Compiler Construction

Lecture 21: Code Generation V (Translation of Static Data Structures)

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/cc09/`

Winter semester 2009/10

Outline

Modified Syntax of EPL

Definition (Modified syntax of EPL)

The modified syntax of EPL is defined as follows (where $n \geq 1$):

$$
\begin{array}{lrll}
\mathbb{Z}: & & z & \text{(* } z \text{ is an integer *)} \\
\mathbb{B}: & b & ::= \texttt{true} \mid \texttt{false} & \text{(* } b \text{ is a Boolean *)} \\
\mathbb{R}: & & r & \text{(* } r \text{ is a real number *)} \\
Con: & c & ::= z \mid b \mid r & \text{(* } c \text{ is a constant *)} \\
Ide: & & I & \text{(* } I \text{ is an identifier *)} \\
Type: & T & ::= \texttt{bool} \mid \texttt{int} \mid \texttt{real} \mid I \mid \texttt{array[}z_1\texttt{..}z_2\texttt{] of } T \mid & \\
 & & \quad \texttt{record } I_1\texttt{:}T_1\texttt{;} \ldots \texttt{;}I_n\texttt{:}T_n \texttt{ end} & \\
Var: & V & ::= I \mid V\texttt{[}E\texttt{]} \mid V\texttt{.}I & \\
Exp: & E & ::= c \mid V \mid E_1 \texttt{ + } E_2 \mid E_1 \texttt{ < } E_2 \mid E_1 \texttt{ and } E_2 \mid \ldots & \\
Cmd: & C & ::= V\texttt{:=}E \mid C_1\texttt{;}C_2 \mid & \\
 & & \quad \texttt{if } E \texttt{ then } C_1 \texttt{ else } C_2 \mid \texttt{while } E \texttt{ do } C & \\
Dcl: & D & ::= D_C\ D_T\ D_V & \\
 & D_C & ::= \varepsilon \mid \texttt{const } I_1 \texttt{ := } c_1\texttt{;} \ldots \texttt{;}I_n \texttt{ := } c_n\texttt{;} & \\
 & D_T & ::= \varepsilon \mid \texttt{type } I_1 \texttt{ := } T_1\texttt{;} \ldots \texttt{;}I_n \texttt{ := } T_n\texttt{;} & \\
 & D_V & ::= \varepsilon \mid \texttt{var } I_1 \texttt{ : } T_1\texttt{;} \ldots \texttt{;}I_n \texttt{ : } T_n\texttt{;} & \\
Pgm: & P & ::= D\ C &
\end{array}
$$

Since (recursive) procedures are no longer supported, a procedure stack is not required anymore.

Definition (Modified abstract machine for EPL)

The modified abstract machine for EPL (AM) is defined by the state space

$$S := PC \times DS \times MS$$

with

- the program counter $PC := \mathbb{N}$,
- the data stack $DS := \mathbb{R}^*$, and
- the main storage $MS := \{\sigma \mid \sigma : \mathbb{N} \to \mathbb{R}\}$.

New AM Instructions

Definition (New AM instructions)

- Procedure instructions are no longer needed.
- Transfer instructions ($\texttt{LOAD(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}$, $\texttt{STORE(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}$) are replaced by the following instructions with the respective semantics $[\![O]\!] : S \dashrightarrow S$:

$$
\begin{aligned}
[\![\texttt{LOAD}]\!](a, d : n, \sigma) &:= (a+1, d : \sigma(n), \sigma) \\
&\ \text{if } n \in \mathbb{N} \\
[\![\texttt{STORE}]\!](a, d : n : r, \sigma) &:= (a+1, d, \sigma[n \mapsto r]) \\
&\ \text{if } n \in \mathbb{N}
\end{aligned}
$$

- Moreover the following instruction for checking array bounds is introduced:

$$
[\![\texttt{CAB(}z_1\texttt{,}z_2\texttt{)}]\!](a, d : z, \sigma) := \begin{cases} (a+1, d : z, \sigma) & \text{if } z \in \{z_1, \ldots, z_2\} \\ (0, d : \underbrace{\texttt{RTE}}_{\text{runtime error}}, \sigma) & \text{otherwise} \end{cases}
$$

Modifying the Symbol Table

$$
\begin{aligned}
Tab := \{\mathrm{st} \mid \mathrm{st} : Ide \dashrightarrow\ & (\{\texttt{const}\} \times (\mathbb{B} \cup \mathbb{Z} \cup \mathbb{R})) \\
\cup\ & (\{\texttt{var}\} \times Ide \times \mathbb{N}) \\
\cup\ & (\{\texttt{type}\} \times \{\texttt{bool}, \texttt{int}, \texttt{real}\} \times \{1\}) \\
\cup\ & (\{\texttt{type}\} \times \{\texttt{array}\} \times \mathbb{Z}^2 \times Ide \times \mathbb{N}) \\
\cup\ & (\{\texttt{type}\} \times \{\texttt{record}\} \times (Ide^2 \times \mathbb{N})^* \times \mathbb{N})\}
\end{aligned}
$$

Remarks:

- Variable descriptor $(\texttt{var}, I, n)$: type I, memory address n
- Last component of `type` entry: memory requirement (base types: 1 "cell")
- Array descriptor $(\texttt{type}, \texttt{array}, z_1, z_2, I, n)$: bounds z_1, z_2, component type I
- Record descriptor $(\texttt{type}, \texttt{record}, I_1, J_1, o_1, \ldots, I_l, J_l, o_l, n)$: selector I_k, component type J_k, memory offset o_k
- "Indexed" table lookup: $\mathrm{st}(I.I_k) := (J_k, o_k)$ if $\mathrm{st}(I) = (\texttt{type}, \texttt{record}, \ldots, I_k, J_k, o_k, \ldots, n)$

Outline

Translation of Variables I

The translation employs the following auxiliary function to determine the type identifier of a given variable:

Definition 21.1 (vtype function)

The mapping

$$\text{vtype} : \mathit{Var} \times \mathit{Tab} \dashrightarrow \mathit{Ide}$$

is given by

$$\begin{array}{rcl}
\text{vtype}(I, \text{st}) & := & J \\
 & & \text{if } \text{st}(I) = (\texttt{var}, J, n) \\
\text{vtype}(V\,\texttt{[}E\texttt{]}, \text{st}) & := & J \\
 & & \text{if } \text{vtype}(V, \text{st}) = I \\
 & & \text{and } \text{st}(I) = (\texttt{type}, \texttt{array}, z_1, z_2, J, n) \\
\text{vtype}(V\,\texttt{.}\,I, \text{st}) & := & J \\
 & & \text{if } \text{vtype}(V, \text{st}) = I' \text{ and } \text{st}(I'.I) = (J, o)
\end{array}$$

Translation of Variables II

The function vt generates code for computing the memory address of a variable (and storing it on the data stack):

Definition 21.2 (Translation of variables)

The mapping

$$\mathrm{vt} : \mathit{Var} \times \mathit{Tab} \dashrightarrow \mathit{AM}$$

is given by

$$
\begin{array}{rll}
\mathrm{vt}(I,\mathrm{st}) := & \texttt{LIT(}n\texttt{);} & \\
 & \text{if } \mathrm{st}(I) = (\texttt{var}, J, n) & \\
\mathrm{vt}(V\texttt{[}E\texttt{]},\mathrm{st}) := & \mathrm{vt}(V,\mathrm{st}) & \text{\% address of } V \\
 & \mathrm{et}(E,\mathrm{st}) & \text{\% array index} \\
 & \texttt{CAB(}z_1\texttt{,}z_2\texttt{);} & \text{\% bounds checking} \\
 & \texttt{LIT(}z_1\texttt{); SUB;} & \text{\% index difference} \\
 & \texttt{LIT(}n\texttt{); MULT;} & \text{\% relative address} \\
 & \texttt{ADD;} & \text{\% address of } V\texttt{[}E\texttt{]} \\
 & \multicolumn{2}{l}{\text{if } \mathrm{vtype}(V,\mathrm{st}) = I \text{ and } \mathrm{st}(I) = (\texttt{type}, \texttt{array}, z_1, z_2, J, m)} \\
 & \multicolumn{2}{l}{\text{and } \mathrm{st}(J) = (\texttt{type}, \ldots, n)} \\
\mathrm{vt}(V\texttt{.}I,\mathrm{st}) := & \mathrm{vt}(V,\mathrm{st}) & \text{\% address of } V \\
 & \texttt{LIT(}o\texttt{);} & \text{\% offset} \\
 & \texttt{ADD;} & \text{\% address of } V\texttt{.}I \\
 & \multicolumn{2}{l}{\text{if } \mathrm{vtype}(V,\mathrm{st}) = I' \text{ and } \mathrm{st}(I'.I) = (J, o)}
\end{array}
$$

Translation of Expressions

Definition 21.3 (Translation of expressions)

The mapping

$$\mathrm{et} : \mathit{Exp} \times \mathit{Tab} \dashrightarrow \mathit{AM}$$

is given by

$$
\begin{aligned}
\mathrm{et}(c, \mathrm{st}) \quad &:= \quad \texttt{LIT(}c\texttt{);} \\
\mathrm{et}(V, \mathrm{st}) \quad &:= \quad \begin{cases} \texttt{LIT(}c\texttt{);} & \text{if } V \in \mathit{Ide} \text{ and } \mathrm{st}(V) = (\texttt{const}, c) \\ \mathrm{vt}(V, \mathrm{st}) & \text{if } \mathrm{st}(V) = (\texttt{var}, J, n) \\ \texttt{LOAD;} & \end{cases} \\
\mathrm{et}(E_1 \texttt{+} E_2, \mathrm{st}) \quad &:= \quad \mathrm{et}(E_1, \mathrm{st}) \\
& \quad \mathrm{et}(E_2, \mathrm{st}) \\
& \quad \texttt{ADD;} \\
&\vdots
\end{aligned}
$$

Translation of Commands and Programs

Definition 21.4 (Translation of commands)

For the mapping

$$\mathrm{ct} : \mathit{Cmd} \times \mathit{Tab} \dashrightarrow \mathit{AM}$$

only the handling of assignments needs to be adapted:

$$\begin{aligned}\mathrm{ct}(V \texttt{:=} E, \mathrm{st}) := {} & \mathrm{vt}(V, \mathrm{st}) && \text{\% address of left-hand side} \\ & \mathrm{et}(E, \mathrm{st}) && \text{\% value of right-hand side} \\ & \texttt{STORE;} \end{aligned}$$

Definition 21.5 (Translation of programs)

The mapping

$$\mathrm{trans} : \mathit{Pgm} \dashrightarrow \mathit{AM}$$

is defined by

$$\mathrm{trans}(D\ C) := \mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}_\emptyset))$$

Outline

Translation Example I

Example 21.6

$$P = \left.\begin{array}{l}\texttt{type Int=int; Array=array[1..10] of Int;}\\ \texttt{var a:Array; i:Int;}\end{array}\right\} D$$
$$\left.\begin{array}{l}\texttt{i:=1;}\\ \texttt{while i<=10 do}\\ \quad\texttt{a[i]:=i; i:=i+1;}\end{array}\right\} C$$

$$\begin{aligned}
\text{trans}(P) &= \text{ct}(C, \text{update}(D, \text{st}_\emptyset))\\
\text{st} &:= \text{update}(D, \text{st}_\emptyset)\\
&= \text{st}_\emptyset[\ \ \texttt{Int} \mapsto (\texttt{type}, \texttt{int}, 1),\\
&\qquad \texttt{Array} \mapsto (\texttt{type}, \texttt{array}, 1, 10, \texttt{Int}, 10),\\
&\qquad \texttt{a} \mapsto (\texttt{var}, \texttt{Array}, 0),\\
&\qquad \texttt{i} \mapsto (\texttt{var}, \texttt{Int}, 10)]\\
\text{ct}(C, \text{st}) &= \text{ct}(\texttt{i:=1}, \text{st})\\
&\ \text{ct}(\texttt{while i<=10 do a[i]:=i; i:=i+1}, \text{st})\\
\text{ct}(\texttt{i:=1}, \text{st}) &= \text{vt}(\texttt{i}, \text{st}) \quad \%\ \text{adr}(\texttt{i})\\
&\ \text{et}(\texttt{1}, \text{st}) \quad \%\ \text{val}(\texttt{1})\\
&\ \texttt{STORE;}\\
&= \texttt{LIT(10); LIT(1); STORE;}
\end{aligned}$$

Translation Example II

Example 21.6 (continued)

$$\begin{aligned}
\mathrm{ct}(\texttt{while i<=10 do a[i]:=i; i:=i+1}, \mathrm{st}) & \\
= l:\ & \mathrm{et}(\texttt{i<=10}, \mathrm{st}) \\
& \texttt{JFALSE(}l'\texttt{);} \\
& \mathrm{ct}(\texttt{a[i]:=i; i:=i+1}, \mathrm{st}) \\
& \texttt{JMP(}l\texttt{);} \\
l':\ & \\
\mathrm{et}(\texttt{i<=10}, \mathrm{st}) &= \texttt{LIT(10); LOAD; LIT(10); LE;} \\
\mathrm{ct}(\texttt{a[i]:=i; i:=i+1}, \mathrm{st}) &= \mathrm{ct}(\texttt{a[i]:=i}, \mathrm{st})\ \mathrm{ct}(\texttt{i:=i+1}, \mathrm{st}) \\
\mathrm{ct}(\texttt{a[i]:=i}, \mathrm{st}) &= \mathrm{vt}(\texttt{a[i]}, \mathrm{st}) \quad \%\ \mathrm{adr}(\texttt{a[i]}) \\
&\quad\ \mathrm{et}(\texttt{i}, \mathrm{st}) \quad \%\ \mathrm{val}(\texttt{i}) \\
&\quad\ \texttt{STORE;} \\
\mathrm{vt}(\texttt{a[i]}, \mathrm{st}) &= \mathrm{vt}(\texttt{a}, \mathrm{st}) \quad \%\ \mathrm{adr}(\texttt{a}) \\
&\quad\ \mathrm{et}(\texttt{i}, \mathrm{st}) \quad \%\ \mathrm{val}(\texttt{i}) \\
&\quad\ \texttt{CAB(1,10);} \quad \%\ \text{bounds checking} \\
&\quad\ \texttt{LIT(1); SUB;} \quad \%\ \text{index diff.} \\
&\quad\ \texttt{LIT(1); MULT;} \quad \%\ \text{rel. address} \\
&\quad\ \texttt{ADD;} \quad \%\ \mathrm{adr}(\texttt{a[i]}) \\
\mathrm{vt}(\texttt{a}, \mathrm{st}) &= \texttt{LIT(0);} \\
\mathrm{et}(\texttt{i}, \mathrm{st}) &= \texttt{LIT(10); LOAD;} \\
\mathrm{ct}(\texttt{i:=i+1}, \mathrm{st}) &= \texttt{LIT(10); LIT(10); LOAD; LIT(1); ADD; STORE;}
\end{aligned}$$