With CBN, we generate for the access to a variable:

$$\begin{array}{lll} \text{code}_V\ x\ \rho\ \text{sd} & = & \text{getvar}\ x\ \rho\ \text{sd} \\ & & \text{eval} \end{array}$$

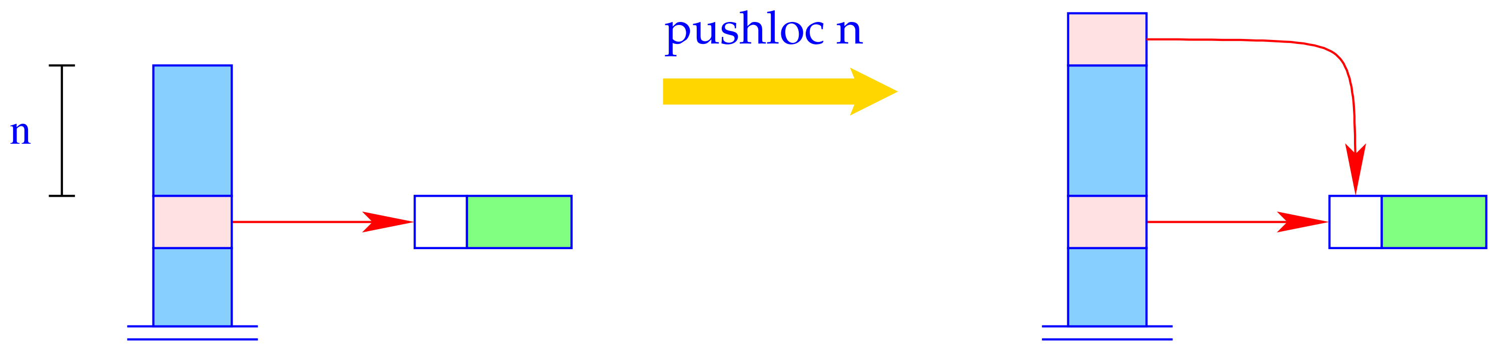
The instruction eval checks, whether the value has already been computed or whether its evaluation has to yet to be done ($\Longrightarrow$ will be treated later :-)

With CBV, we can just delete eval from the above code schema.

The (compile-time) macro getvar is defined by:

$$\begin{array}{lll} \text{getvar}\ x\ \rho\ \text{sd} & = & \textbf{let}\ (t, i) = \rho\ x\ \textbf{in} \\ & & \textbf{case}\ t\ \textbf{of} \\ & & \quad L \Rightarrow \text{pushloc}\ (\text{sd} - i) \\ & & \quad G \Rightarrow \text{pushglob i} \\ & & \textbf{end} \end{array}$$

The access to local variables:



S[SP+1] =S[SP - n]; SP++;

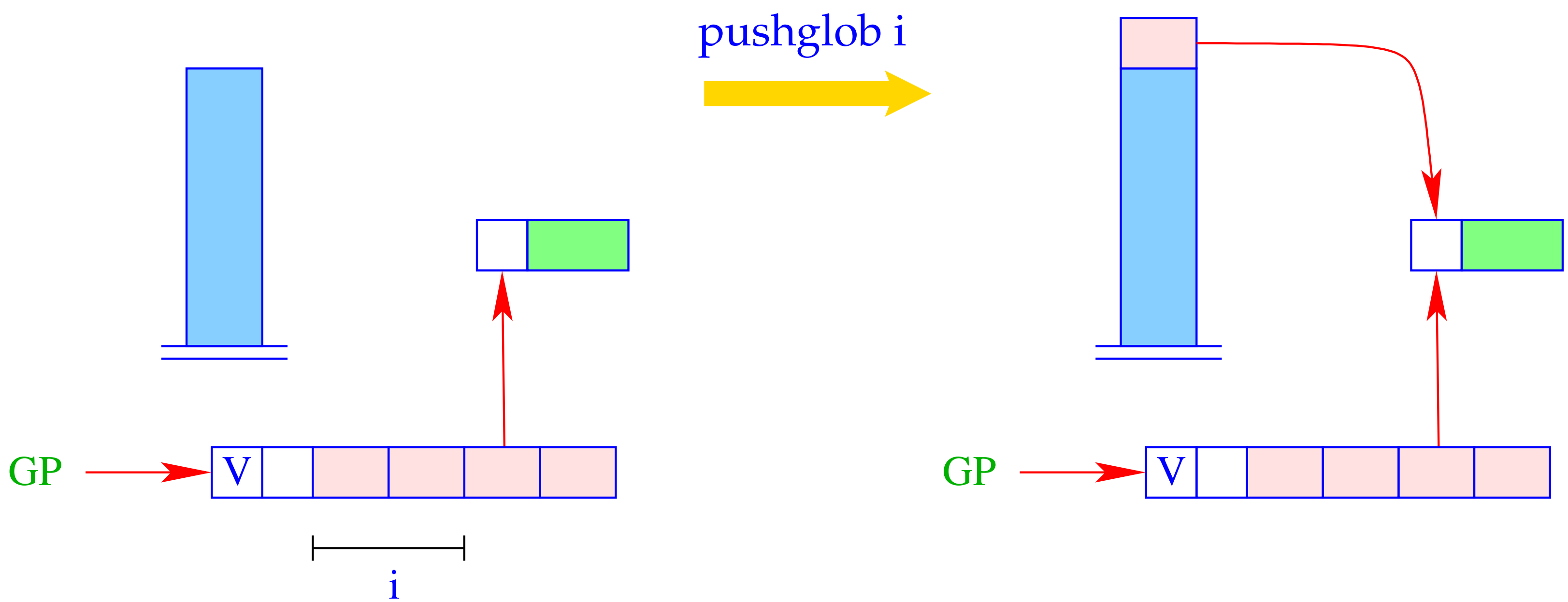
Correctness argument:

Let sp and sd be the values of the stack pointer resp. stack distance before the execution of the instruction. The value of the local variable with address i is loaded from $S[a]$ with

$$a = \text{sp} - (\text{sd} - i) = (\text{sp} - \text{sd}) + i = \text{sp}_0 + i$$

... exactly as it should be :-)

The access to global variables is much simpler:



SP = SP + 1;
S[SP] = GP→v[i];

Example:

Regard $e \equiv (b + c)$ for $\rho = \{b \mapsto (L, 1), c \mapsto (G, 0)\}$ and $\mathrm{sd} = 1$.

With CBN, we obtain:

$$
\begin{array}{llllll}
\mathrm{code}_V\ e\ \rho\ 1 & = & \mathrm{getvar}\ b\ \rho\ 1 & = & 1 & \mathrm{pushloc}\ 0 \\
 & & \mathrm{eval} & & 2 & \mathrm{eval} \\
 & & \mathrm{getbasic} & & 2 & \mathrm{getbasic} \\
 & & \mathrm{getvar}\ c\ \rho\ 2 & & 2 & \mathrm{pushglob}\ 0 \\
 & & \mathrm{eval} & & 3 & \mathrm{eval} \\
 & & \mathrm{getbasic} & & 3 & \mathrm{getbasic} \\
 & & \mathrm{add} & & 3 & \mathrm{add} \\
 & & \mathrm{mkbasic} & & 2 & \mathrm{mkbasic}
\end{array}
$$

15 let-Expressions

As a warm-up let us first consider the treatment of local variables :-)

Let $e \equiv \textbf{let } y_1 = e_1; \ldots; y_n = e_n \textbf{ in } e_0$ be a **let**-expression.

The translation of e must deliver an instruction sequence that

- allocates local variables $y_1, \ldots, y_n$;
- in the case of
  CBV: evaluates $e_1, \ldots, e_n$ and binds the y_i to their values;
  CBN: constructs closures for the $e_1, \ldots, e_n$ and binds the y_i to them;
- evaluates the expression e_0 and returns its value.

Here, we consider the non-recursive case only, i.e. where y_j only depends on $y_1, \ldots, y_{j-1}$. We obtain for CBN:

$$
\begin{array}{lcl}
\mathrm{code}_V\ e\ \rho\ \mathrm{sd} & = & \mathrm{code}_C\ e_1\ \rho\ \mathrm{sd} \\
 & & \mathrm{code}_C\ e_2\ \rho_1\ (\mathrm{sd}+1) \\
 & & \ldots \\
 & & \mathrm{code}_C\ e_n\ \rho_{n-1}\ (\mathrm{sd}+n-1) \\
 & & \mathrm{code}_V\ e_0\ \rho_n\ (\mathrm{sd}+n) \\
 & & \mathtt{slide}\ \mathtt{n} \qquad\qquad // \text{ deallocates local variables}
\end{array}
$$

where $\quad \rho_j = \rho \oplus \{y_i \mapsto (L, \mathrm{sd}+i) \mid i = 1, \ldots, j\}$.

In the case of CBV, we use code_V for the expressions $e_1, \ldots, e_n$.

Warning!

All the e_i must be associated with the same binding for the global variables!

Example:

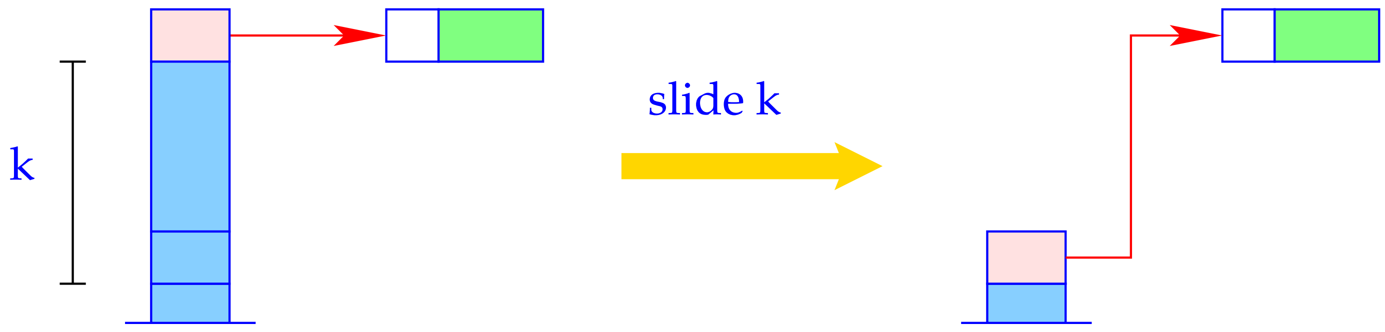
Consider the expression

$$e \equiv \textbf{let } a = 19; b = a * a \textbf{ in } a + b$$

for $\rho = \emptyset$ and sd $= 0$. We obtain (for CBV):

0	loadc 19	3	getbasic	3	pushloc 1
1	mkbasic	3	mul	4	getbasic
1	pushloc 0	2	mkbasic	4	add
2	getbasic	2	pushloc 1	3	mkbasic
2	pushloc 1	3	getbasic	3	slide 2

The instruction slide k deallocates again the space for the locals:



S[SP-k] = S[SP];
SP = SP - k;

16 Function Definitions

The definition of a function f requires code that allocates a functional value for f in the heap. This happens in the following steps:

- Creation of a Global Vector with the binding of the free variables;
- Creation of an (initially empty) argument vector;
- Creation of an F-Object, containing references to these vectors and the start address of the code for the body;

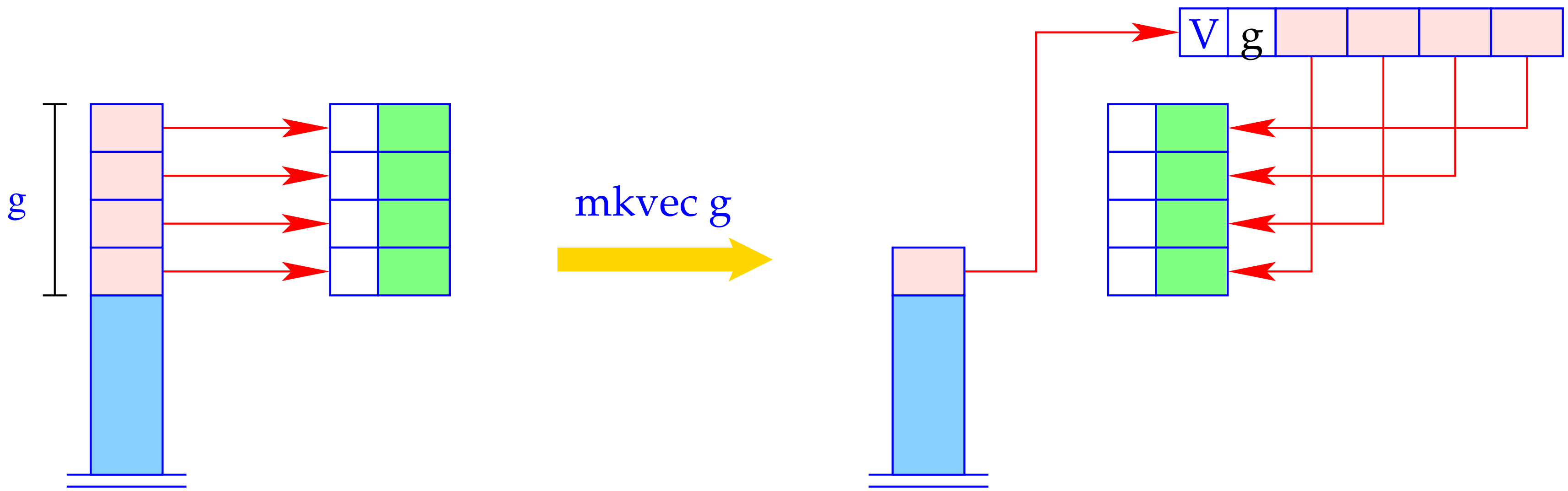
Separately, code for the body has to be generated.

Thus:

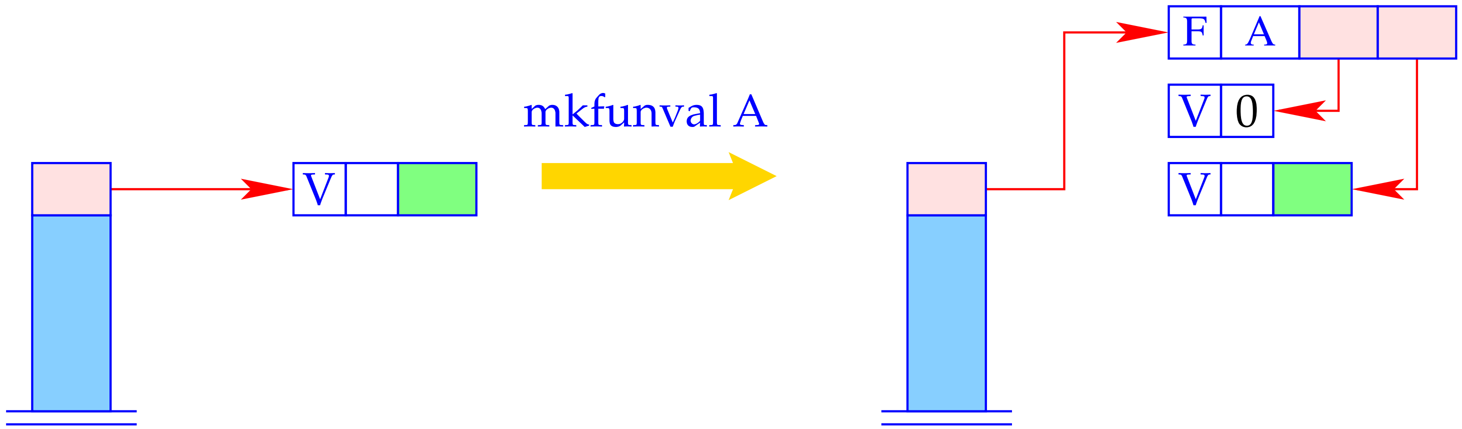
$$
\begin{array}{rcll}
\mathrm{code}_V\,(\mathbf{fn}\ x_0,\ldots,x_{k-1} \Rightarrow e)\,\rho\,\mathrm{sd} & = & & \mathrm{getvar}\ z_0\ \rho\ \mathrm{sd} \\
& & & \mathrm{getvar}\ z_1\ \rho\ (\mathrm{sd}+1) \\
& & & \ldots \\
& & & \mathrm{getvar}\ z_{g-1}\ \rho\ (\mathrm{sd}+g-1) \\
& & & \mathrm{mkvec\ g} \\
& & & \mathrm{mkfunval\ A} \\
& & & \mathrm{jump\ B} \\
& & \mathrm{A:} & \mathrm{targ\ k} \\
& & & \mathrm{code}_V\ e\ \rho'\ 0 \\
& & & \mathrm{return\ k} \\
& & \mathrm{B:} & \ldots
\end{array}
$$

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

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




```
h = new (V, n);
SP = SP - g + 1;
for (i=0; i<g; i++)
    h→v[i] = S[SP + i];
S[SP] = h;
```



a = new (V,0);
S[SP] = new (F, A, a, S[SP]);

Example:

Regard $f \equiv \textbf{fn}\ b \Rightarrow a + b$ for $\rho = \{a \mapsto (L, 1)\}$ and sd $= 1$.

$\text{code}_V\ f\ \rho\ 1$ produces:

1		pushloc 0	0	pushglob 0	2		getbasic
2		mkvec 1	1	eval	2		add
2		mkfunval A	1	getbasic	1		mkbasic
2		jump B	1	pushloc 1	1		return 1
0	A :	targ 1	2	eval	2	B :	...

The secrets around targ k and return k will be revealed later :-)

17 Function Application

Function applications correspond to function calls in C.
The necessary actions for the evaluation of $e' \; e_0 \; \ldots \; e_{m-1}$ are:

- Allocation of a stack frame;
- Transfer of the actual parameters , i.e. with:
  CBV: Evaluation of the actual parameters;
  CBN: Allocation of closures for the actual parameters;
- Evaluation of the expression e' to an F-object;
- Application of the function.

Thus for CBN:

$$
\begin{array}{rcll}
\text{code}_V\ (e'\ e_0\ \ldots\ e_{m-1})\ \rho\ \text{sd} & = & \text{mark A} & \text{// Allocation of the frame} \\
 & & \text{code}_C\ e_{m-1}\ \rho\ (\text{sd}+3) & \\
 & & \text{code}_C\ e_{m-2}\ \rho\ (\text{sd}+4) & \\
 & & \ldots & \\
 & & \text{code}_C\ e_0\ \rho\ (\text{sd}+m+2) & \\
 & & \text{code}_V\ e'\ \rho\ (\text{sd}+m+3) & \text{// Evaluation of } e' \\
 & & \text{apply} & \text{// corresponds to call} \\
 & A: & \ldots &
\end{array}
$$

To implement CBV, we use code_V instead of code_C for the arguments e_i.

Example: For $(f\ 42)$, $\rho = \{f \mapsto (L, 2)\}$ and $\text{sd} = 2$, we obtain with CBV:

2	mark A	6	mkbasic	7		apply
5	loadc 42	6	pushloc 4	3	A :	...

A Slightly Larger Example:

$$\textbf{let}\ a = 17;\ f = \textbf{fn}\ b \Rightarrow a + b\ \textbf{in}\ f\ 42$$

For CBV and $\quad$ sd $= 0$ $\quad$ we obtain:

0	loadc 17	2		jump B	2		getbasic	5		loadc 42
1	mkbasic	0	A:	targ 1	2		add	5		mkbasic
1	pushloc 0	0		pushglob 0	1		mkbasic	6		pushloc 4
2	mkvec 1	1		getbasic	1		return 1	7		apply
2	mkfunval A	1		pushloc 1	2	B:	mark C	3	C:	slide 2