10 Translation of Whole Programs

Before program execution, we have:

\[ SP = -1 \quad FP = EP = -1 \quad PC = 0 \quad NP = MAX \]

Let \( p = V_{def1} \ldots V_{defr} \) denote a program where \( V_{defi} \) is the definition of a function \( f_i \) of which one is called \( \text{main} \).

The code for the program \( p \) consists of:
- code for the function definitions \( V_{defi} \);
- code for the allocation of global variables;
- code for the call of \( \text{int main}() \);
- the instruction \( \text{halt} \) which returns control to the operating system together with the value at address 0.

where \( \rho_{\text{fac}} : x \mapsto (L_r - 3) \) and \( q = 5 \).

Then we define:

\[
\text{code } p \not\emptyset = \begin{cases} \text{enter } (k+4) \\ \text{alloc } (k+1) \\ \text{mark} \\ \text{load } \text{main} \\ \text{call} \\ \text{slide } k \\ \text{halt} \end{cases}
\]

\[
\text{f}_i: \text{code } V_{\text{defi}} \rho \\
\vdots \\
\text{f}_n: \text{code } V_{\text{defn}} \rho
\]

where \( \not\emptyset \equiv \text{empty address environment} \);
\( \rho \equiv \text{global address environment} \);
\( k \equiv \text{size of the global variables} \)
11 The language PuF

We only regard a mini-language PuF ("Pure Functions").

We do not treat, as yet:

- Side effects;
- Data structures;
- Exceptions.

The Translation of Functional Programming Languages

A program is an expression $e$ of the form:

\[
e ::= \, b \mid x \mid (\exists e) \mid (e_1 \oplus e_2) \mid (\text{let } x_1 = e_1 \text{ in } e_2) \mid (\text{rec } x_1 = e_1 \text{ and } \ldots \text{ and } x_n = e_n \text{ in } e_0)
\]

An expression is therefore

- a basic value, a variable, the application of an operator, or
- a function-application, a function-abstraction, or
- a let-expression, i.e. an expression with locally defined variables, or
- a let-rec-expression, i.e. an expression with simultaneously defined local variables.

For simplicity, we only allow $\text{int}$ as basic type.
Example

The following well-known function computes the factorial of a natural number:

\[
\text{let rec fac } = \begin{cases} 
\text{fun } x \rightarrow \text{if } x \leq 1 \text{ then } 1 \\
\text{else } x \cdot \text{fac } (x - 1) \end{cases} 
\]

\[\text{in fac 7}\]

As usual, we only use the minimal amount of parentheses.

There are two Semantics:

**CBV:** Arguments are evaluated before they are passed to the function (as in SML);

**CBN:** Arguments are passed unevaluated; they are only evaluated when their value is needed (as in Haskell).

12 Architecture of the MaMa:

We know already the following components:

\[
\begin{align*}
C &= \text{Code-store – contains the MaMa-program; each cell contains one instruction;} \\
PC &= \text{Program Counter – points to the instruction to be executed next;}
\end{align*}
\]

We also need a heap \(H\):

\[
\begin{align*}
S &= \text{Runtime-Stack – each cell can hold a basic value or an address;}
SP &= \text{Stack-Pointer – points to the topmost occupied cell;}
\text{as in the CMs implicitly represented;}
FP &= \text{Frame-Pointer – points to the actual stack frame.}
\end{align*}
\]
... it can be thought of as an abstract data type, being capable of holding data objects of the following form:

\[
\begin{array}{c}
\text{v} \\
B ~ -173 \\
\text{cp} \qquad \text{gp} \\
C \\
\text{cp} \qquad \text{gp} \\
F \\
v[0] \quad \ldots \quad v[n-1] \\
V \quad n \\
\end{array}
\]

Basic Value
Closure
Function
Vector

The instruction \textit{new (tag, args)} creates a corresponding object (B, C, F, V) in H and returns a reference to it.

We distinguish three different kinds of code for an expression \( e \):

- \texttt{codeV} \( e \) — (generates code that) computes the Value of \( e \), stores it in the heap and returns a reference to it on top of the stack (the normal case);
- \texttt{codeE} \( e \) — computes the value of \( e \), and returns it on the top of the stack (only for Basic types);
- \texttt{codeC} \( e \) — does \textit{not} evaluate \( e \), but stores a Closure of \( e \) in the heap and returns a reference to the closure on top of the stack.

We start with the code schemata for the first two kinds:
13 Simple expressions

Expressions consisting only of constants, operator applications, and conditionals are translated like expressions in imperative languages:

\[
\begin{align*}
\text{code}_\mathcal{E} b \sigma d & \quad = \quad \text{loadc } b \\
\text{code}_\mathcal{E} (e_1 \cdot e_2) \sigma d & \quad = \quad \text{code}_\mathcal{E} e_1 \sigma d \quad \text{op}_1 \\
\text{code}_\mathcal{E} (e_1 \downarrow e_2) \sigma d & \quad = \quad \text{code}_\mathcal{E} e_1 \sigma d \quad \text{code}_\mathcal{E} e_2 \sigma d + 1 \quad \text{op}_2
\end{align*}
\]