WiM — lihtsustatud loogiliste keelte abstraktne masin

Loogiline keel Proll

Käsitleme loogilist mini-keelt Proll (”Prolog-light”).

Proll on lihtsustatud versioon Prologist; sh. me ei vaatle:

- aritmeetilisi operatsioone;
- lõike operaatorit;
- assert ja retract abil ennast modifitseerivaid programme.

Loogiline keel Proll

Programm p on järgmise süntaksiga:

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

- *term* t on kas aatom (konstant), muutuja, anonüümne muutuja või konstruktori aplikatsioon;
- *eesmärk* (goal) g on kas predikaadi aplikatsioon või unifikatsioon;
- *reegel* (clause) c koosneb *peast* $p(X_1,\ldots,X_k)$ ja *kehast*; (so. eesmärkide jadast);
- programm koosneb reeglitest ja *päringust* (query).

Loogiline keel Proll

Näide:

$$
\begin{array}{lcl}
\texttt{bigger}(X,Y) & \leftarrow & X = \mathit{elephant},\ Y = \mathit{horse} \\
\texttt{bigger}(X,Y) & \leftarrow & X = \mathit{horse},\ Y = \mathit{donkey} \\
\texttt{bigger}(X,Y) & \leftarrow & X = \mathit{donkey},\ Y = \mathit{dog} \\
\texttt{bigger}(X,Y) & \leftarrow & X = \mathit{donkey},\ Y = \mathit{monkey} \\
\texttt{is\_bigger}(X,Y) & \leftarrow & \texttt{bigger}(X,Y) \\
\texttt{is\_bigger}(X,Y) & \leftarrow & \texttt{bigger}(X,Z),\ \texttt{is\_bigger}(Z,Y)
\end{array}
$$

? $\texttt{is\_bigger}(\mathit{elephant}, \mathit{dog})$

Loogiline keel Proll

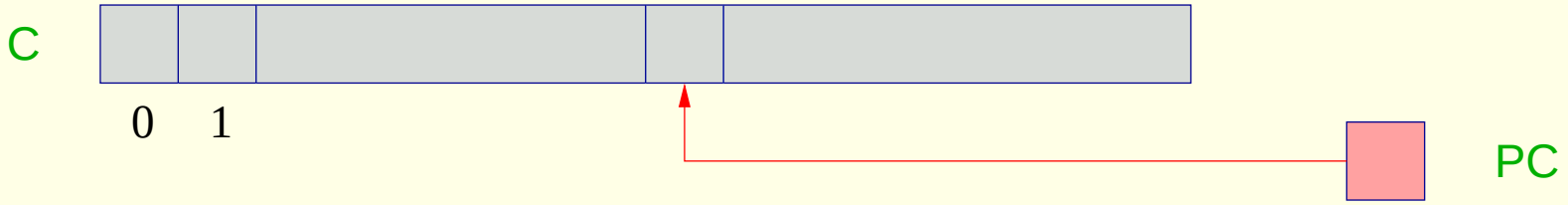
Näide:

$$\begin{aligned}
&\texttt{app}(X,Y,Z) \quad \leftarrow \quad X = [\,],\ Y = Z \\
&\texttt{app}(X,Y,Z) \quad \leftarrow \quad X = [H \mid X'],\ Z = [H \mid Z'],\ \texttt{app}(X',Y,Z') \\
&?\ \texttt{app}(X,[Y,c],[a,b,Z])
\end{aligned}$$

- $[\,]$ on tühja listi aatom;
- $[H \mid Z]$ on listi konstruktori aplikatsioon;
- $[a,b,Z]$ on termi $[a \mid [b \mid [Z \mid [\,]]]]$ lühend.

WiM arhitektuur

Kood:

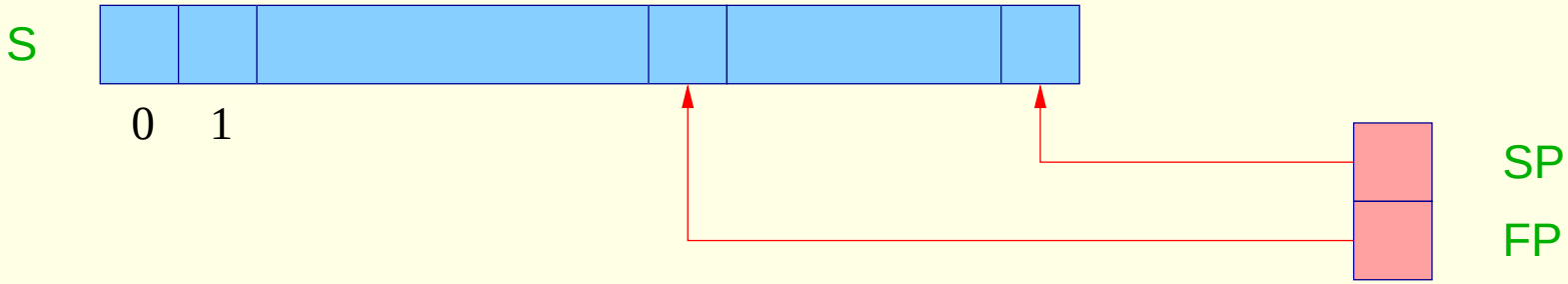


C = Code-store — mälupiirkond WiM programmikoodi hoidmiseks; igas pesas on täpselt üks abstraktse masina käsk.

PC = Program Counter — viitab järgmisena täidetavale käsule.

WiM arhitektuur

Magasin:



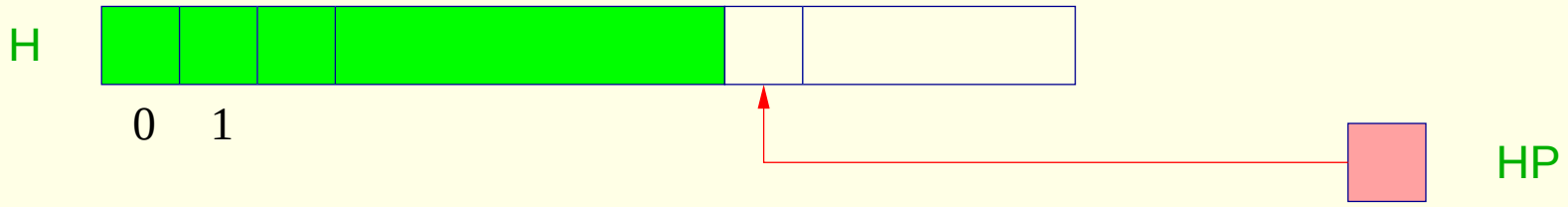
S = Stack — igas pesas võib olla kas baasväärtus või aadress;

SP = Stack-Pointer — viitab tipmisele täidetud pesale;

FP = Frame-Pointer — viitab kehtivale freimile.

WiM arhitektuur

Kuhi:



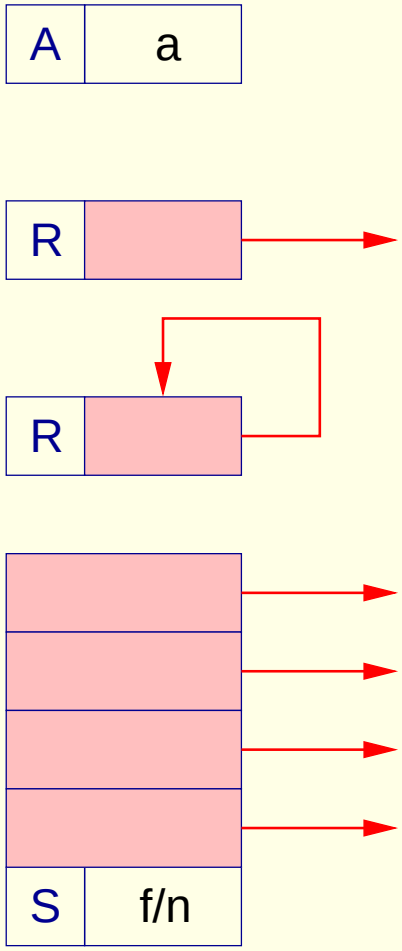
H = Heap — mälupiirkond dünaamiliste andmete hoidmiseks;

HP = Heap-Pointer — viitab esimesele vabale pesale.

- Käsk `new` loob kuhjas uue objekti.
- Objektid on märgendatud nende tüüpidega (nagu MaMa-s).

WiM arhitektuur

Kuhjas võivad olla järgmised objektid:



Atom	1 cell
Variable	1 cell
Unbound Variable	1 cell
Structure	n+1 cells

Termide konstrueerimine

Eesmärkide parameetertermid luuakse enne ülekandmist kuhja.

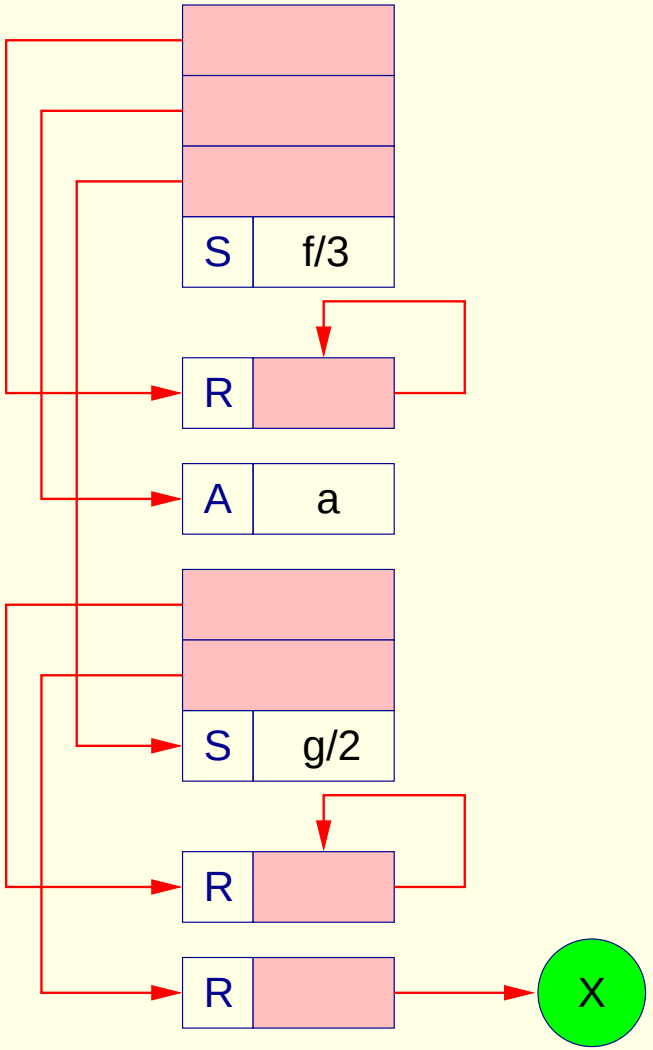
Aadesskeskond ρ seab igale muutujale X vastavusse tema aadressi magasinis (FP suhtes).

Termide konstrueerimine toimub funktsiooni $\text{code}_A\ t\ \rho$ abil, mis:

- loob kuhjas termi t esituse puu kujul;
- väljastab viida sellele puule magasini tipus.

Termide konstrueerimine

Näide: Termi $t \equiv f(g(X,Y),a,Z)$ esitamine, kus X on juba initsialiseerud muutuja ja Y ning Z on veel initsialiseerimata:



Termide konstrueerimine

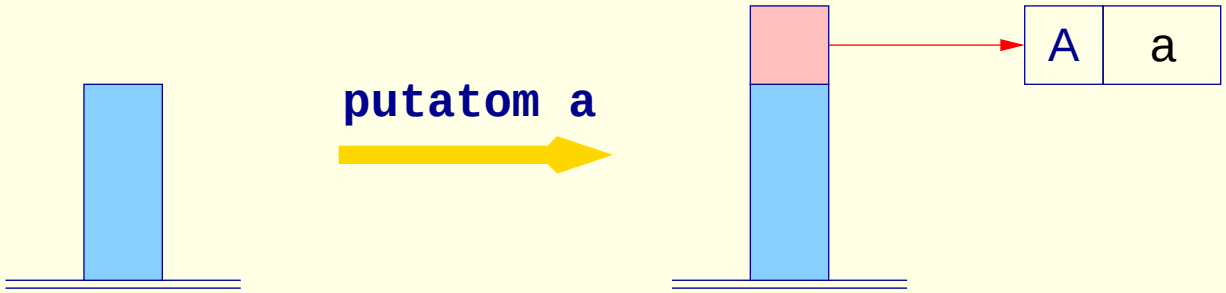
$$
\begin{array}{lcl}
\text{code}_A\ a\ \rho & = & \texttt{putatom a} \\
\text{code}_A\ X\ \rho & = & \texttt{putvar}\ (\rho\ X) \\
\text{code}_A\ \overline{X}\ \rho & = & \texttt{putref}\ (\rho\ X) \\
\text{code}_A\ \_\ \rho & = & \texttt{putanon}
\end{array}
\qquad
\begin{array}{lcl}
\text{code}_A\ f(t_1,\ldots,t_n)\ \rho & = & \text{code}_A\ t_1\ \rho \\
 & & \ldots \\
 & & \text{code}_A\ t_n\ \rho \\
 & & \texttt{putstruct f/n}
\end{array}
$$

kus X on initsialiseerimata ning $\overline{X}$ initsialiseeritud muutuja.

Näide: olgu $t \equiv f(g(X,Y),a,Z)$ ja $\rho = \{X \mapsto 1,\ Y \mapsto 2,\ Z \mapsto 3\}$, siis $\text{code}_A\ t\ \rho$ emiteerib koodi:

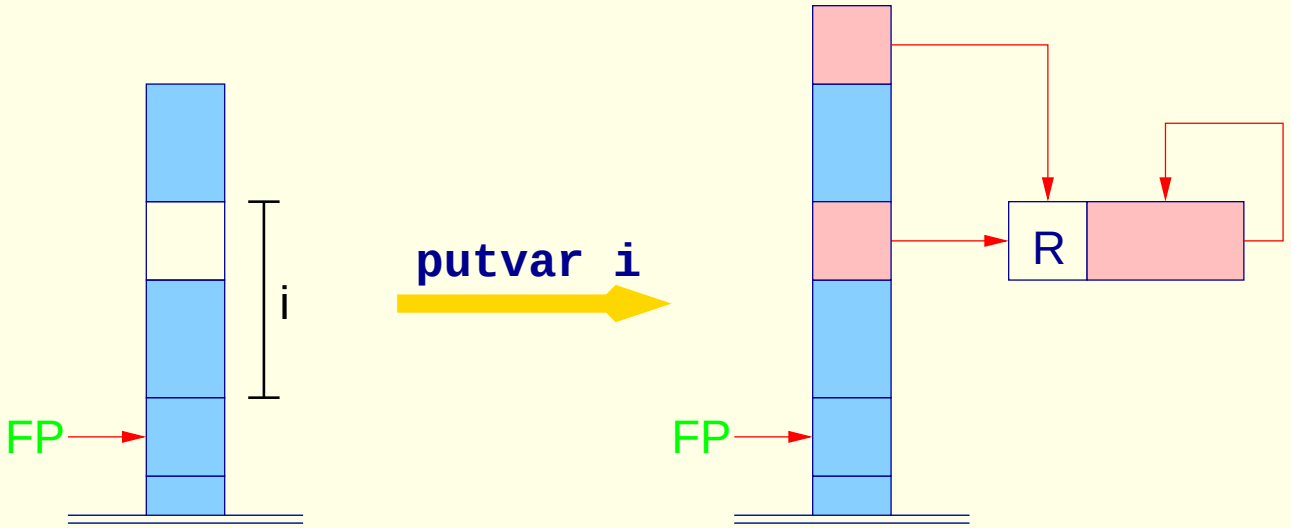
```
putref 1          putatom a
putvar 2          putvar 3
putstruct g/2     putstruct f/3
```

Termide konstrueerimine



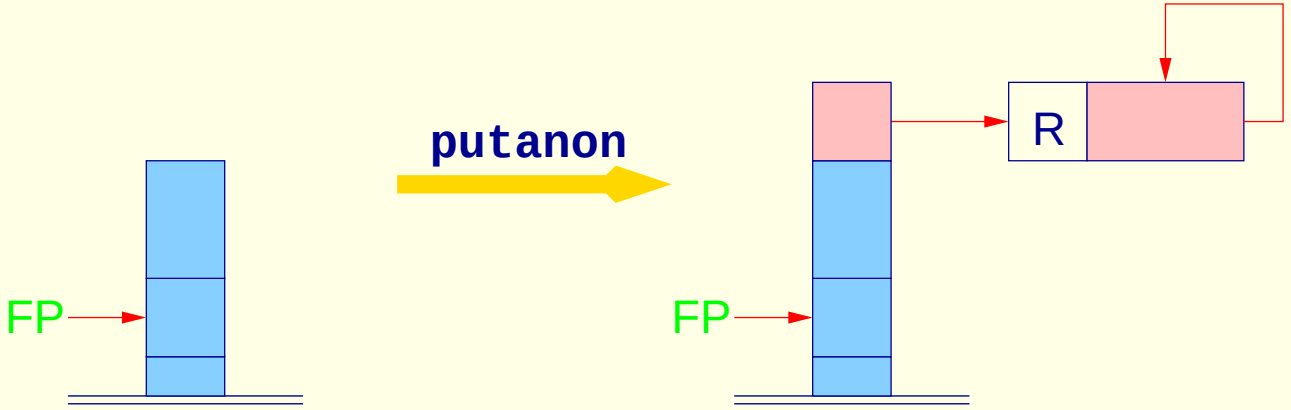
SP++;
S[SP] = new (A,a);

Termide konstrueerimine



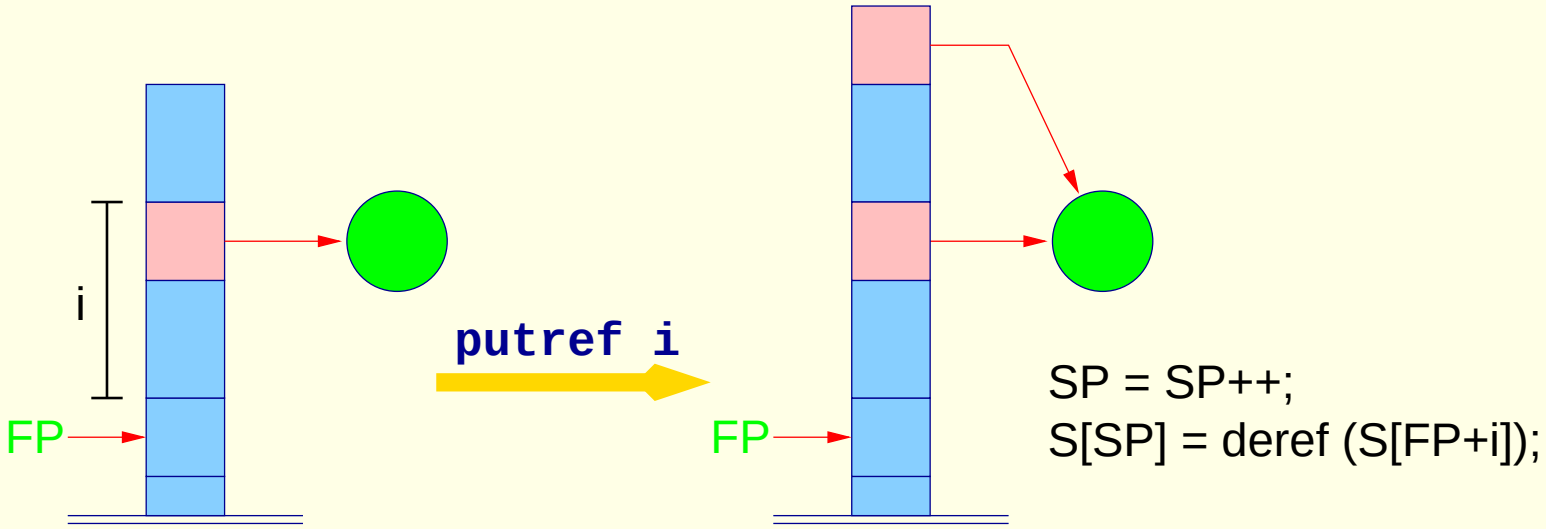
SP = SP++;
S[SP] = new (R,HP);
S[FP+i] = S[SP];

Termide konstrueerimine



SP = SP++;
S[SP] = new (R,HP);

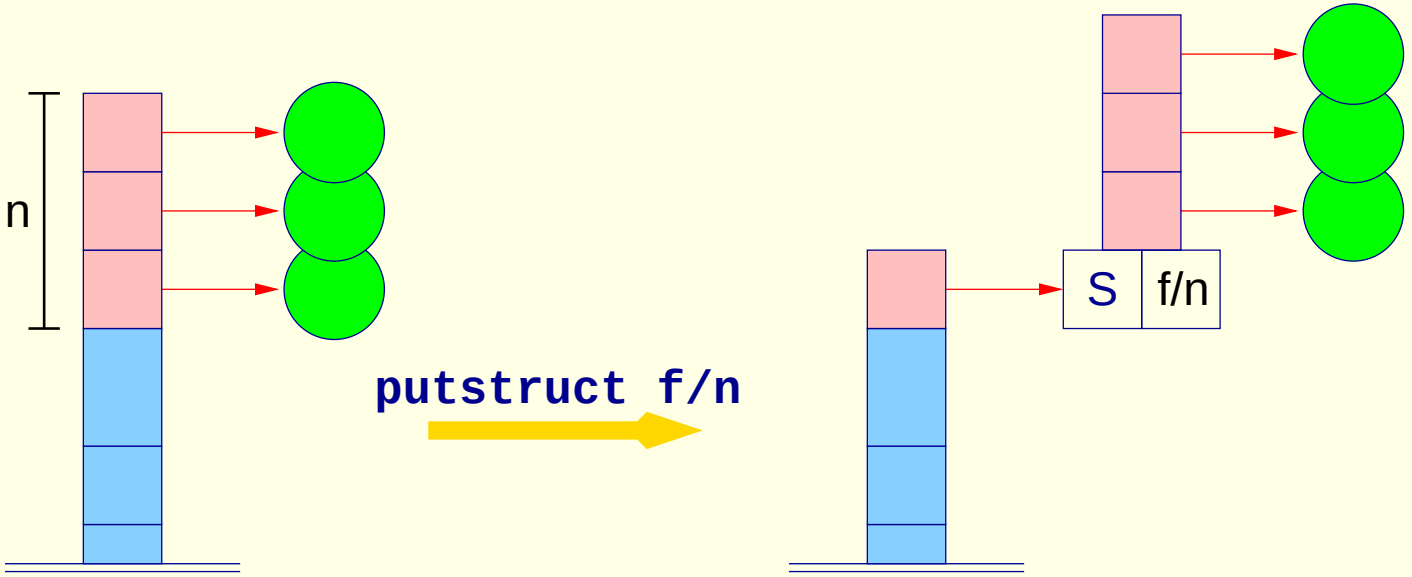
Termide konstrueerimine



Abifunktsioon `deref` normaliseerib viitade ahelaid:

```
ref deref (ref v) {
  if (H[v] == (R,w) && v != w) return deref (w);
  else return v;
}
```

Termide konstrueerimine




```
v = new (S,f,n);
SP = SP − n + 1;
for (i=1; i≤n; i++)
   H[v+i] = S[SP+i-1];
S[SP] = v;
```

Termide konstrueerimine

Märkused:

- Käsk `pushref i` mitte ainult ei kopeeri viita aadressilt `S[FP+i]`, vaid ka eemaldab viitade ahelad niipalju kui võimalik.
- Termide konstrueerimisel viitavad viidad alati väiksematele kuhja aadressitele. Olgugi, sama kehtib ka paljudel muudel juhtudel, pole üldjuhul see garanteeritud.

Eesmärkide transleerimine

- Eesmärgid vastavad protseduuride väljakutsetele.
- Eesmärkide transleerimine toimub funktsiooni code_G abil.
- Kõigepealt luuakse magasinis freim.
- Seejärel konstrueeritakse aktuaalsed parameetrid (kuhjas)
- . . . ja salvestatakse viidad neile magasini.
- Lõpuks hüpatakse predikaadile vastava koodi algusse.

Eesmärkide transleerimine

$\text{code}_G\ p(t_1,\ldots,t_k)\ \rho$ =

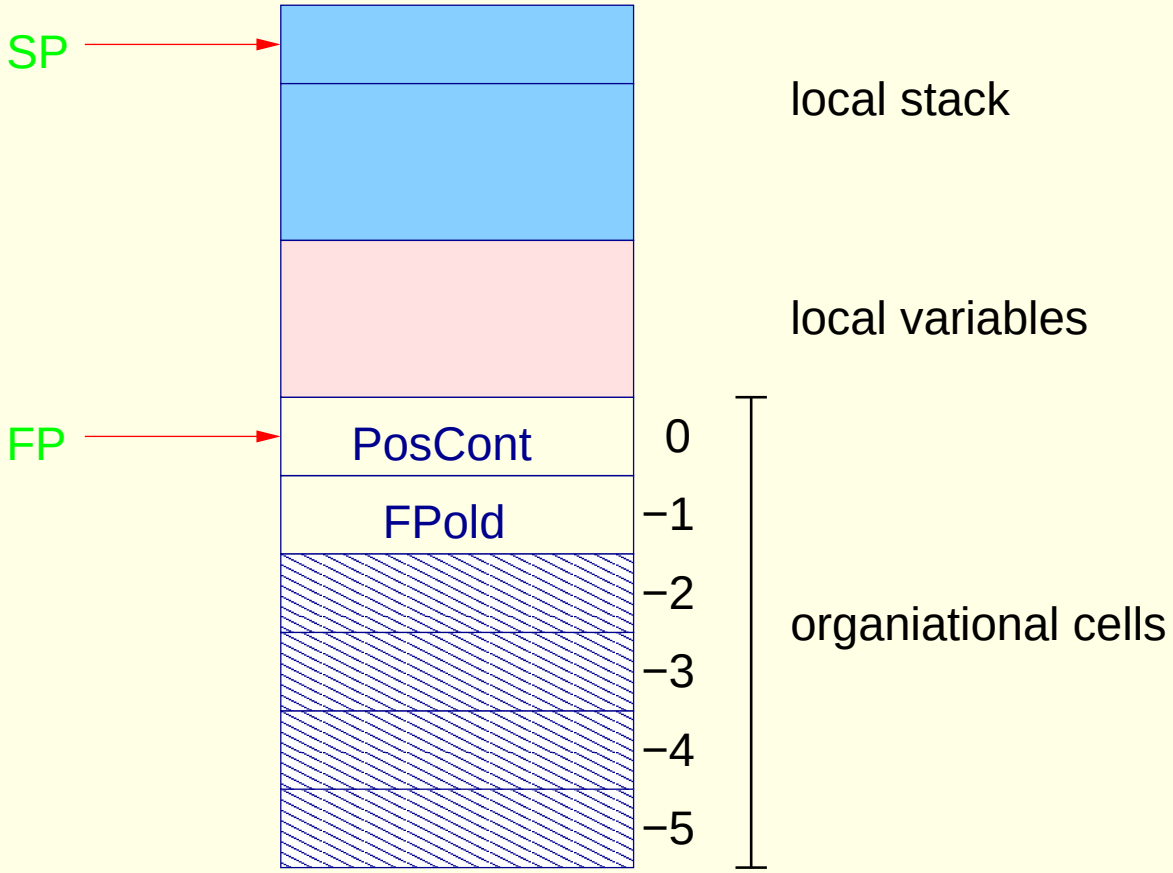
```
    mark A
```
$\text{code}_A\ t_1\ \rho$
```
    ...
```
$\text{code}_A\ t_k\ \rho$
```
    call p/k
A:  ...
```

Näide: olgu $g \equiv p(a, X, g(\bar{X}, Y))$ ja $\rho = \{X \mapsto 1,\ Y \mapsto 2\}$,
siis $\text{code}_G\ g\ \rho$ emiteerib koodi:

```
mark B          putref 1            call p/3
putatom a       putvar 2        B:  ...
putvar 1        putstruct g/2
```

Eesmärkide transleerimine

Freimi struktuur:

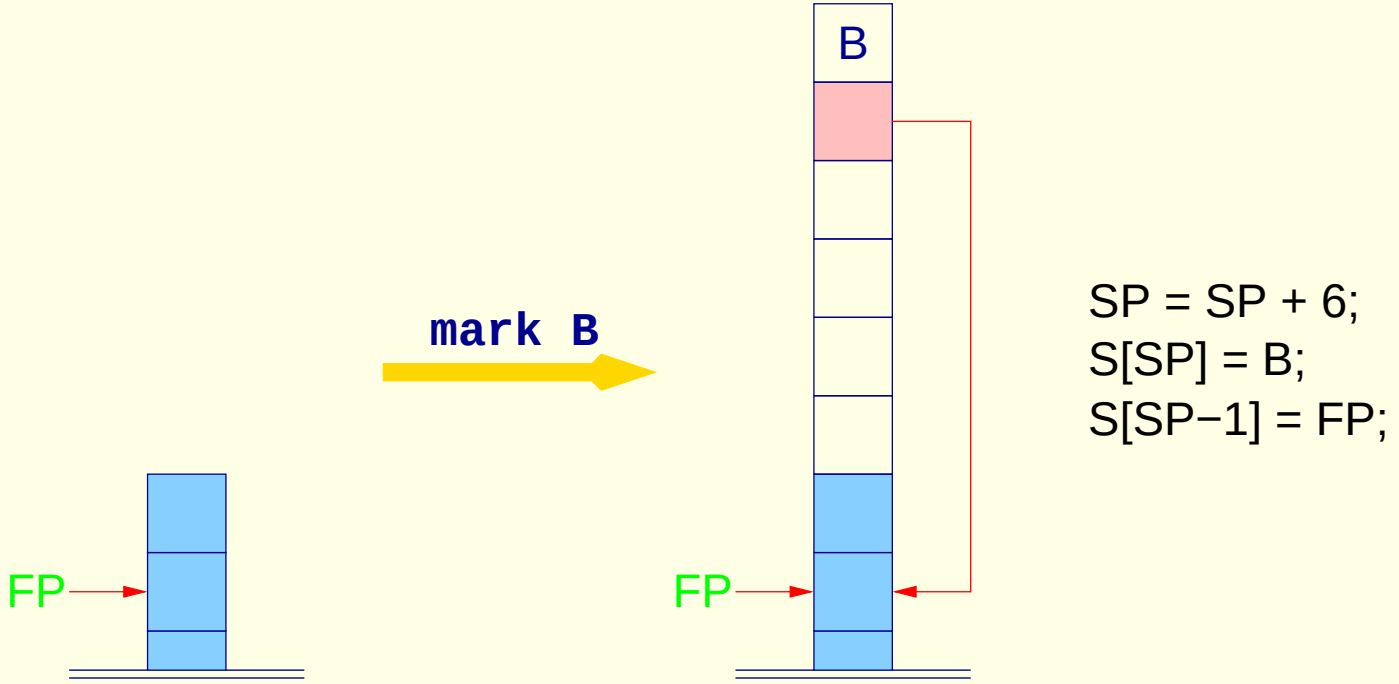


Eesmärkide transleerimine

Märkused:

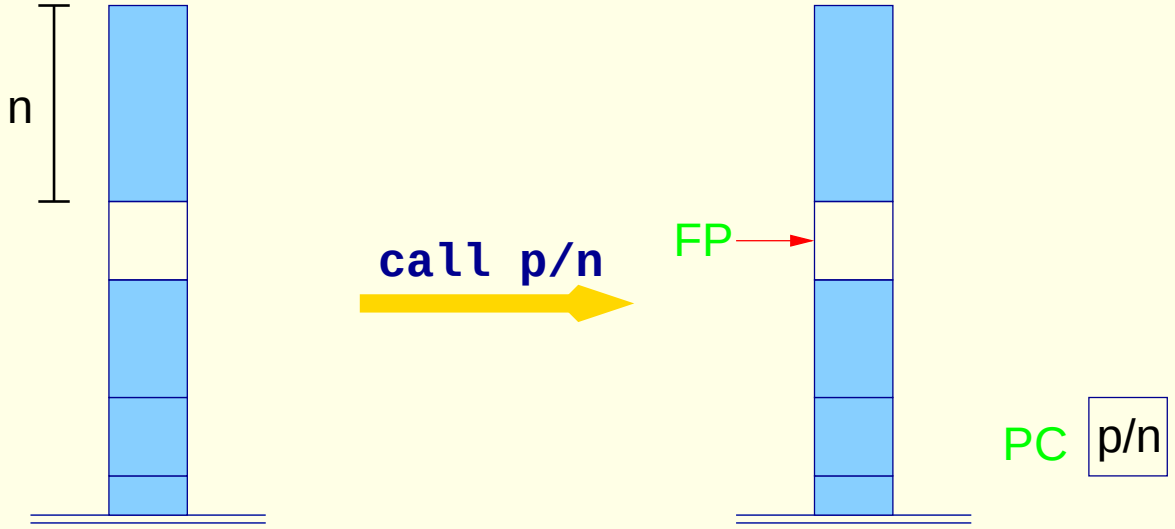
- *Positiivse jätku* aadress `PosCont` viitab koodile, kust jätkata kui eesmärgi täitmine oli edukas.
- Ülejäänud organisatoorsed pesad on vajalikud ebaõnnestumisel *tagasipöördumiseks* (backtracking).

Eesmärkide transleerimine



SP = SP + 6;
S[SP] = B;
S[SP−1] = FP;

Eesmärkide transleerimine



FP = SP − n;
PC = p/n;