Script generated by TTT

Title: Petter: Virtual Machines (27.05.2019)

Date: Mon May 27 10:19:28 CEST 2019

Duration: 85:52 min

Pages: 8

25 Last Calls

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

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

Examples

$r\ t\ (h::y)$ is a last call in

$$\begin{array}{l} \textbf{match } x \textbf{ with} \\ [\,] \ \to y \\ \mid h::t \ \to r\ t\ (h::y) \end{array}$$

$f\ (x-1)$ is not a last call in

$$\boxed{\begin{array}{l} \textbf{if } x \leq 1 \\ \textbf{then } \boxed{1} \\ \textbf{else } \boxed{x * f\ (x-1)} \end{array}}$$

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

$\Longrightarrow$ Automatic transformation of tail recursion into loops!!!

The code for a last call $l \equiv (e'\ e_0 \ldots e_{m-1})$ 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. de-allocate the local variables and the k consumed arguments of f;
3. execute an apply.

$$\begin{array}{lll} \text{code}_V\ l\ \rho\ \text{sd} & = & \text{code}_C\ e_{m-1}\ \rho\ \text{sd} \\ & & \text{code}_C\ e_{m-2}\ \rho\ (\text{sd}+1) \\ & & \ldots \\ & & \text{code}_C\ e_0\ \rho\ (\text{sd}+m-1) \\ & & \text{code}_V\ e'\ \rho\ (\text{sd}+m) \quad // \text{ evaluation of the function} \\ & & \text{move } r\ (m+1) \quad // \text{ de-allocation of } r \text{ cells} \\ & & \text{apply} \end{array}$$

where $r = \text{sd}+k$ is the number of stack cells to de-allocate.

A More Realistic Example

$[\,]$

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

$X = [\alpha]$

$X = [a]$

$[Y, c] = [b, Z]$

$[b, z] = [b, z]$

$Z = c$

$Y = b$

$H = a$

Procedural View of Proll programs

literal	$\equiv$	$\boxed{\text{procedure call}}$
predicate	$\equiv$	$\boxed{\text{procedure}}$
clause	$\equiv$	$\boxed{\text{definition}}$
term	$\equiv$	$\boxed{\text{value}}$
unification	$\equiv$	$\boxed{\text{basic computation step}}$
binding of variables	$\equiv$	side effect

Note: Predicate calls ...

- ... do not have a return value.
- ... affect the caller through side effects only.
- ... may fail. Then the next definition is tried.

$\Longrightarrow$ backtracking

A program p is constructed as follows:

$$\begin{array}{lcl} t & ::= & a \mid X \mid \_ \mid f(t_1, \ldots, t_n) \\ g & ::= & p(t_1, \ldots, t_k) \mid X = t \\ c & ::= & p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_r \\ p & ::= & c_1 . \ldots . c_m ? g \end{array}$$

- 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	$\equiv$	procedure call
predicate	$\equiv$	procedure
clause	$\equiv$	definition
term	$\equiv$	value
unification	$\equiv$	basic computation step
binding of variables	$\equiv$	side effect

Note: Predicate calls ...

- ... do not have a return value.
- ... affect the caller through side effects only.
- ... may fail. Then the next definition is tried.

$\Longrightarrow$ backtracking

A program p is constructed as follows:

$$\begin{array}{lcl} t & ::= & a \mid X \mid \_ \mid f(t_1, \ldots, t_n) \\ g & ::= & p(t_1, \ldots, t_k) \mid X = t \\ c & ::= & p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_r \\ p & ::= & c_1 . \ldots . c_m ? g \end{array}$$

- 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	$==$	procedure call
predicate	$==$	procedure
clause	$==$	definition
term	$==$	value
unification	$==$	basic computation step
binding of variables	$==$	side effect

Note: Predicate calls ...

- ... do not have a return value.
- ... affect the caller through side effects only.
- ... may fail. Then the next definition is tried.

$\Longrightarrow$ backtracking