MaMa — lihtsustatud funktsionaalsete keelte abstraktne masin

Funktsionaalne keel PuF

Käsitleme funktsionaalset mini-keelt PuF (”Pure Functions”).

Programm on avaldis e kujul:

$$
\begin{array}{rcl}
e & ::= & b \mid x \mid (\Box_1\, e) \mid (e_1\, \Box_2\, e_2) \\
& \mid & (\textbf{if } e_0 \textbf{ then } e_1 \textbf{ else } e_3) \\
& \mid & (e'\ e_0\ \ldots\ e_{k-1}) \\
& \mid & (\textbf{fn } x_0, \ldots, x_{k-1} \Rightarrow e) \\
& \mid & (\textbf{let } x_1 = e_1; \ldots; x_n = e_n \textbf{ in } e_0) \\
& \mid & (\textbf{letrec } x_1 = e_1; \ldots; x_n = e_n \textbf{ in } e_0)
\end{array}
$$

- Lihtsuse mõttes on ainsaks baastüübiks `int`.
- Hiljem lisame ka andmestruktuurid.

Funktsionaalne keel PuF

Näide: faktoriaali arvutava funktsiooni saab defineerida alljärgnevalt:

$$\begin{aligned} \text{fac} \quad = \quad & \textbf{fn}\ x \Rightarrow \textbf{if}\ x \leq 1\ \textbf{then}\ 1 \\ & \textbf{else}\ x \cdot \text{fac}\,(x-1) \end{aligned}$$

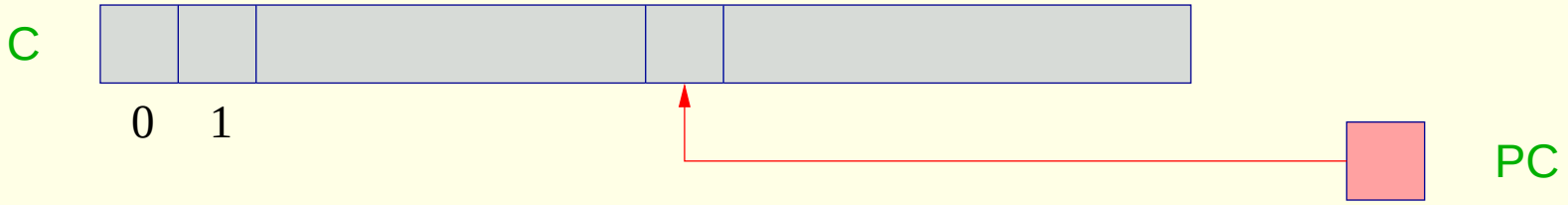
Funktsionaalsetes keeltes on kasutusel kaks erinevat semantikat:

CBV: argumendid väärtustatakse enne nende funktsioonile edastamist (näit. SML);

CBN: argumendid edastatakse funktsioonile ilma väärtustamata ning väärtustatakse alles sis kui nende väärtust on vaja (näit. Haskell).

MaMa arhitektuur

Kood:

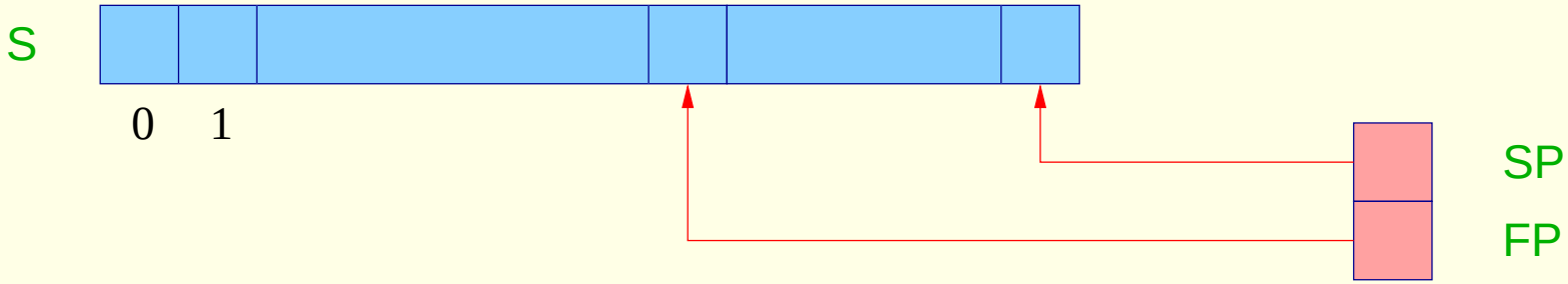


C = Code-store — mälupiirkond MaMa programmikoodi hoidmiseks; igas pesas on täpselt üks abstraktse masina käsk.

PC = Program Counter — viitab järgmisena täidetavale käsule.

MaMa arhitektuur

Magasin:



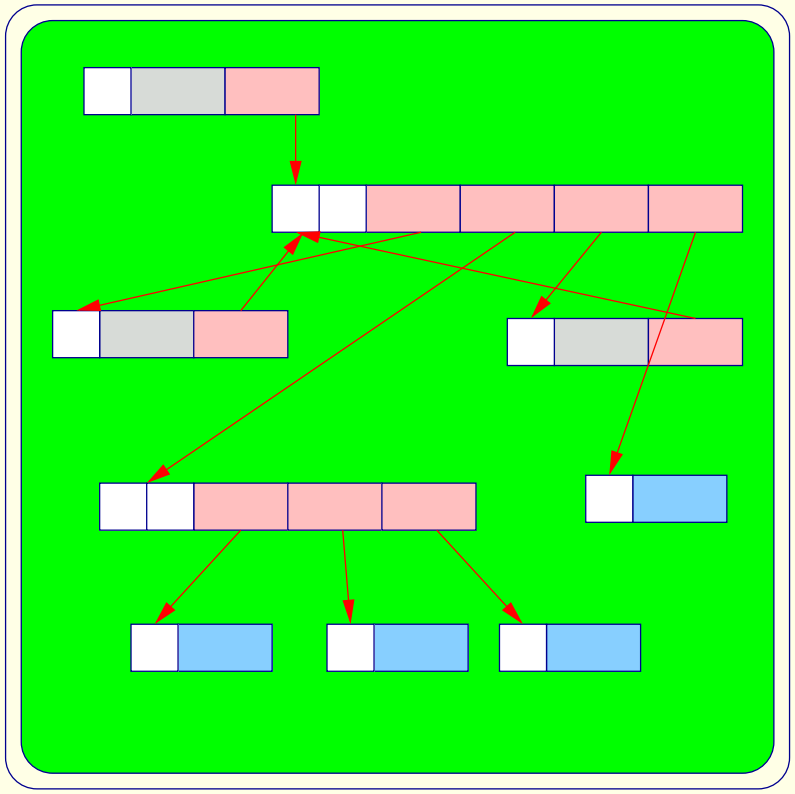
S = Stack — igas pesas võib olla kas baasväärtus või aadress;

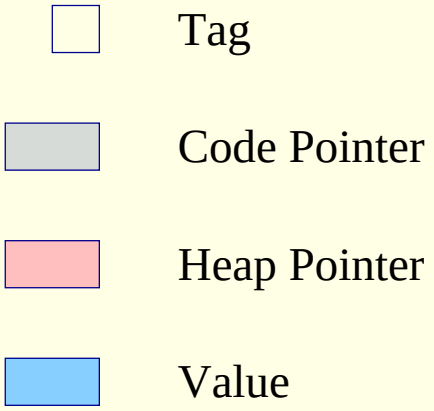
SP = Stack-Pointer — viitab tipmisele täidetud pesale;

FP = Frame-Pointer — viitab kehtivale freimile.

MaMa arhitektuur

Kuhi:

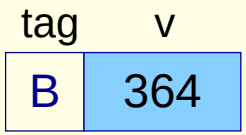




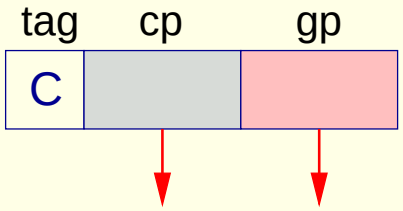
H = Heap — mälupiirkond dünaamiliste andmete hoidmiseks.

MaMa arhitektuur

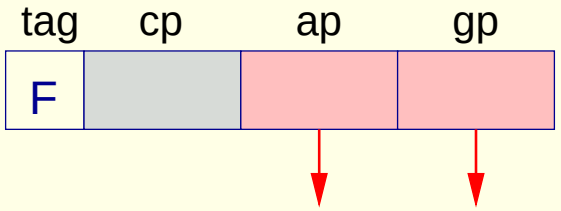
Kuhjas võivad olla järgmised objektid:



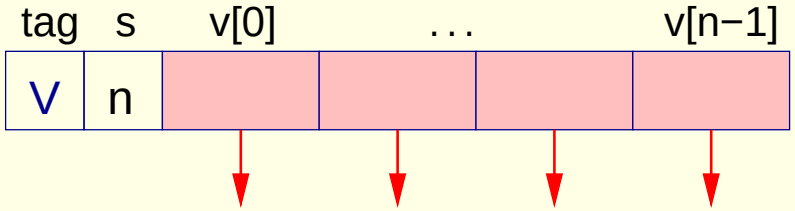
Basic Value



Closure



Function



Vector

MaMa arhitektuur

Käsk **new**$(tag, args)$ loob etteantud liiki uue kuhjaobjekti ning väljastab viida temale.

Avaldisele e vastava koodi genereerimiseks kasutame kolme erinevat funktsiooni:

- $\text{code}_B\ e$ — leiab avaldise e väärtuse ja salvestab selle magasini tippu (ainult baastüüpide jaoks);
- $\text{code}_V\ e$ — leiab avaldise e väärtuse, salvestab selle kuhja ning väljastab magasini tippu viida vastavale kuhjaobjektile;
- $\text{code}_C\ e$ — ei väärtusta avaldist, vaid salvestab *e sulundi* (closure) kuhja ning väljastab viida sellele sulundile magasini tippu.

Lihtsate avaldiste transleerimine

Avaldised, mis koosnevad ainult konstantidest, operaatorite rekendustest ja tingimusavaldistest, transleeritakse analoogselt imperatiivsete keeltega:

$$
\begin{aligned}
\text{code}_B\ b\ \rho\ \text{sd} &= \texttt{loadc b} \\
\text{code}_B\ (\square_1\, e)\ \rho\ \text{sd} &= \text{code}_B\ e\ \rho\ \text{sd} \\
&\quad\ \texttt{op}_1 \\
\text{code}_B\ (e_1\, \square_2\, e_2)\ \rho\ \text{sd} &= \text{code}_B\ e_1\ \rho\ \text{sd} \\
&\quad\ \text{code}_B\ e_2\ \rho\ (\text{sd}+1) \\
&\quad\ \texttt{op}_2
\end{aligned}
$$

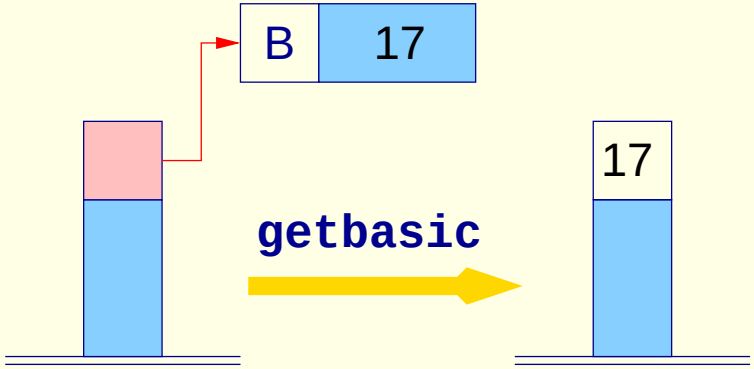
Lihtsate avaldiste transleerimine

$$
\begin{array}{lll}
\text{code}_B\ (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\ \rho\ \text{sd} & = & \text{code}_B\ e_0\ \rho\ \text{sd} \\
 & & \texttt{jumpz A} \\
 & & \text{code}_B\ e_1\ \rho\ \text{sd} \\
 & & \texttt{jump B} \\
 & \text{A:} & \text{code}_B\ e_2\ \rho\ \text{sd} \\
 & \text{B:} & \ldots
\end{array}
$$

Teiste avaldiste korral arvutame kõigepealt tema väärtuse kuhjas ja seejärel võtame väärtuse väljastatud viida kaudu:

$$
\begin{array}{lll}
\text{code}_B\ e\ \rho\ \text{sd} & = & \text{code}_V\ e\ \rho\ \text{sd} \\
 & & \texttt{getbasic}
\end{array}
$$

Lihtsate avaldiste transleerimine




```
if (S[SP]→tag ≠ B)
      Error ("Not Basic");
else
      S[SP] = S[SP]→v;
```

- ρ tähistab *aadresskeskkonda* milles avaldisi transleeritakse. Aadresskeskkonnad on kujul:

$$\rho : \mathit{Vars} \to \{L, G\} \times \mathbb{Z}$$

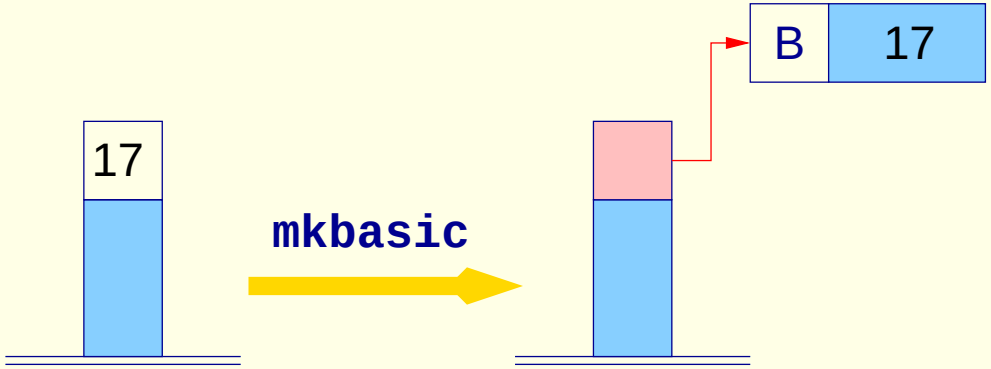
- Ekstraargument sd (stack difference) simuleerib registri SP liikumist kui käsud modifitseerivad magasini. Kasutame hiljem muutujate adresseerimisel.

Lihtsate avaldiste transleerimine

Funktsioon code_V lihtsate avaldiste jaoks on analoogne funktsiooniga code_B, kuid loob kuhja baasobjekti.

$$
\begin{array}{lcl}
\text{code}_V\ b\ \rho\ \text{sd} & = & \texttt{loadc b} \\
 & & \texttt{mkbasic} \\
\text{code}_V\ (\square_1\, e)\ \rho\ \text{sd} & = & \text{code}_B\ e\ \rho\ \text{sd} \\
 & & \texttt{op}_1 \\
 & & \texttt{mkbasic} \\
\text{code}_V\ (e_1\, \square_2\, e_2)\ \rho\ \text{sd} & = & \text{code}_B\ e_1\ \rho\ \text{sd} \\
 & & \text{code}_B\ e_2\ \rho\ (\text{sd}+1) \\
 & & \texttt{op}_2 \\
 & & \texttt{mkbasic}
\end{array}
$$

Lihtsate avaldiste transleerimine



S[SP] = new(B,S[SP]);

$\text{code}_V\ (\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2)\ \rho\ sd\ =$

$\text{code}_B\ e_0\ \rho\ sd$
`jumpz A`
$\text{code}_V\ e_1\ \rho\ sd$
`jump B`
A: $\text{code}_V\ e_2\ \rho\ sd$
B: $\ldots$

Ligipääs muutujatele

Näide: vaatame funktsiooni f

$$
\begin{array}{lll}
\textbf{let} & c = 5 & \\
 & f = \textbf{fn}\ a \quad \Rightarrow & \textbf{let}\ b = a * a \\
 & & \textbf{in}\ b + c \\
\textbf{in}\ \ldots & &
\end{array}
$$

Funktsioon f kasutab *globaalset* muutujat c ning *lokaalseid* muutujaid a (formaalne parameeter) ja b (defineeritud sisemise **let-**avaldise poolt).

Globaalse muutuja väärtus on määratud funktsiooni *konstrueerimise* ajal (staatiline skoopimine!) ning on täitmisajal otse kasutatav.

Ligipääs muutujatele

Globaalsed muutujad

- Globaalsete muutujatega seotud väärtused hoitakse kuhjas vektorina (Global Vector).
- Adresseeritakse üksteisele järgnevalt alates 0-st.
- F-objekti või C-objekti konstrueerimisel leitakse funktsiooni või avaldise globaalvektor ja tema aadress salvestatakse objekti gp-komponenti.
- Avaldise väärtustamise ajal viitab register GP (Global Pointer) hetkel kehtivale globaalvektorile.

Ligipääs muutujatele

Lokaalsed muutujad

Lokaalseid muutujaid hallatakse magasinis freimides.

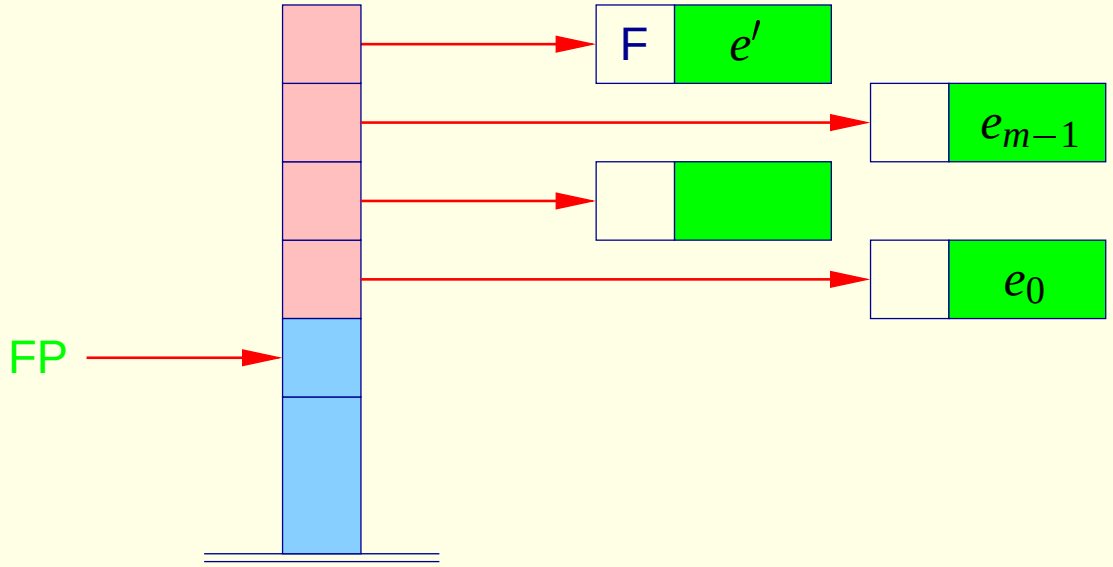
Olgu $e \equiv e'\ e_0\ \ldots\ e_{m-1}$ funktsiooni e' aplikatsioon argumentidele $e_0, \ldots, e_{m-1}$.

NB! Funktsiooni e' aarsus ei pea olema m.

- Lokaalseid muutujaid hallatakse magasinis freimides.
- PuF funktsioonid on *karritud* (curried) $f : t_1 \to t_2 \to \cdots \to t_n \to t$.
- Seega võib f olla rakendatud vähemale kui n argumendile.
- Kui t on funktsionalset tüüpi, siis võib f olla rakendatud ka rohkemale kui n argumendile.

Ligipääs muutujatele

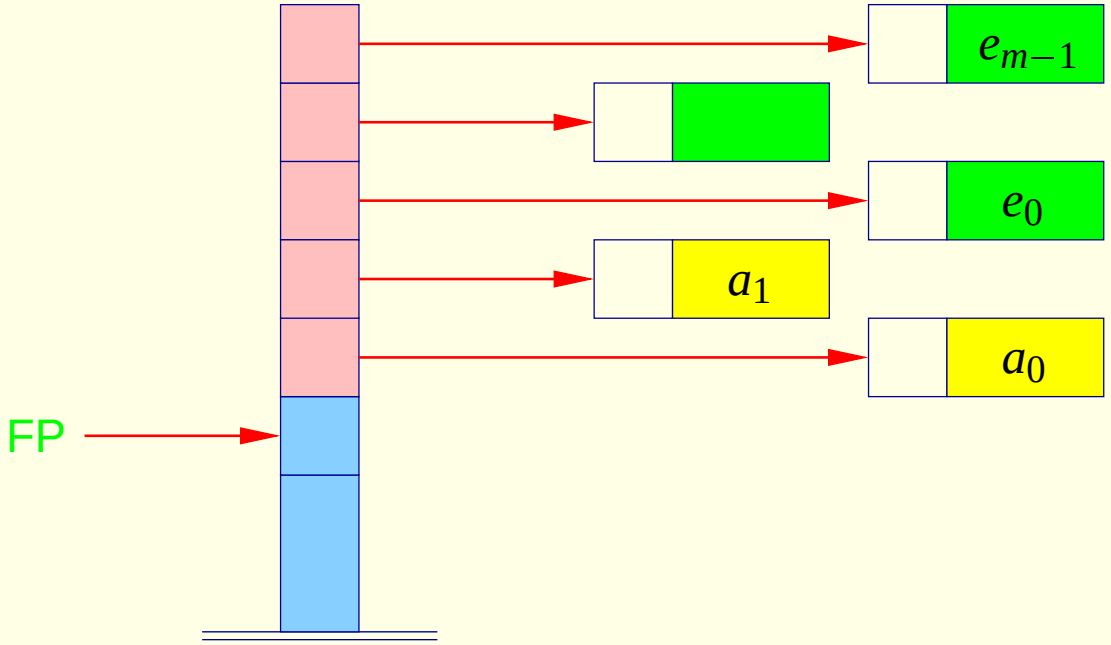
Võimalik magasini organiseerimine:



+ Parameetreid saab adresseerida FP suhtes.
– Avaldise e' lokaalseid muutujaid ei saa adresserida FP suhtes.
– Kui e' on n-aarne funktsioon ja $n < m$, siis ülejäänud $m - n$ argumenti tuleb freimis ümberpaigutada.

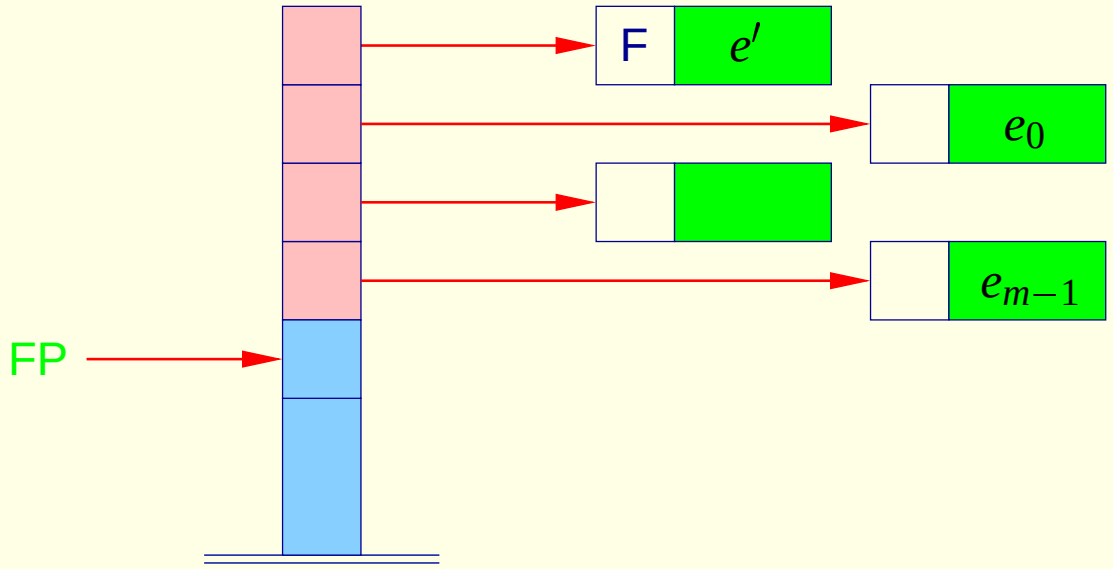
Ligipääs muutujatele

– Kui e' väärtustub funktsiooniks, mida on juba osaliselt rakendatud parameetritele $a_0, \ldots, a_{k-1}$, siis tuleb need liigutada e_0-i alla.



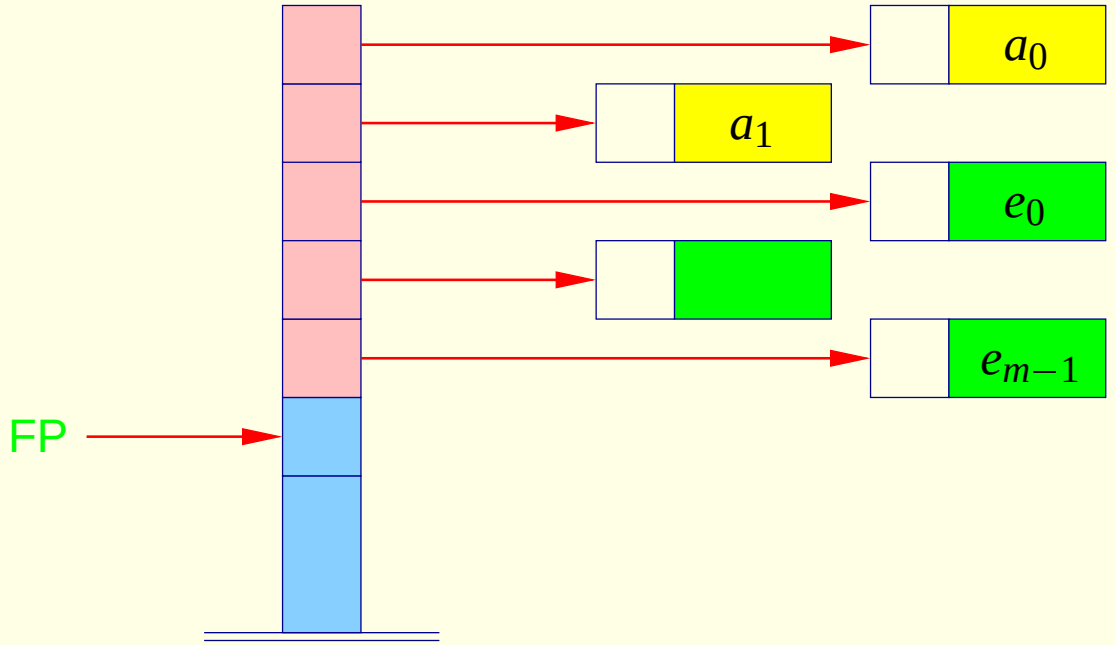
Ligipääs muutujatele

Alternatiivne magasini organiseerimine:



+ Lisaargumendid $a_0, \ldots, a_{k-1}$ ja lokaalsed muutujad saab salvestada magasini pärast argumente.

Ligipääs muutujatele

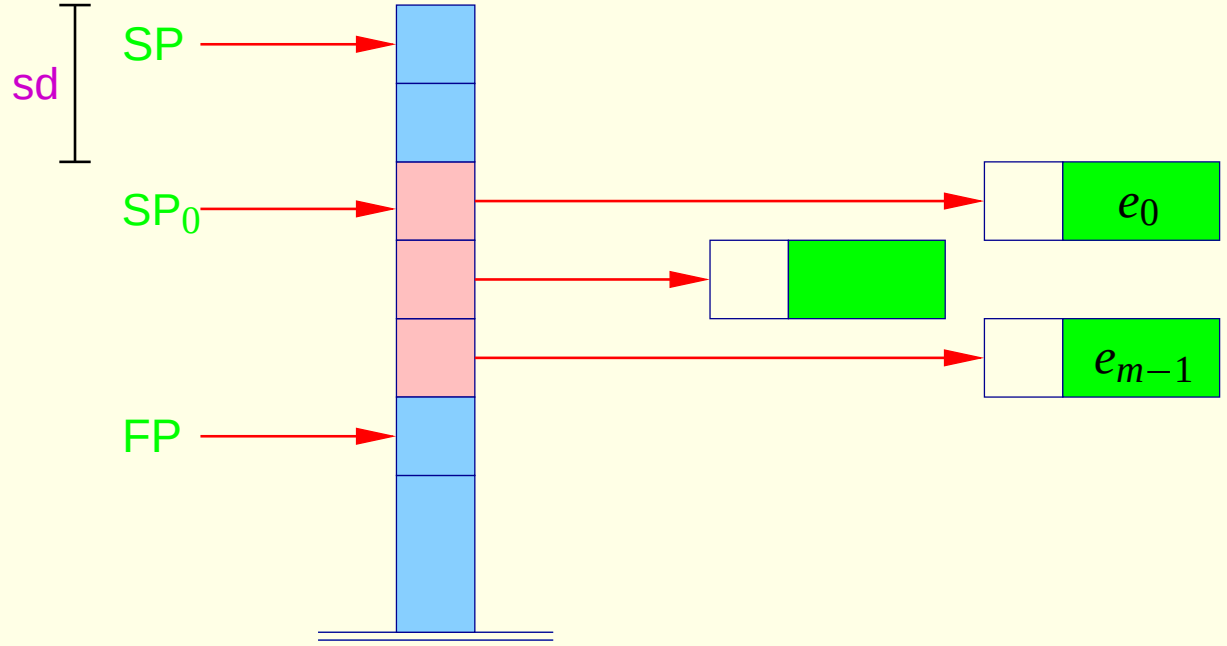


– Formaalsete parameetrite adresseerimine FP suhtes pole enam võimalik.

Ligipääs muutujatele

Lahendus:

- Adresseerime nii argumente, kui lokaalseid muutujaid, registri SP suhtes!
- Kuid SP muutub programmi täitmisajal . . .



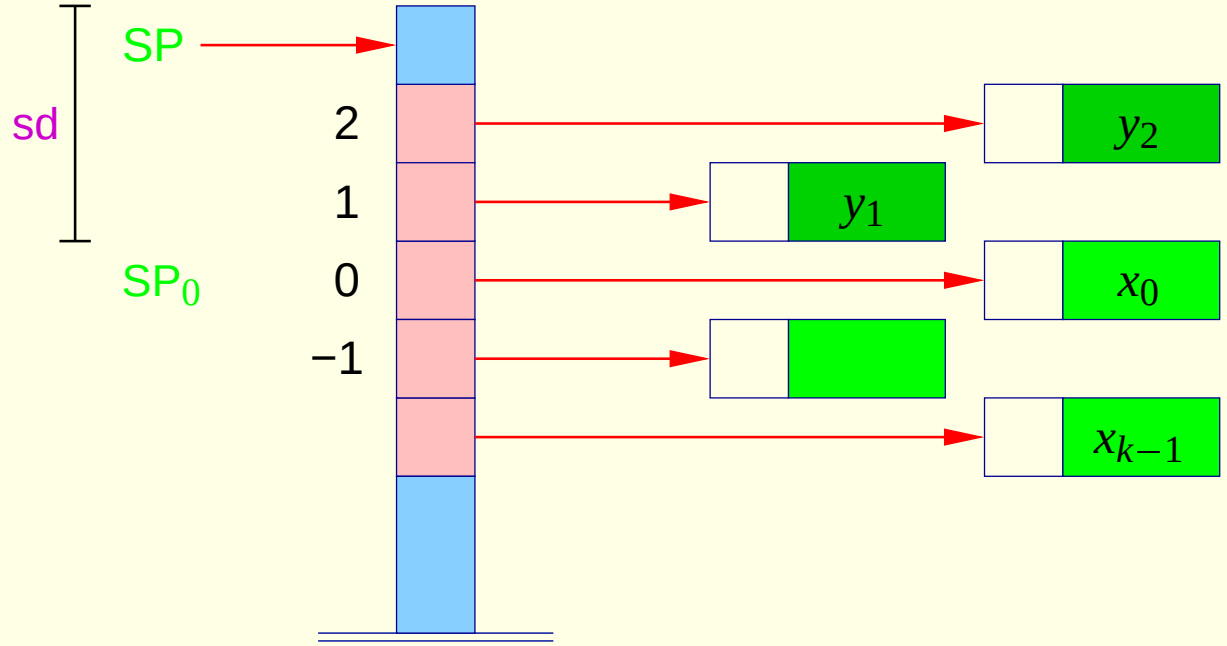
Ligipääs muutujatele

- Magasini kaugus, sd, näitab registri SP hetkeväärtuse erinevust tema väärtusest SP_0 funktsiooni kehasse sisenemisel.
- Magasini kaugust on võimalik määrata staatiliselt simuleerides magasini muutumist käskude täitmisel.
- Formaalsed parameetrid $x_0, x_1, x_2, \ldots$ seotakse mittepositiivsete suhtadressitega vastavalt 0, -1, -2, ...; so. $\rho\, x_i = (L, -i)$.
- i-nda formaalse parameetri absoluutaadress on:

$$SP_0 - i = (SP - sd) - i$$

- Lokaalsed **let**-muutujad lisatakse üksteisejärel magasini tippu.

Ligipääs muutujatele



- Lokaalsed muutujad $y_1, y_2, \ldots$ seotakse postiivsete suhtaadressitega; so. $\rho\, y_i = (L, i)$.
- i-nda lokaalse muutuja absoluutaadress on:

$$\mathrm{SP}_0 + i = (\mathrm{SP} - sd) + i$$

Ligipääs muutujatele

CBN semantika korral emiteeritakse muutujate väärtustamiseks kood:

$$\begin{array}{lcll} \text{code}_V\ x\ \rho\ sd & = & \texttt{pushloc}\ (sd - i) & \text{kui } \rho\ x = (L, i) \\ & & \texttt{eval} & \\ \text{code}_V\ x\ \rho\ sd & = & \texttt{pushglob i} & \text{kui } \rho\ x = (G, i) \\ & & \texttt{eval} & \end{array}$$

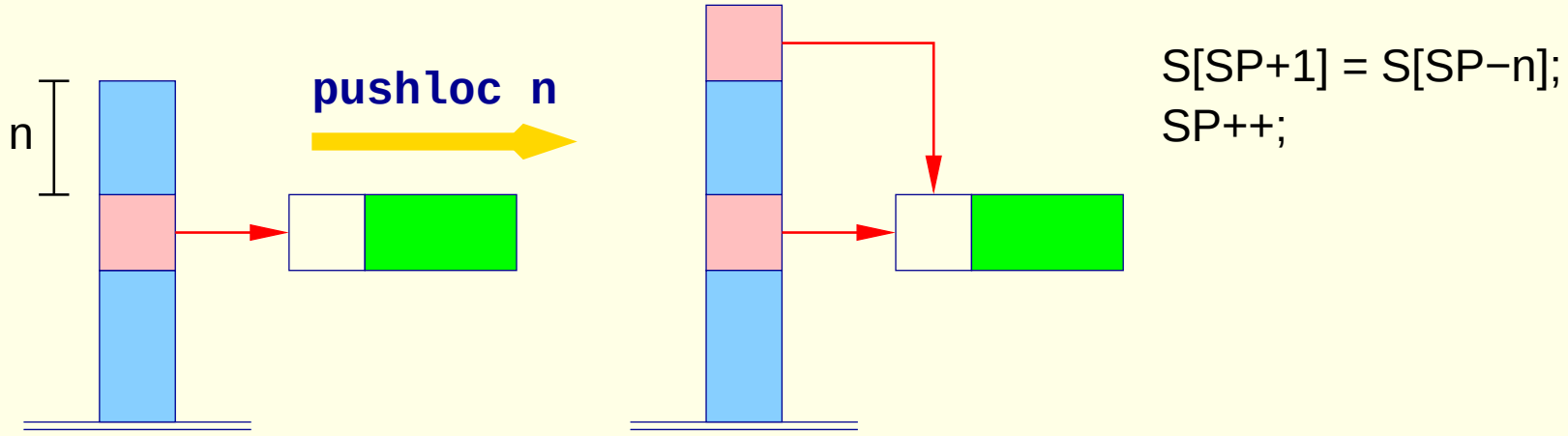
Käsk `eval` kontrollib, kas muutuja on juba väärtustatud või mitte ning teisel juhul väärtustab selle (vaatame lähemalt hiljem).

CBV semantika korral ei ole `eval` käsku vaja genereerida.

Ligipääs muutujatele

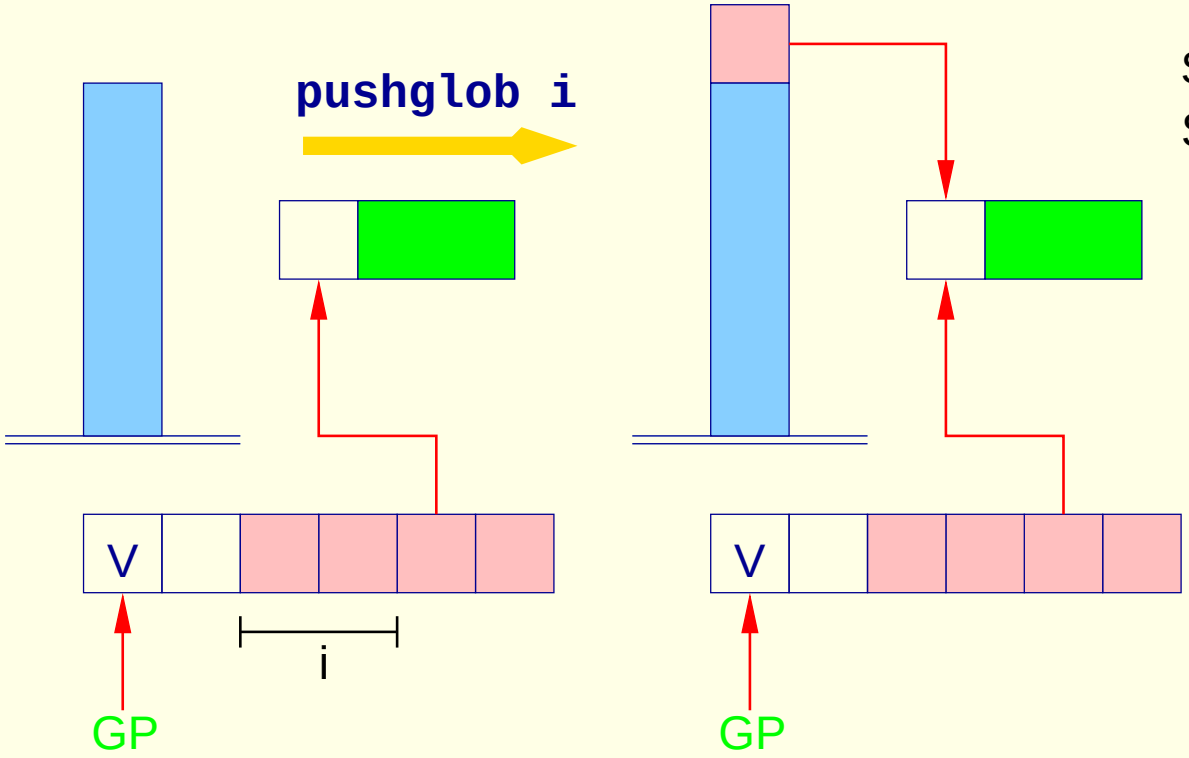
Lokaalne muutuja suhtaadressiga i loetakse magasinipesast $S[a]$, kus

$$a = \mathrm{SP} - (sd - i) = (\mathrm{SP} - sd) + i = \mathrm{SP}_0 + i$$



Ligipääs muutujatele

Globaalne muutuja loetakse globaalvektorist.



S[SP+1] = GP→v[i];
SP++;

Ligipääs muutujatele

Näide:

Olgu $e \equiv (b + c)$ keskkonnas $\rho = \{b \mapsto (L, 1), c \mapsto (G, 0)\}$ ja $sd = 1$.

CBN semantika korral $\text{code}_V\ e\ \rho$ sd emiteerib koodi:

```
1   pushloc 0        3   eval
2   eval             3   getbasic
2   getbasic         3   add
2   pushglob 0       2   mkbasic
```