COSE312: Compilers

Lecture 9 — Translation (1)

Hakjoo Oh
2026 Spring

While: Syntax

n will range over numerals, **Num**
x will range over variables, **Var**
a will range over arithmetic expressions, **Aexp**
b will range over boolean expressions, **Bexp**
S will range over statements, **Stm**

$$
\begin{array}{lcl}
a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & \rightarrow & x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if}\ b\ S_1\ S_2 \mid \texttt{while}\ b\ S
\end{array}
$$

While: Semantics

- The semantics of arithmetic expressions is defined by function $\mathcal{A}[\![\ a\ ]\!]$:

$$
\begin{array}{rcl}
\mathcal{A}[\![\ a\ ]\!] & : & \mathbf{State} \to \mathbb{Z} \\
\mathcal{A}[\![\ n\ ]\!](s) & = & n \\
\mathcal{A}[\![\ x\ ]\!](s) & = & s(x) \\
\mathcal{A}[\![\ a_1 + a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) + \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{A}[\![\ a_1 \star a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) \times \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{A}[\![\ a_1 - a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) - \mathcal{A}[\![\ a_2\ ]\!](s)
\end{array}
$$

- The semantics of boolean expressions is defined by function $\mathcal{B}[\![\ b\ ]\!]$:

$$
\begin{array}{rcl}
\mathcal{B}[\![\ b\ ]\!] & : & \mathbf{State} \to \mathbf{T} \\
\mathcal{B}[\![\ \mathtt{true}\ ]\!](s) & = & \mathit{true} \\
\mathcal{B}[\![\ \mathtt{false}\ ]\!](s) & = & \mathit{false} \\
\mathcal{B}[\![\ a_1 = a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) = \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{B}[\![\ a_1 \leq a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) \leq \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{B}[\![\ \neg b\ ]\!](s) & = & \mathcal{B}[\![\ b\ ]\!](s) = \mathit{false} \\
\mathcal{B}[\![\ b_1 \wedge b_2\ ]\!](s) & = & \mathcal{B}[\![\ b_1\ ]\!](s) \wedge \mathcal{B}[\![\ b_2\ ]\!](s)
\end{array}
$$

- The semantics of statements is defined by function $\mathcal{C}[\![S]\!]$:

$$\begin{array}{rcl} \mathcal{C}[\![S]\!] & : & \mathbf{State} \hookrightarrow \mathbf{State} \\ \mathcal{C}[\![S]\!](s) & = & \begin{cases} s' & \text{if } \langle S, s\rangle \to s' \\ \mathbf{undef} & \text{otherwise} \end{cases} \end{array}$$

where transition relation $(\to) \subseteq \mathbf{State} \times \mathbf{State}$ is defined by the rules:

$$\textsc{B-Assn} \quad \frac{}{\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!](s)]}$$

$$\textsc{B-Skip} \quad \frac{}{\langle \mathtt{skip}, s\rangle \to s}$$

$$\textsc{B-Seq} \quad \frac{\langle S_1, s\rangle \to s'' \qquad \langle S_2, s''\rangle \to s'}{\langle S_1; S_2, s\rangle \to s'}$$

$$\textsc{B-IfT} \quad \frac{\langle S_1, s\rangle \to s'}{\langle \mathtt{if}\ b\ S_1\ S_2, s\rangle \to s'} \ \text{if } \mathcal{B}[\![b]\!](s) = \mathit{true}$$

$$\textsc{B-IfF} \quad \frac{\langle S_2, s\rangle \to s'}{\langle \mathtt{if}\ b\ S_1\ S_2, s\rangle \to s'} \ \text{if } \mathcal{B}[\![b]\!](s) = \mathit{false}$$

$$\textsc{B-WhileT} \quad \frac{\langle S, s\rangle \to s'' \qquad \langle \mathtt{while}\ b\ S, s''\rangle \to s'}{\langle \mathtt{while}\ b\ S, s\rangle \to s'} \ \text{if } \mathcal{B}[\![b]\!](s) = \mathit{true}$$

$$\textsc{B-WhileF} \quad \frac{}{\langle \mathtt{while}\ b\ S, s\rangle \to s} \ \text{if } \mathcal{B}[\![b]\!](s) = \mathit{false}$$

Abstract Machine **M**

Instructions and code:

$$
\begin{array}{rcl}
inst & \rightarrow & \mathsf{push}(n) \\
& | & \mathsf{add} \\
& | & \mathsf{mult} \\
& | & \mathsf{sub} \\
& | & \mathsf{true} \\
& | & \mathsf{false} \\
& | & \mathsf{eq} \\
& | & \mathsf{le} \\
& | & \mathsf{and} \\
& | & \mathsf{neg} \\
& | & \mathsf{fetch}(x) \\
& | & \mathsf{store}(x) \\
& | & \mathsf{noop} \\
& | & \mathsf{branch}(c, c) \\
& | & \mathsf{loop}(c, c) \\
\mathbf{Code} \ni c & \rightarrow & \epsilon \\
& | & inst :: c
\end{array}
$$

Small-Step Operational Semantics

A configuration of **M** consists of three components:

$$\langle c, e, s \rangle \in \mathbf{Code} \times \mathbf{Stack} \times \mathbf{State}$$

- c is a sequence of instructions (code) to be executed.
- e is an evaluation stack. An evaluation stack is a list of values:

  $$\mathbf{Stack} = (\mathbb{Z} \cup \mathbf{T})^*$$

  and used to evaluate arithmetic and boolean expressions.
- s is a memory state. A memory state s maps variables to values:

  $$\mathbf{State} = \mathbf{Var} \to \mathbb{Z}$$

A configuration is a terminal (or final) configuration if its code component is the empty sequence: i.e., $\langle \epsilon, e, s \rangle$ for some e and s.

Transition Relation: $\langle c, e, s \rangle \triangleright \langle c', e', s' \rangle$

$$
\begin{array}{lcl}
\langle \mathsf{push}(n) :: c, e, s \rangle & \triangleright & \langle c, n :: e, s \rangle \\
\langle \mathsf{add} :: c, z_1 :: z_2 :: e, s \rangle & \triangleright & \langle c, (z_1 + z_2) :: e, s \rangle \\
\langle \mathsf{mult} :: c, z_1 :: z_2 :: e, s \rangle & \triangleright & \langle c, (z_1 \star z_2) :: e, s \rangle \\
\langle \mathsf{sub} :: c, z_1 :: z_2 :: e, s \rangle & \triangleright & \langle c, (z_1 - z_2) :: e, s \rangle \\
\langle \mathsf{true} :: c, e, s \rangle & \triangleright & \langle c, \mathit{true} :: e, s \rangle \\
\langle \mathsf{false} :: c, e, s \rangle & \triangleright & \langle c, \mathit{false} :: e, s \rangle \\
\langle \mathsf{eq} :: c, z_1 :: z_2 :: e, s \rangle & \triangleright & \langle c, (z_1 = z_2) :: e, s \rangle \\
\langle \mathsf{le} :: c, z_1 :: z_2 :: e, s \rangle & \triangleright & \langle c, (z_1 \leq z_2) :: e, s \rangle \\
\langle \mathsf{and} :: c, t_1 :: t_2 :: e, s \rangle & \triangleright & \begin{cases} \langle c, \mathit{true} :: e, s \rangle & \text{if } t_1 = \mathit{true} \text{ and } t_2 = \mathit{true} \\ \langle c, \mathit{false} :: e, s \rangle & \text{otherwise} \end{cases} \\
\langle \mathsf{neg} :: c, t :: e, s \rangle & \triangleright & \begin{cases} \langle c, \mathit{true} :: e, s \rangle & \text{if } t = \mathit{false} \\ \langle c, \mathit{false} :: e, s \rangle & \text{otherwise} \end{cases} \\
\langle \mathsf{fetch}(x) :: c, e, s \rangle & \triangleright & \langle c, s(x) :: e, s \rangle \\
\langle \mathsf{store}(x) :: c, z :: e, s \rangle & \triangleright & \langle c, e, s[x \mapsto z] \rangle \\
\langle \mathsf{noop} :: c, e, s \rangle & \triangleright & \langle c, e, s \rangle \\
\langle \mathsf{branch}(c_1, c_2) :: c, t :: e, s \rangle & \triangleright & \begin{cases} \langle c_1 :: c, e, s \rangle & \text{if } t = \mathit{true} \\ \langle c_2 :: c, e, s \rangle & \text{otherwise} \end{cases} \\
\langle \mathsf{loop}(c_1, c_2) :: c, e, s \rangle & \triangleright & \langle c_1 :: \mathsf{branch}(c_2 :: \mathsf{loop}(c_1, c_2), \mathsf{noop}) :: c, e, s \rangle
\end{array}
$$

Semantic Function

The semantics of code $c \in \mathbf{Code}$ is defined by the partial function:

$$\begin{array}{rcl} \mathcal{M}[\![\, c \,]\!] & : & \mathbf{State} \hookrightarrow \mathbf{State} \\ \mathcal{M}[\![\, c \,]\!](s) & = & \begin{cases} s' & \text{if } \langle c, \epsilon, s \rangle \rhd^* \langle \epsilon, e, s' \rangle \\ \mathbf{undef} & \text{otherwise} \end{cases} \end{array}$$

Compilation Rules

- $\mathcal{T}_A : \textbf{Aexp} \to \textbf{Code}$:

$$\begin{array}{rcl} \mathcal{T}_A(n) &=& \mathsf{push}(n) \\ \mathcal{T}_A(x) &=& \mathsf{fetch}(x) \\ \mathcal{T}_A(a_1 + a_2) &=& \mathcal{T}_A(a_2) :: \mathcal{T}_A(a_1) :: \mathsf{add} \\ \mathcal{T}_A(a_1 \star a_2) &=& \mathcal{T}_A(a_2) :: \mathcal{T}_A(a_1) :: \mathsf{mult} \\ \mathcal{T}_A(a_1 - a_2) &=& \mathcal{T}_A(a_2) :: \mathcal{T}_A(a_1) :: \mathsf{sub} \end{array}$$

- $\mathcal{T}_B : \textbf{Bexp} \to \textbf{Code}$:

$$\begin{array}{rcl} \mathcal{T}_B(\texttt{true}) &=& \mathsf{true} \\ \mathcal{T}_B(\texttt{false}) &=& \mathsf{false} \\ \mathcal{T}_B(a_1 = a_2) &=& \mathcal{T}_A(a_2) :: \mathcal{T}_A(a_1) :: \mathsf{eq} \\ \mathcal{T}_B(a_1 \le a_2) &=& \mathcal{T}_A(a_2) :: \mathcal{T}_A(a_1) :: \mathsf{le} \\ \mathcal{T}_B(\neg b) &=& \mathcal{T}_B(b) :: \mathsf{neg} \\ \mathcal{T}_B(b_1 \land b_2) &=& \mathcal{T}_B(b_2) :: \mathcal{T}_B(b_1) :: \mathsf{and} \end{array}$$

- $\mathcal{T}_S : \textbf{Stm} \to \textbf{Code}$:

$$\begin{array}{rcl} \mathcal{T}_S(x := a) &=& \mathcal{T}_A(a) :: \mathsf{store}(x) \\ \mathcal{T}_S(\texttt{skip}) &=& \mathsf{noop} \\ \mathcal{T}_S(S_1; S_2) &=& \mathcal{T}_S(S_1) :: \mathcal{T}_S(S_2) \\ \mathcal{T}_S(\texttt{if}\ b\ S_1\ S_2) &=& \mathcal{T}_B(b) :: \mathsf{branch}(\mathcal{T}_S(S_1), \mathcal{T}_S(S_2)) \\ \mathcal{T}_S(\texttt{while}\ b\ S) &=& \mathsf{loop}(\mathcal{T}_B(b), \mathcal{T}_S(S)) \end{array}$$

Compiler Correctness

Theorem

For any statement S of **While** *and a memory state $s \in \mathbf{State}$,*

$$\mathcal{C}[\![\ S\ ]\!](s) = \mathcal{M}[\![\ \mathcal{T}_S(S)\ ]\!](s).$$

Proof) By Lemma (1) and (2).

Lemma (1)

For every statement S of **While** *and states s and s',*

$$\text{if } \langle S, s\rangle \to s' \text{ then } \langle \mathcal{T}_S(S), \epsilon, s\rangle \rhd^* \langle \epsilon, \epsilon, s'\rangle.$$

Proof) By induction on the derivation of $\langle S, s\rangle \to s'$.

Lemma (2)

For every statement S of **While** *and states s and s',*

$$\text{if } \langle \mathcal{T}_S(S), \epsilon, s\rangle \rhd^k \langle \epsilon, e, s'\rangle \text{ then } \langle S, s\rangle \to s' \text{ and } e = \epsilon.$$

Proof) By induction on the length k of the computation sequence.

Auxiliary Lemmas

Lemma (3)

If $\langle c_1, e_1, s\rangle \rhd^k \langle c', e', s'\rangle$ *then* $\langle c_1 :: c_2, e_1 :: e_2, s\rangle \rhd^k \langle c' :: c_2, e' :: e_2, s'\rangle$.

Lemma (4)

If $\langle c_1 :: c_2, e, s\rangle \rhd^k \langle \epsilon, e'', s''\rangle$ *then there exists a configuration* $\langle \epsilon, e', s'\rangle$ *and natural numbers* k_1 *and* k_2 *with* $k_1 + k_2 = k$ *such that*

$$\langle c_1, e, s\rangle \rhd^{k_1} \langle \epsilon, e', s'\rangle \text{ and } \langle c_2, e', s'\rangle \rhd^{k_2} \langle \epsilon, e'', s''\rangle.$$

Lemma (5)

The abstract machine is deterministic, i.e., for all choices $\gamma, \gamma', \gamma''$,

$$\gamma \rhd \gamma' \text{ and } \gamma \rhd \gamma'' \text{ impliy } \gamma' = \gamma''.$$

From Lemma (5), we can deduce that there is exactly one computation sequence starting in a configuration $\langle c, e, s\rangle$.

Auxiliary Lemmas

Lemma (6)

For all arithmetic expression $a \in \mathbf{Aexp}$*,*

$$\langle \mathcal{T}_A(a), \epsilon, s\rangle \triangleright^* \langle \epsilon, \mathcal{A}[\![a]\!](s), s\rangle$$

and all intermediate stacks appearing in this computation sequence are non-empty.

Lemma (7)

For all boolean expression $b \in \mathbf{Bexp}$*,*

$$\langle \mathcal{T}_B(b), \epsilon, s\rangle \triangleright^* \langle \epsilon, \mathcal{B}[\![b]\!](s), s\rangle$$

and all intermediate stacks appearing in this computation sequence are non-empty.

Summary

Designed a simple compiler and proved its correctness:

- Source language: **While**.
- Target language: $\mathbf{M}$.
- Compilation rules: $\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_S$
- Correctness theorem