Beispiel:

Betrachte $e \equiv (b + c)$ für $\rho = \{b \mapsto (L, 1), c \mapsto (G, 0)\}$ und $\mathrm{kp} = 1$.

Dann ist für CBN:

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

15 let-Ausdrücke

Zum Aufwärmen betrachten wir zuerst die Behandlung lokaler Variablen :-)

Sei $e \equiv \textbf{let } y_1 = e_1; \ldots; y_n = e_n \textbf{ in } e_0$ ein **let**-Ausdruck. Die Übersetzung von e muss eine Befehlsfolge liefern, die

- lokale Variablen $y_1, \ldots, y_n$ auf dem Stack anlegt;
- im Falle von
  CBV: $e_1, \ldots, e_n$ auswertet und die y_i an deren Werte bindet;
  CBN: Abschlüsse für $e_1, \ldots, e_n$ herstellt und die y_i daran bindet;
- den Ausdruck e_0 auswertet und schließlich dessen Wert zurück liefert.

Wir betrachten hier zuerst nur den nicht-rekursiven Fall, d.h. wo y_j nur von $y_1, \ldots, y_{j-1}$ abhängt. Dann erhalten wir für CBN:

$$
\begin{array}{lcll}
\text{code}_V\ e\ \rho_0\ \text{kp} & = & \text{code}_C\ e_1\ \rho_0\ \text{kp} & \\
 & & \text{code}_C\ e_2\ \rho_1\ (\text{kp}+1) & \\
 & & \ldots & \\
 & & \text{code}_C\ e_n\ \rho_{n-1}\ (\text{kp}+n-1) & \\
 & & \text{code}_V\ e_0\ \rho_n\ (\text{kp}+n) & \\
 & & \text{slide n} & \quad // \text{ gibt lok. Variablen auf}
\end{array}
$$

wobei $\rho_j = \rho_{j-1} \oplus \{y_j \mapsto (L, \text{kp}+j)\}$ für $j = 1, \ldots, n$.

Im Falle von CBV müssen die Werte der Variablen y_i sofort ermittelt werden!

Dann benutzen wir für die Ausdrücke $e_1, \ldots, e_n$ ebenfalls code_V.

Achtung!

Die e_i müssen mit den gleichen Bindungen für die (nicht verdeckten) globalen Variablen versehen werden!

Beispiel:

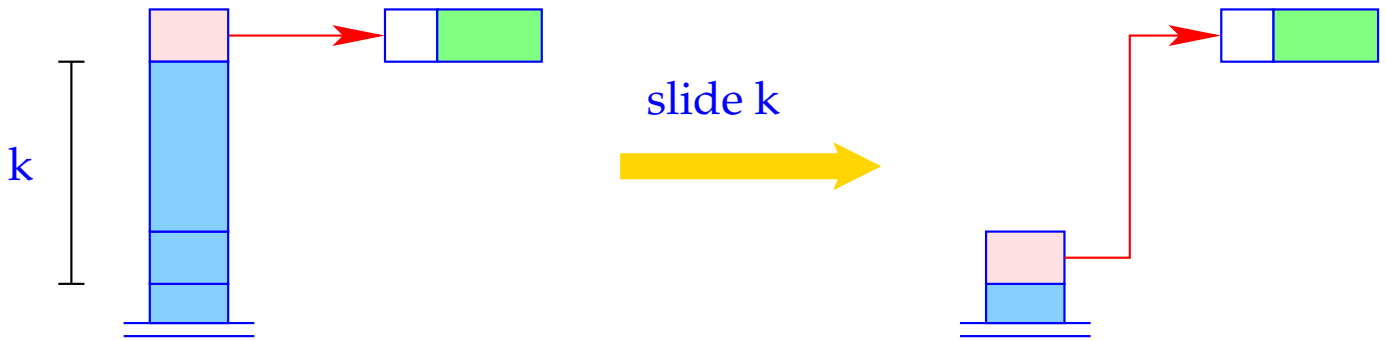
Betrachte den Ausdruck

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

für $\rho = \emptyset$ und kp $= 0$. Dann ergibt sich (für 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

Der Befehl slide k gibt den Platz von k lokalen Variablen wieder auf:




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

16 Funktions-Definitionen

Für eine Funktion f müssen wir Code erzeugen, die einen funktionalen Wert für f in der Halde anlegt. Das erfordert:

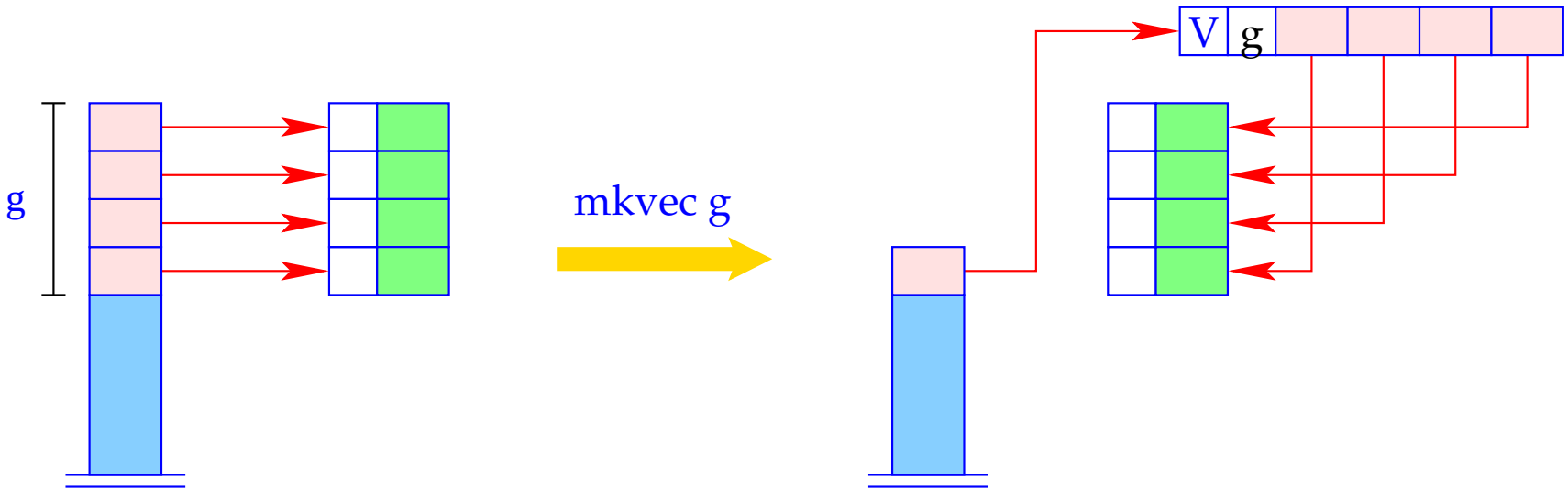
- Erzeugen des Global Vector mit den Bindungen der freien Variablen;
- Erzeugen eines (anfänglich leeren) Argument-Vektors;
- Erzeugen eines F-Objekts, das zusätzlich die Anfangs-Adresse des Codes zur Auswertung des Rumpfs enthält;
- Code zur Auswertung des Rumpfs.

Folglich:

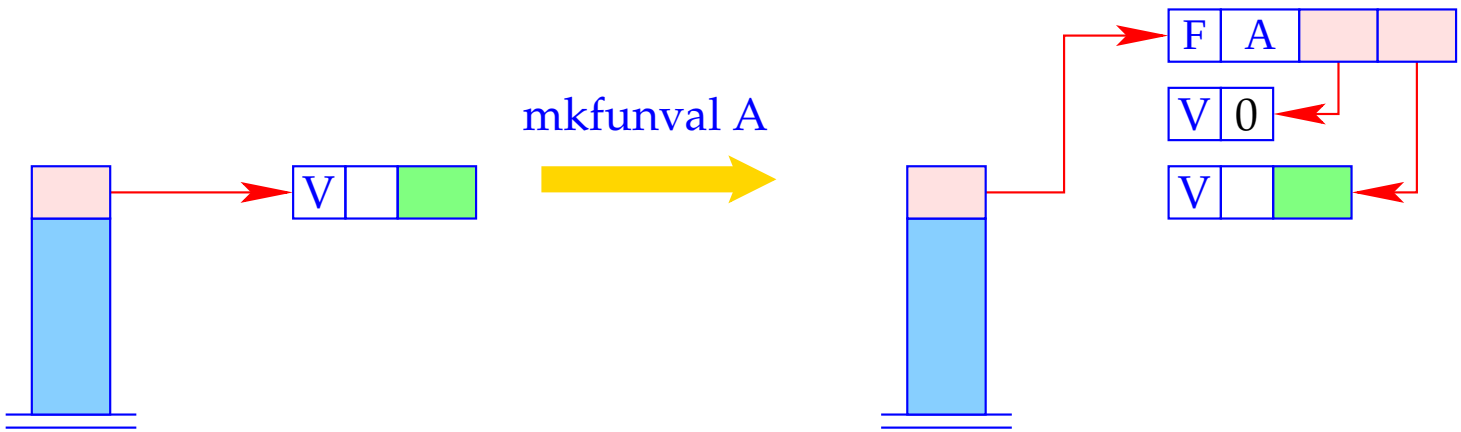
$$
\begin{array}{lll}
\text{code}_V\ (\textbf{fn}\ x_0, \ldots, x_{k-1} \Rightarrow e)\ \rho\ \text{kp} & = & \text{getvar}\ z_0\ \rho\ \text{kp} \\
 & & \text{getvar}\ z_1\ \rho\ (\text{kp}+1) \\
 & & \ldots \\
 & & \text{getvar}\ z_{g-1}\ \rho\ (\text{kp}+g-1) \\
 & & \text{mkvec g} \\
 & & \text{mkfunval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

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

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




```
h = new (V, g);
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]);

Beispiel:

Betrachte $f \equiv \mathbf{fn}\ b \Rightarrow a + b$ für $\rho = \{a \mapsto (L, 1)\}$ und $\mathrm{kp} = 1$.

Dann liefert $\mathrm{code}_V\ f\ \rho\ 1$:

```
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 :   ...
```

Die Geheimnisse um targ k und return k lüften wir später.

17 Funktionsanwendungen

Funktions-Anwendungen entsprechen Funktions-Aufrufen in C. Notwendige Aktionen zur Auswertung von $e'\ e_0\ \ldots\ e_{m-1}$ sind:

- Anlegen eines Kellerrahmens;
- Parameter-Übergabe, d.h. bei:
  CBV: Auswerten der aktuellen Parameter;
  CBN: Anlegen von Abschlüssen für die aktuellen Parameter;
- Auswerten der Funktion e' zu einem F-Objekt;
- Anwenden der Funktion.

Folglich für CBN:

$$
\begin{array}{lll}
\text{code}_V\ (e'\ e_0\ \ldots\ e_{m-1})\ \rho\ \text{kp} & = & \text{mark A} \qquad\qquad\qquad\qquad // \text{ Anlegen des Rahmens} \\
 & & \text{code}_C\ e_{m-1}\ \rho\ (\text{kp}+3) \\
 & & \text{code}_C\ e_{m-2}\ \rho\ (\text{kp}+4) \\
 & & \ldots \\
 & & \text{code}_C\ e_0\ \rho\ (\text{kp}+m+2) \\
 & & \text{code}_V\ e'\ \rho\ (\text{kp}+m+3) \qquad // \text{ Auswerten der Funktion} \\
 & & \text{apply} \qquad\qquad\qquad\qquad // \text{ entspricht call} \\
 & A: & \ldots
\end{array}
$$

Wenn wir CBV implementieren wollen, müssen die Argumente vor dem Funktions-Aufruf ausgewertet werden.

Dann benutzen wir code_V anstelle von code_C für die Argumente e_i :-)

Beispiel:

Für $(f\ 42)$, $\rho = \{f \mapsto (L, 2)\}$ und kp $= 2$ liefert das bei CBV:

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

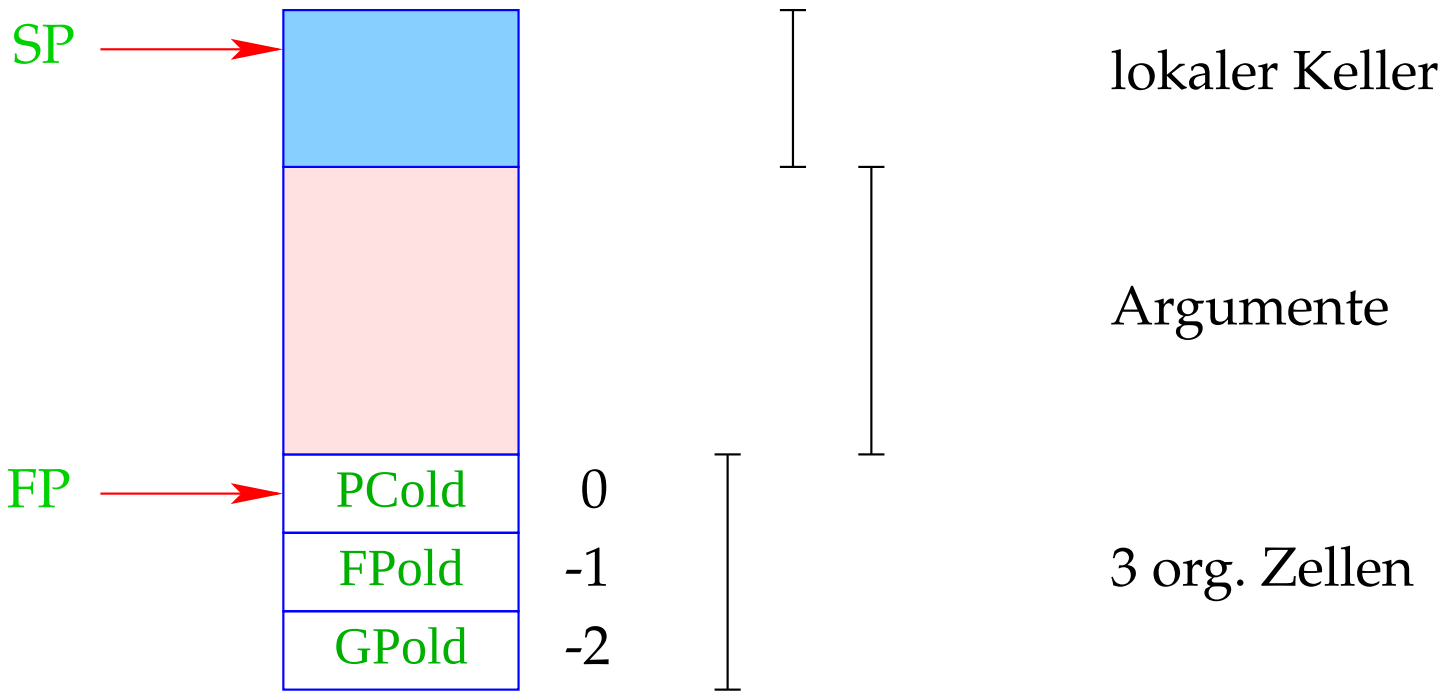
Ein etwas größeres Beispiel:

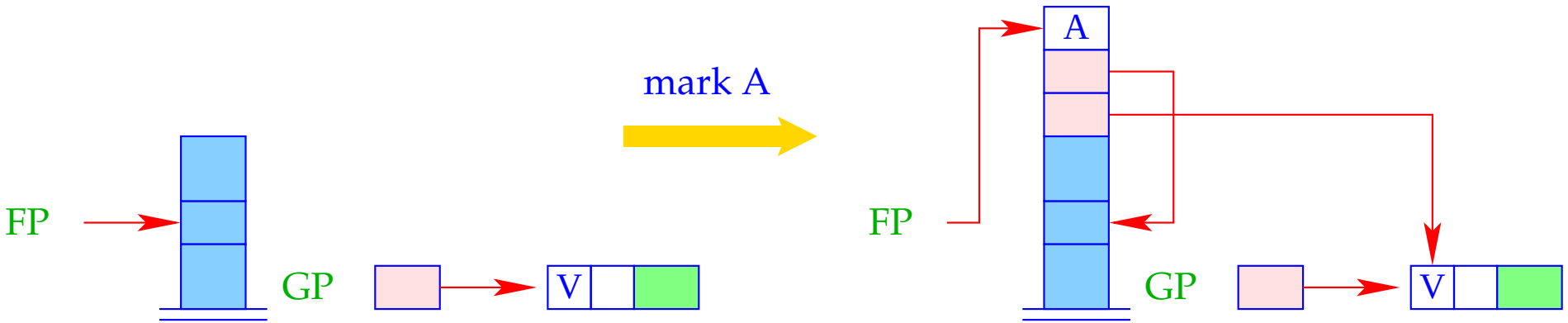
$$\textbf{let } a = 17; f = \textbf{fn } b \Rightarrow a + b \textbf{ in } f\ 42$$

Bei CBV erhalten wir für $\mathrm{kp} = 0$:

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

Vor der Implementierung der neuen Instruktionen müssen wir die Organisation eines Kellerrahmens festlegen:





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