Technische Universität München *Summer Semester 2009*
Fakultät für Informatik *V. Vojdani, A. Lehmann*
Prof. Dr. H. Seidl *{vojdanig,lehmanna}@in.tum.de*

Code Generation for OCaml and the MaMa VM

General

As usual, an *address environment* is needed for the association of *local* and *global* addresses to variables. This is given by

$$\rho : Vars \rightarrow \{L, G\} \times \mathbb{Z}.$$

Besides the address environment and due to the layout of the stack, it is also neccessary to keep track of the so-called *stack distance*, which is used to keep track of the movement of the `SP` whenever instructions modify the stack.

Free Variables

A recursive definition of $\mathit{free} : \mathit{Expr} \rightarrow 2^{\mathit{Vars}}$ is achieved using the auxiliary function $f\colon \mathit{Expr} \times 2^{\mathit{Vars}} \rightarrow 2^{\mathit{Vars}}$, which is defined as follows. Then, $\mathit{free}(e) = f(e, \emptyset)$.

$$f(e,\Omega) = \begin{cases} \emptyset & e \;\equiv\; b \\ \{x\} \Leftrightarrow x \notin \Omega \, else \, \emptyset & e \;\equiv\; x \\ f(e_1,\Omega) & e \;\equiv\; (\Box_1 \, e_1) \\ f(e_1,\Omega) \cup f(e_2,\Omega) & e \;\equiv\; (e_1 \, \Box_2 \, e_2) \\ f(e_1,\Omega) \cup f(e_2,\Omega) \cup f(e_3,\Omega) & e \;\equiv\; (\textbf{if}\, e_1 \,\textbf{then}\, e_2 \,\textbf{else}\, e_3) \\ f(e',\Omega) \cup \left(\bigcup_{i=1}^{k} f(e_i,\Omega)\right) & e \;\equiv\; (e' \, e_1 \ldots e_k) \\ f(e', \Omega \cup \{x_1,\ldots,x_k\}) & e \;\equiv\; (\textbf{fun}\, x_1,\ldots,x_k \;\rightarrow\; e') \\ f(e, \Omega \cup \{x_1\}) & e \;\equiv\; (\textbf{let}\, x_1 = e_1 \,\textbf{in}\, e_k) \\ \bigcup_{i=1}^{k} \left[f(e_i, \Omega \cup \left(\bigcup_{j=1}^{k-1} \{x_j\}\right)\right] & e \;\equiv\; (\textbf{let rec}\, x_1 = e_1 \,\textbf{and} \ldots\, x_{k-1} = e_{k-1} \,\textbf{in}\, e_k) \end{cases}$$

As can be seen from the above, Ω is used to accumulate any "local" variables. Hence the set of free variables can be determined from the set of those variables in an expression that are not contained in Ω.

> However, attention must be paid to the clear distinction of the variables' identifiers. For example, consider the following expression:
>
> ```
> let x1 = 5
> in let x1 = x1
> in x1
> ```

Clearly, the outer `x1` does *not* determine the same variable as the inner `x1`, although at first sight both are denoted by identical *literals*. Thus, when adding variables to Ω during any recursive descent through $f(e, \Omega)$, some meaningful way has to be established so as to be able to clearly distinguish any variables despite of possible identical identifiers. This could, for instance, be achieved by replacing the identifiers with unique symbols before evaluating $\mathit{free}(e)$.

Basic Expressions code_B

$$
\begin{aligned}
\mathit{code}_B\ b\ \rho\ \mathit{sd} &\equiv \texttt{loadc}\ b \\
\mathit{code}_B\ x\ \rho\ \mathit{sd} &\equiv \texttt{getvar}\ x\ \rho\ \mathit{sd} \\
&\ \texttt{getbasic} \\
\mathit{code}_B\ (\Box_1\ e)\ \rho\ \mathit{sd} &\equiv \mathit{code}_B\ e\ \rho\ \mathit{sd} \\
&\ \texttt{op}_1 \\
\mathit{code}_B\ (e_1\ \Box_2\ e_2)\ \rho\ \mathit{sd} &\equiv \mathit{code}_B\ e_1\ \rho\ \mathit{sd} \\
&\ \mathit{code}_B\ e_2\ \rho\ (\mathit{sd}+1) \\
&\ \texttt{op}_2 \\
\mathit{code}_B\ (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\ \rho\ \mathit{sd} &\equiv \mathit{code}_B\ e_0\ \rho\ \mathit{sd} \\
&\ \texttt{jumpz A} \\
&\ \mathit{code}_B\ e_1\ \rho\ \mathit{sd} \\
&\ \texttt{jump B} \\
&\ \texttt{A}: \mathit{code}_B\ e_2\ \rho\ \mathit{sd} \\
&\ \texttt{B}: \ldots \\
\mathit{code}_B\ e\ \rho\ \mathit{sd} &\equiv \mathit{code}_V\ e\ \rho\ \mathit{sd} \\
&\ \texttt{getbasic}
\end{aligned}
$$

Note that `getvar` $x\ \rho\ \mathit{sd}$ is a *macro* that expands to either

$$
\begin{array}{lll}
\texttt{pushglob}\ i & \text{for } \rho\ x = (G, i) & \text{or} \\
\texttt{pushloc}\ (\mathit{sd} - i) & \text{for } \rho\ x = (L, i). &
\end{array}
$$

Basic Expressions $code_V$

$$
\begin{aligned}
code_V\ b\ \rho\ sd &\equiv \texttt{loadc}\ b \\
&\ \texttt{mkbasic} \\
code_V\ x\ \rho\ sd &\equiv \texttt{getvar}\ x\ \rho\ sd \\
code_V\ (\Box_1\ e)\ \rho\ sd &\equiv code_B\ e\ \rho\ sd \\
&\ \texttt{op}_1 \\
&\ \texttt{mkbasic} \\
code_V\ (e_1\ \Box_2\ e_2)\ \rho\ sd &\equiv code_B\ e_1\ \rho\ sd \\
&\ code_B\ e_2\ \rho\ (sd+1) \\
&\ \texttt{op}_2 \\
&\ \texttt{mkbasic} \\
code_V\ (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\ \rho\ sd &\equiv code_B\ e_0\ \rho\ sd \\
&\ \texttt{jumpz A} \\
&\ code_V\ e_1\ \rho\ sd \\
&\ \texttt{jump B} \\
&\ \texttt{A}: code_V\ e_2\ \rho\ sd \\
&\ \texttt{B}: \ldots
\end{aligned}
$$

Note that `getvar` $x\ \rho\ sd$ is a *macro* that expands to either

$$
\begin{array}{lll}
\texttt{pushglob}\ i & \text{for } \rho\ x = (G, i) & \text{or} \\
\texttt{pushloc}\ (sd - i) & \text{for } \rho\ x = (L, i). &
\end{array}
$$

`let` Expressions

With *CBN*, the expression $e \equiv \textbf{let}\ x_1 = e_1\ \textbf{in}\ e_0$ is translated as follows:

$$
\begin{aligned}
code_V\ e\ \rho\ sd &\equiv code_C\ e_1\ \rho\ sd \\
&\ code_V\ e_0\ \rho_1\ (sd+1) \\
&\ \texttt{slide 1}
\end{aligned}
$$

where $\rho_1 = \rho \oplus \{x_i \mapsto (L, sd+1)\}$.

If it were to be *CBV*, then the above $code_C$ had to be replaced with $code_V$ for e_1.

`let rec` Expressions

With *CBN*, the expression $e \equiv \textbf{let rec}\ x_1 = e_1\ \textbf{and} \ldots\ \textbf{and}\ x_n = e_n\ \textbf{in}\ e_0$ is translated as follows:

$$\begin{array}{rcl} \mathit{code}_V\ e\ \rho\ sd & \equiv & \texttt{alloc}\ n \\ & & \mathit{code}_C\ e_1\ \rho'\ (sd+n) \\ & & \texttt{rewrite}\ n \\ & & \ldots \\ & & \mathit{code}_C\ e_n\ \rho'\ (sd+n) \\ & & \texttt{rewrite}\ 1 \\ & & \mathit{code}_V\ e_0\ \rho'\ (sd+n) \\ & & \texttt{slide}\ n \end{array}$$

where $\rho' = \rho \oplus \{x_i \mapsto (L, sd+i) \mid i = 1, \ldots, n\}$.

If it were to be *CBV*, then each of the above code_C had to be replaced with code_V.

Function Definitions

$$\begin{array}{rcl} \mathit{code}_V\ (\textbf{fun}\ x_0, \ldots, x_{k-1} \to e)\ \rho\ sd & \equiv & \texttt{getvar}\ z_0\ \rho\ sd \\ & & \texttt{getvar}\ z_1\ \rho\ (sd+1) \\ & & \ldots \\ & & \texttt{getvar}\ z_{g-1}\ \rho\ (sd+g-1) \\ & & \texttt{mkvec}\ g \\ & & \texttt{mkfunval A} \\ & & \texttt{jump B} \\ & & \texttt{A}:\ \texttt{targ}\ k \\ & & \mathit{code}_V\ e\ \rho'\ 0 \\ & & \texttt{return}\ k \\ & & \texttt{B}:\ \ldots \end{array}$$

where

$$\{z_0, \ldots, z_{g-1}\} = \mathit{free}\,(\textbf{fn}\ x_0, \ldots, x_{k-1} \Rightarrow e)$$

and

$$\rho' = \{x_i \mapsto (L, -i) \mid i = 0, \ldots, k-1\} \cup \{z_j \mapsto (G, j) \mid j = 0, \ldots, g-1\}.$$

Function Applications

$$
\begin{array}{rcl}
code_V\ (e'\ e_0\ \ldots\ e_{m-1})\ \rho\ sd & \equiv & \texttt{mark A} \\
 & & code_C\ e_{m-1}\ \rho\ (sd+3) \\
 & & code_C\ e_{m-2}\ \rho\ (sd+4) \\
 & & \ldots \\
 & & code_C\ e_0\ \rho\ (sd+m+2) \\
 & & code_V\ e'\ \rho\ (sd+m+3) \\
 & & \texttt{apply} \\
 & & \texttt{A} :\ \ldots
\end{array}
$$

Note that the above code generation scheme is for *CBN*. If it were to be *CBV*, then each of the above $code_C$ had to be replaced with $code_V$.

Construction of Closures

Since closures are closely related to functions without formal parameters, the following translation scheme is quite similar to the one of function definitions:

$$
\begin{array}{rcl}
code_C\ e\ \rho\ sd & \equiv & \texttt{getvar}\ z_0\ \rho\ sd \\
 & & \texttt{getvar}\ z_1\ \rho\ (sd+1) \\
 & & \ldots \\
 & & \texttt{getvar}\ z_{g-1}\ \rho\ (sd+g-1) \\
 & & \texttt{mkvec}\ g \\
 & & \texttt{mkclos A} \\
 & & \texttt{jump B} \\
 & & \texttt{A} : code_V\ e\ \rho'\ 0 \\
 & & \texttt{update} \\
 & & \texttt{B} :\ \ldots
\end{array}
$$

where

$$\{z_0, \ldots, z_{g-1}\} = free(e)$$

and

$$\rho' = \{z_i \mapsto (G, i) \mid i = 0, \ldots, g-1\}.$$

Tuples

$$
\begin{aligned}
code_V\ (e_o, \ldots, e_{k-1})\ \rho\ sd \quad &\equiv \quad code_C\ e_0\ \rho\ sd \\
& \qquad code_C\ e_1\ \rho\ (sd+1) \\
& \qquad \ldots \\
& \qquad code_C\ e_{k-1}\ \rho\ (sd+k-1) \\
& \qquad \texttt{mkvec}\ k \\
code_V\ (\#j\ e)\ \rho\ sd \quad &\equiv \quad code_V\ e\ \rho\ sd \\
& \qquad \texttt{get}\ j \\
code_V\ \ (\mathbf{let}\ (y_0, \ldots, y_{k-1})\ =\ e_1\ \mathbf{in}\ e_0)\ \ \rho\ sd \quad &\equiv \quad code_V\ e_1\ \rho\ sd \\
& \qquad \texttt{getvec}\ k \\
& \qquad code_V\ e_0\ \rho'\ (sd+k) \\
& \qquad \texttt{slide}\ k
\end{aligned}
$$

Lists

$$
\begin{aligned}
code_V\ []\ \rho\ sd \quad &\equiv \quad \texttt{nil} \\
code_V\ (e_1\ :\ e_2)\ \rho\ sd \quad &\equiv \quad code_C\ e_1\ \rho\ sd \\
& \qquad code_C\ e_2\ \rho\ (sd+1) \\
& \qquad \texttt{cons}
\end{aligned}
$$

Pattern Matching

The expression $e\ \equiv\ \mathbf{match}\ e_o\ \mathbf{with}\ []\ \rightarrow\ e_1\ \mid\ h : t\ \rightarrow\ e_2$ is translated as follows:

$$
\begin{aligned}
code_V\ e\ \rho\ sd \quad &\equiv \quad code_V\ e_0\ \rho\ sd \\
& \qquad \texttt{tlist A} \\
& \qquad code_V\ e_1\ \rho\ sd \\
& \qquad \texttt{jump B} \\
& \qquad \texttt{A} : code_V\ e_2\ \rho'\ (sd+2) \\
& \qquad \texttt{slide 2} \\
& \qquad \texttt{B} :\ \ldots
\end{aligned}
$$

where $\rho' = \rho \oplus \{h \mapsto (L, sd+1), t \mapsto (L, sd+2)\}$.