\!]\sigma & := & \mathfrak{A}[\![a_1]\!]\sigma - \mathfrak{A}[\![a_2]\!]\sigma \\
 & & & & \mathfrak{A}[\![a_1\texttt{*}a_2]\!]\sigma & := & \mathfrak{A}[\![a_1]\!]\sigma * \mathfrak{A}[\![a_2]\!]\sigma
\end{array}$$

Lemma 25.2 (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) $\square$

Definition (Semantics of Boolean expressions (Def. 4.5))

The (denotational) semantic functional for Boolean expressions,

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

is given by:

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

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

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

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

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

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

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

Lemma 25.3 (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) $\square$

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

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

The functional of the operational semantics,

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

assigns to every statement $c \in Cmd$ 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. 24.8))

The semantics of an instruction sequence is given by the mapping

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

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

For every $c \in Cmd$,

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

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

Lemma 25.5

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

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

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 25.6

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

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

Proof.

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