Script **generated by TTT**

Title: Seidl: Virtual_Machines (23.05.2016)

Date: Mon May 23 10:27:05 CEST 2016

Duration: 80:22 min

Pages: 34

24.5 Closures of Tuples and Lists

The general schema for code_C can be optimized for tuples and lists:

$$
\begin{array}{lclcl}
\text{code}_C\ (e_0,\ldots,e_{k-1})\ \rho\ \text{sd} & = & \text{code}_V\ (e_0,\ldots,e_{k-1})\ \rho\ \text{sd} & = & \text{code}_C\ e_0\ \rho\ \text{sd} \\
 & & & & \text{code}_C\ e_1\ \rho\ (\text{sd}+1) \\
 & & & & \ldots \\
 & & & & \text{code}_C\ e_{k-1}\ \rho\ (\text{sd}+k-1) \\
 & & & & \text{mkvec k} \\
\text{code}_C\ []\ \rho\ \text{sd} & = & \text{code}_V\ []\ \rho\ \text{sd} & = & \text{nil} \\
\text{code}_C\ (e_1::e_2)\ \rho\ \text{sd} & = & \text{code}_V\ (e_1::e_2)\ \rho\ \text{sd} & = & \text{code}_C\ e_1\ \rho\ \text{sd} \\
 & & & & \text{code}_C\ e_2\ \rho\ (\text{sd}+1) \\
 & & & & \text{cons}
\end{array}
$$

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 **match** x **with** $[] \to y \mid h::t \to r\ t\ (h::y)$

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

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. deallocate the local variables and the k consumed arguments of f;
3. execute an apply.

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

where $r = sd + k$ is the number of stack cells to deallocate.

Example

$x \mapsto (L,0)$ $\quad$ $h \mapsto (L,1)$ $\quad$ $r \mapsto (G,0)$
$y \mapsto (L,-1)$ $\quad$ $t \mapsto (L,2)$

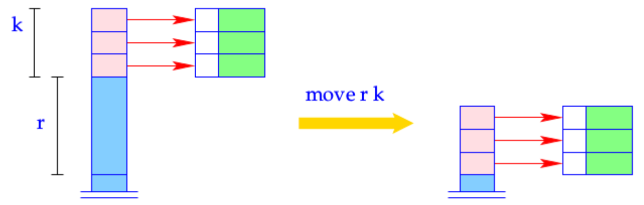
V-code for the body of the function

$$r = \textbf{fun}\, x\ y\ \rightarrow\ \textbf{match}\ x\ \textbf{with}\ [\,]\ \rightarrow y\ |\ h :: t\ \rightarrow r\ t\ (h :: y)$$

with CBN semantics:

0		targ 2	1		jump B	4		pushglob 0
0		pushloc 0				5		eval
1		eval	2	A:	pushloc 1	5		move 4 3
1		tlist A	3		pushloc 4			apply
0		pushloc 1	4		cons			~~slide 2~~
1		eval	3		pushloc 1	1	B:	return 2

Since the old stack frame is kept, 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;

26 Exceptions

Example

$$\begin{array}{l}
\textbf{let rec}\ gcd = \textbf{fun}\ x\ y \rightarrow \\
\quad \textbf{if}\ x \leq 0\ ||\ y \leq 0\ \textbf{then raise}\ [\,] \\
\quad \textbf{else if}\ x = y\ \textbf{then}\ x \\
\quad \textbf{else if}\ y < x\ \textbf{then}\ gcd\ (x - y)\ y \\
\quad \textbf{else}\ gcd\ x\ (y - x) \\
\textbf{in try}\ gcd\ 0\ 5 \\
\textbf{with}\ z \rightarrow 42
\end{array}$$

For simplicity, we assume that raised exception values can be of any type.

For every try expression, we maintain:

- An exception frame on the stack, which contains all relevant information to handle the exception;
- The exception pointer XP, which points the the current exception frame.

Each exception frame must record

- the negative continuation address, i.e., the address of the code for the handler;
- the global pointer and
- the frame pointer; as well as
- the old exception pointer.

For an expression of the following form:

$$e \equiv \textbf{try } e_1 \textbf{ with } x \to e_2$$

we generate:

$$\begin{array}{rcl} \mathrm{code}_V\; e\; \rho\; \mathrm{sd} & = & \mathrm{try\ A} \\ & & \mathrm{code}_V\; e_1\; \rho\; (\mathrm{sd}+4) \\ & & \mathrm{restore\ B} \\ & A: & \mathrm{code}_V\; e_2\; \rho'\; (\mathrm{sd}+1) \\ & & \mathrm{slide\ 1} \\ & B: & \ldots \end{array}$$

where $\rho' = \rho \oplus \{x \mapsto (L, \mathrm{sd}+1)\}$.

try

try ... with

with ..

26 Exceptions

Example

$$\begin{array}{l} \textbf{let rec } gcd = \textbf{fun } x\ y \to \\ \quad \textbf{if } x \leq 0 \mid\mid y \leq 0 \textbf{ then raise } 0 \\ \quad \textbf{else if } x = y \textbf{ then } x \\ \quad \textbf{else if } y < x \textbf{ then } gcd\ (x-y)\ y \\ \quad \textbf{else } gcd\ x\ (y-x) \\ \textbf{in try } gcd\ 0\ 5 \\ \textbf{with } z \to z \end{array}$$

For simplicity, we assume that raised exception values can be of any type.

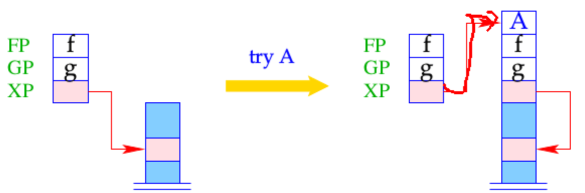
For an expression of the following form:

$$e \equiv \textbf{try } e_1 \textbf{ with } x \to e_2$$

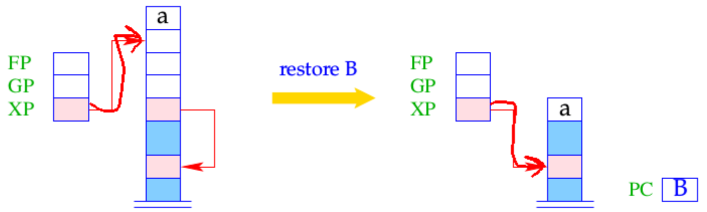
we generate:

$$\begin{array}{rcl} \mathrm{code}_V\; e\; \rho\; \mathrm{sd} & = & \mathrm{try\ A} \\ & & \mathrm{code}_V\; e_1\; \rho\; (\mathrm{sd}+4) \\ & & \mathrm{restore\ B} \\ & A: & \mathrm{code}_V\; e_2\; \rho'\; (\mathrm{sd}+1) \\ & & \mathrm{slide\ 1} \\ & B: & \ldots \end{array}$$

where $\rho' = \rho \oplus \{x \mapsto (L, \mathrm{sd}+1)\}$.



S[SP+1] = XP;
S[SP+2] = GP;
S[SP+3] = FP;
S[SP+4] = A;
XP = SP = SP+4;



PC = B;
XP = S[SP-4];
S[SP-4] = S[SP];
SP = SP-4;

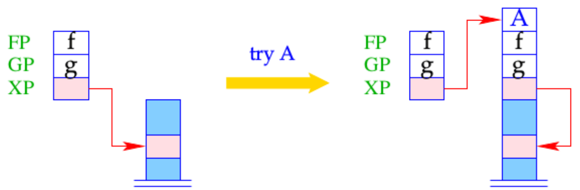
Now we have all provisions to raise exceptions.

For these, we do:

- We give up the current computational context;
- We restore the context of the closest surrounding try expression;
- We hand over the exception value to the exception handler.

Thus, we translate:

$$\begin{array}{lll} \text{code}_V\ (\textbf{raise}\ e)\ \rho\ \text{sd} & = & \text{code}_V\ e\ \rho\ \text{sd} \\ & & \text{raise} \end{array}$$



S[SP+1] = XP;
S[SP+2] = GP;
S[SP+3] = FP;
S[SP+4] = A;
XP = SP = SP+4;

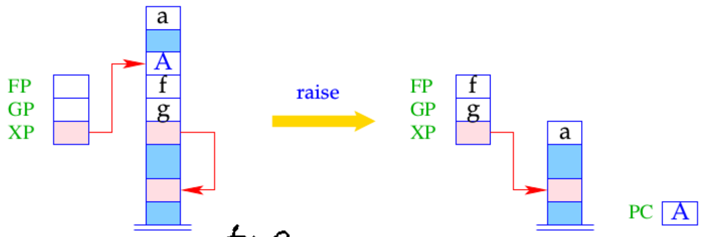
For an expression of the following form:

$$e \equiv \textbf{try}\ e_1\ \textbf{with}\ x \to e_2$$

we generate:

$$\begin{array}{rlll} & \text{code}_V\ e\ \rho\ \text{sd} & = & \text{try A} \\ & & & \text{code}_V\ e_1\ \rho\ (\text{sd}+4) \\ & & & \text{restore B} \\ A: & & & \text{code}_V\ e_2\ \rho'\ (\text{sd}+1) \\ & & & \text{slide 1} \\ B: & & & \dots \end{array}$$

where $\rho' = \rho \oplus \{x \mapsto (L, \text{sd}+1)\}$.



tmp = S[SP];
SP = XP-3;
PC = S[XP];
FP = S[XP-1];
GP = S[XP-2];
XP = S[XP-3];
S[SP] = tmp;

26 Exceptions

Example

let rec gcd = **fun** $x\ y \rightarrow$
  if $x \leq 0 \mid\mid y \leq 0$ **then raise** 0
  else if $x = y$ **then** x
  else if $y < x$ **then** $gcd\ (x - y)\ y$
  else $gcd\ x\ (y - x)$
in try gcd 0 5
with $z \rightarrow z$

For simplicity, we assume that raised exception values can be of any type.

Example

The V-code for *gcd* is given by:

0		alloc 1	2	B:	rewrite 1	10		mkbasic
1		pushloc 0	1		try C	10		pushloc 9
2		mkvec 1	5		mark D	11		apply
2		mkfun A	8		loadc 5	6	D	restore E
2		jump B	9		mkbasic	2	C:	pushloc 0
0	A:	targ 2	9		loadc 0	3		slide 1
		...				2	E:	slide 1
		return 2						

Example

The V-code for *gcd* is given by:

0		alloc 1	2	B:	rewrite 1	10		mkbasic
1		pushloc 0	1		try C	10		pushloc 9
2		mkvec 1	5		mark D	11		apply
2		mkfun A	8		loadc 5	6	D:	restore E
2		jump B	9		mkbasic	2	C:	pushloc 0
0	A:	targ 2	9		loadc 0	3		slide 1
		...				2	E:	slide 1
		return 2						

For an expression of the following form:

$$e \equiv \textbf{try}\ e_1\ \textbf{with}\ x \to e_2$$

we generate:

$$\begin{array}{rcl}
\text{code}_V\ e\ \rho\ \text{sd} & = & \text{try A} \\
& & \text{code}_V\ e_1\ \rho\ (\text{sd}+4) \\
& & \text{restore B} \\
& A: & \text{code}_V\ e_2\ \rho'\ (\text{sd}+1) \\
& & \text{slide } 1 \\
& B: & \dots
\end{array}$$

where $\rho' = \rho \oplus \{x \mapsto (L, \text{sd}+1)\}$.

Remarks

- In Ocaml, exceptions may also be raised by the runtime system.
- Therefore, exceptions form a datatype on their own, which can be extended with further constructors by the programmer.
- The handler performs pattern matching on the exception value.
- If the given exception value is not matched, the exception value is raised again.

match e with) try e with
| 0 -> ...
| 1 -> ...
| _ -> ...

x -> match x with
[] -> ..
x::xs -> ..

Remarks

- In Ocaml, exceptions may also be raised by the runtime system.
- Therefore, exceptions form a datatype on their own, which can be extended with further constructors by the programmer.
- The handler performs pattern matching on the exception value.
- If the given exception value is not matched, the exception value is raised again.

Caveat

Exceptions only make sense in CBV languages !!

Why??

The Translation of Logic Languages

27 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.

... in Concrete Syntax:

$$
\begin{array}{lll}
\text{bigger}(\mathit{elephant}, \mathit{horse}). & & \\
\text{bigger}(\mathit{horse}, \mathit{donkey}). & & \\
\text{bigger}(\mathit{donkey}, \mathit{dog}). & & \\
\text{bigger}(\mathit{donkey}, \mathit{monkey}). & & \\
\text{is\_bigger}(X, Y) & :- & \text{bigger}(X, Y). \\
\text{is\_bigger}(X, Y) & :- & \text{bigger}(X, Z), \text{is\_bigger}(Z, Y). \\
?- \;\; \text{is\_bigger}(\mathit{elephant}, \mathit{dog}). & &
\end{array}
$$

... in Concrete Syntax:

$$
\begin{array}{lll}
\text{bigger}(\mathit{elephant}, \mathit{horse}). & & \\
\text{bigger}(\mathit{horse}, \mathit{donkey}). & & \\
\text{bigger}(\mathit{donkey}, \mathit{dog}). & & \\
\text{bigger}(\mathit{donkey}, \mathit{monkey}). & & \\
\text{is\_bigger}(X, Y) & \Leftarrow & \text{bigger}(X, Y). \\
\text{is\_bigger}(X, Y) & :- & \text{bigger}(X, Z), \text{is\_bigger}(Z, Y). \\
?- \;\; \text{is\_bigger}(\mathit{elephant}, \mathit{dog}). & &
\end{array}
$$

Example

$$
\begin{array}{lll}
\text{bigger}(X, Y) & \leftarrow & X = \mathit{elephant}, Y = \mathit{horse}. \\
\text{bigger}(X, Y) & \leftarrow & X = \mathit{horse}, Y = \mathit{donkey}. \\
\text{bigger}(X, Y) & \leftarrow & X = \mathit{donkey}, Y = \mathit{dog}. \\
\text{bigger}(X, Y) & \leftarrow & X = \mathit{donkey}, Y = \mathit{monkey}. \\
\text{is\_bigger}(X, Y) & \leftarrow & \text{bigger}(X, Y) \\
\text{is\_bigger}(X, Y) & \leftarrow & \text{bigger}(X, Z), \text{is\_bigger}(Z, Y) \\
? \;\; \text{is\_bigger}(\mathit{elephant}, \mathit{dog}) & &
\end{array}
$$

… in Concrete Syntax:

bigger(*elephant*, *horse*).
bigger(*horse*, *donkey*).
bigger(*donkey*, *dog*).
bigger(*donkey*, *monkey*).
is_bigger(X, Y) :− bigger(X, Y).
is_bigger(X, Y) :− bigger(X, Z), is_bigger(Z, Y).
?− is_bigger(*elephant*, *dog*).

Example

$$\begin{array}{lll}
\text{bigger}(X,Y) & \leftarrow & X = elephant, Y = horse \\
\text{bigger}(X,Y) & \leftarrow & X = horse, Y = donkey \\
\text{bigger}(X,Y) & \leftarrow & X = donkey, Y = dog \\
\text{bigger}(X,Y) & \leftarrow & X = donkey, Y = monkey \\
\text{is_bigger}(X,Y) & \leftarrow & \text{bigger}(X,Y) \\
\text{is_bigger}(X,Y) & \leftarrow & \text{bigger}(X,Z), \text{is_bigger}(Z,Y) \\
? & \text{is_bigger}(elephant, dog) &
\end{array}$$

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 | XS] ≃ X :: XS

A More Realistic Example

app([ ], Z, Z).
app([$H|X'$], Y, [$H|Z'$]) :− app(X', Y, Z').
?− app(X, [Y, c], [a, b, Z]).

A More Realistic Example

$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [\,],\ Y = Z$$
$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X',Y,Z')$$
$$?\ \ \mathsf{app}(X,[Y,c],[a,b,Z])$$

$$X \; @ \; [Y, c] \overset{?}{\doteq} [a, b, Z]$$

$$X \mapsto [a]$$
$$Y \mapsto b$$
$$Z \mapsto c$$

A More Realistic Example

$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [\,],\ Y = Z$$
$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X',Y,Z')$$
$$?\ \ \mathsf{app}(X,[Y,c],[a,b,Z])$$

Remark

$[\,]$	$\equiv$	the atom empty list
$[H\|Z]$	$\equiv$	binary constructor application
$[a,b,Z]$	$\equiv$	shortcut for: $[a\|[b\|[Z\|[\,]]]]$

$$[a \,|\, [b \,|\, Z]] \overset{?}{\doteq} [a, b, Z]$$

A More Realistic Example

$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [\,],\ Y = Z$$
$$\mathsf{app}(X,Y,Z) \;\leftarrow\; X = [H|X'],\ Z = [H|Z'],\ \mathsf{app}(X',Y,Z')$$
$$?\ \ \mathsf{app}(X,[Y,c],[a,b,Z])$$

Remark

$[\,]$	$\equiv$	the atom empty list
$[H\|Z]$	$\equiv$	binary constructor application
$[a,b,Z]$	$\equiv$	shortcut for: $[a\|[b\|[Z\|[\,]]]]$