25 Last Calls

A function application is called last call in an expression $e$ if this application could deliver the value for $e$.

A function definition is called tail recursive if all recursive calls are last calls.

Examples:

$r t (h :: y)$ is a last call in $\text{match } x \text{ with } [] \rightarrow y \mid h :: t \rightarrow r t (h :: y)$

$f (x - 1)$ is not a last call in $\text{if } x \leq 1 \text{ then } 1 \text{ else } x * f (x - 1)$

Observation: Last calls in a function body need no new stack frame!

$\Rightarrow$

Automatic transformation of tail recursion into loops!!!
The code for a last call \( l \equiv (e' e_0 \ldots e_m) \) inside a function \( f \) with \( k \) arguments must

1. allocate the arguments \( e_i \) and evaluate \( e' \) to a function (note: all this inside \( f \)'s frame!);
2. deallocate the local variables and the \( k \) consumed arguments of \( f \);
3. execute an apply.

\[
\text{code}_V l \, \rho \, \text{sd} = \text{code}_C e_{m-1} \, \rho \, \text{sd} \\
\text{code}_C e_{m-2} \, \rho \, (\text{sd} + 1) \\
\ldots \\
\text{code}_C e_0 \, \rho \, (\text{sd} + m - 1) \\
\text{code}_V e' \, \rho \, (\text{sd} + m) \quad // \text{Evaluation of the function} \\
\text{move} \, r \, (m + 1) \quad // \text{Deallocation of } r \text{ cells} \\
\text{apply}
\]

where \( r = sd + k \) is the number of stack cells to deallocate.
Example:

V-code for the body of the function

\[
\begin{align*}
r = \text{fun } x \ y & \rightarrow \text{match } x \text{ with } [] \rightarrow y \ | \ h :: t \rightarrow r \ t \ (h :: y)
\end{align*}
\]

with CBN semantics:

\[
\begin{array}{cccccc}
0 & \text{targ 2} & 1 & \text{jump B} & 4 & \text{pushglob 0} \\
0 & \text{pushloc 0} & 1 & \text{eval} & 5 & \text{eval} \\
1 & \text{eval} & 2 & A: \text{pushloc 1} & 5 & \text{move 4 3} \\
1 & \text{tlist A} & 3 & \text{pushloc 4} & 5 & \text{apply} \\
0 & \text{pushloc 1} & 4 & \text{cons} & 5 & \text{slide 2} \\
1 & \text{eval} & 3 & \text{pushloc 1} & 1 & B: \text{return 2}
\end{array}
\]

Since the old stack frame is kept, \text{return 2} will only be reached by the direct jump at the end of the []-alternative.
SP = SP – k – r;
for (i=1; i≤k; i++)
    S[SP+i] = S[SP+i+r];
SP = SP + k;
The Translation of Logic Languages
26 The Language Proll

Here, we just consider the core language Proll (“Prolog-light” :). In particular, we omit:

- arithmetic;
- the cut operator;
- self-modification of programs through assert and retract.
Example:

\[ \text{bigger}(X, Y) \leftrightarrow X = \text{elephant}, Y = \text{horse} \]
\[ \text{bigger}(X, Y) \leftrightarrow X = \text{horse}, Y = \text{donkey} \]
\[ \text{bigger}(X, Y) \leftrightarrow X = \text{donkey}, Y = \text{dog} \]
\[ \text{bigger}(X, Y) \leftrightarrow X = \text{donkey}, Y = \text{monkey} \]
\[ \text{is}_\text{bigger}(X, Y) \leftrightarrow \text{bigger}(X, Y) \]
\[ \text{is}_\text{bigger}(X, Y) \leftrightarrow \text{bigger}(X, Z), \text{is}_\text{bigger}(Z, Y) \]
\[ ? \text{ is}_\text{bigger}(\text{elephant}, \text{dog}) \]
A More Realistic Example:

\[
\begin{align*}
\text{app}(X, Y, Z) & \leftarrow X = [ ], Y = Z \\
\text{app}(X, Y, Z) & \leftarrow X = [H|X'], Z = [H|Z'], \text{app}(X', Y, Z') \\
? & \text{app}(X, [Y, c], [a, b, Z])
\end{align*}
\]
A More Realistic Example:

\[
\begin{align*}
\text{app}(X, Y, Z) & \leftarrow X = [], Y = Z \\
\text{app}(X, Y, Z) & \leftarrow X = [H|X'], Z = [H|Z'], \text{app}(X', Y, Z') \\
? & \text{app}(X, [Y, c], [a, b, Z])
\end{align*}
\]

Remark:

\[
\begin{align*}
[ ] & \quad \text{the atom empty list} \\
[H|Z] & \quad \text{binary constructor application} \\
[a, b, Z] & \quad \text{shortcut for: } \ [a|[b|[Z|[ ]]]]
\end{align*}
\]
A program $p$ is constructed as follows:

$$
t ::= a | X | _ | f(t_1, \ldots, t_n)
g ::= p(t_1, \ldots, t_k) | X = t
c ::= p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_r
p ::= c_1 \ldots c_m?g$$

- A **term** $t$ either is an atom, a variable, an anonymous variable or a constructor application.
- A **goal** $g$ either is a literal, i.e., a predicate call, or a unification.
- A **clause** $c$ consists of a **head** $p(X_1, \ldots, X_k)$ with predicate name and list of formal parameters together with a **body**, i.e., a sequence of goals.
- A **program** consists of a sequence of clauses together with a single goal as query.
Procedural View of Proll programs:

- literal \(\longrightarrow\) procedure call
- predicate \(\longrightarrow\) procedure
- clause \(\longrightarrow\) definition
- term \(\longrightarrow\) value
- unification \(\longrightarrow\) basic computation step
- binding of variables \(\longrightarrow\) side effect

Note: Predicate calls ...

- ... do not have a return value.
- ... affect the caller through side effects only \(\text{:)}\)
- ... may fail. Then the next definition is tried \(\text{:-))}\)
  \[
  \implies \quad \text{backtracking}
  \]
27 Architecture of the WiM:

The Code Store:

C = Code store – contains WiM program;
   every cell contains one instruction;

PC = Program Counter – points to the next instruction to executed;
The Runtime Stack:

\[ S = \text{Runtime Stack} - \text{every cell may contain a value or an address}; \]
\[ \text{SP} = \text{Stack Pointer} - \text{points to the topmost occupied cell}; \]
\[ \text{FP} = \text{Frame Pointer} - \text{points to the current stack frame.} \]

Frames are created for predicate calls, contain cells for each variable of the current clause.
The Heap:

- $H$ = Heap for dynamically constructed terms;
- $HP$ = Heap-Pointer – points to the first free cell;

- The heap is maintained like a stack as well.
- A new-instruction allocates an object in $H$.
- Objects are tagged with their types (as in the MaMa). ...
atom
1 cell
variable
1 cell
unbound variable
1 cell
structure
(n+1) cells

226
28 Construction of Terms in the Heap

Parameter terms of goals (calls) are constructed in the heap before passing.

Assume that the address environment $\rho$ returns, for each clause variable $X$ its address (relative to FP) on the stack. Then $\text{code}_A t \rho$ should ...

- construct (a presentation of) $t$ in the heap; and
- return a reference to it on top of the stack.

Idea:

- Construct the tree during a post-order traversal of $t$
- with one instruction for each new node!

Example: $t \equiv f(g(X,Y),a,Z)$.

Assume that $X$ is initialized, i.e., $S[\text{FP} + \rho X]$ contains already a reference, $Y$ and $Z$ are not yet initialized.