WiM — loogiliste keelte lihtsustatud 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, ananüümne 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:

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

Loogiline keel Proll

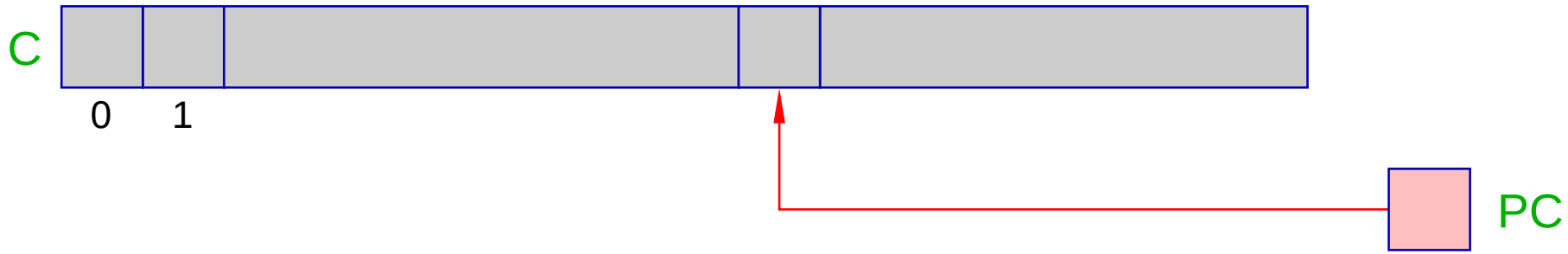
Näide:

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

- $[\,]$ 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:

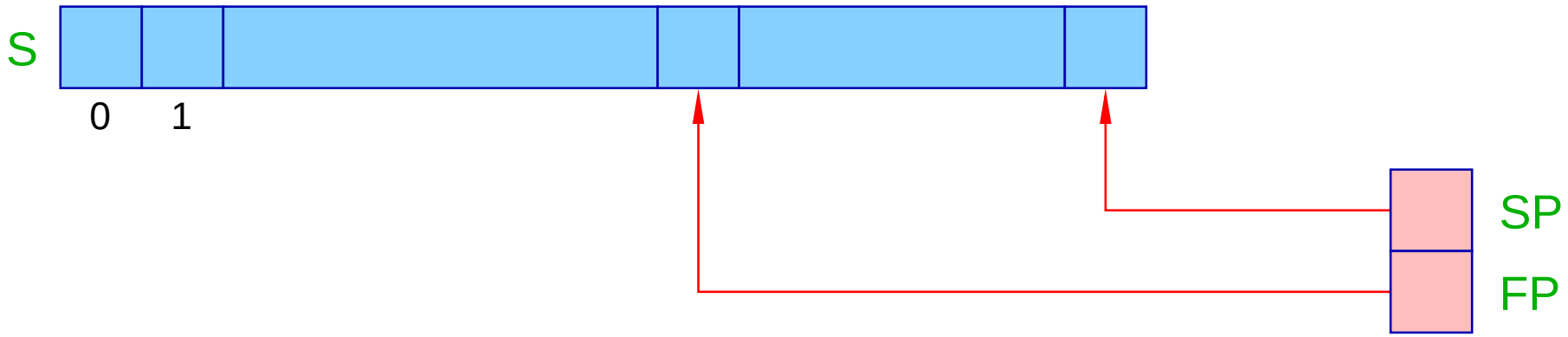


C = Code-store — mälupiirkond WiM programmikoodi hoidmiseks; igas pesas on täpselt üks abstraktse masina käsk.

PC = Program Counter — viitab järgmisena täidetavale käsule.

WiM arhitektuur

Magasin:



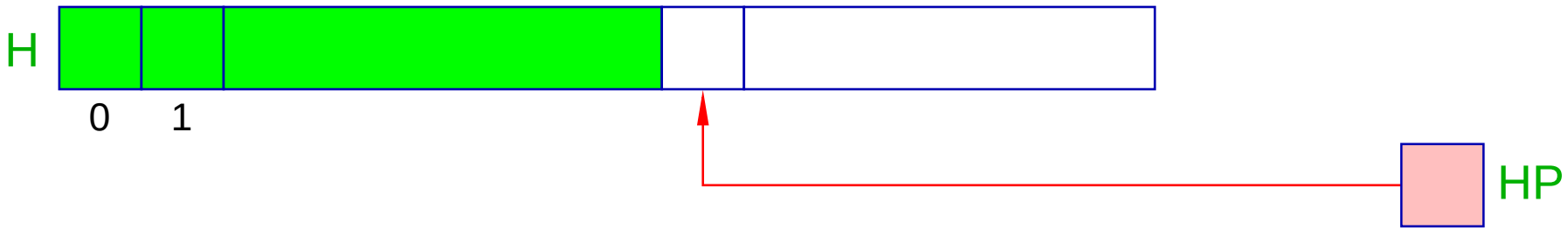
S = Stack — igas pesas võib olla kas baasväärtus või aadress;

SP = Stack-Pointer — viitab tipmisele täidetud pesale;

FP = Frame-Pointer — viitab kehtivale freimile.

WiM arhitektuur

Kuhi:



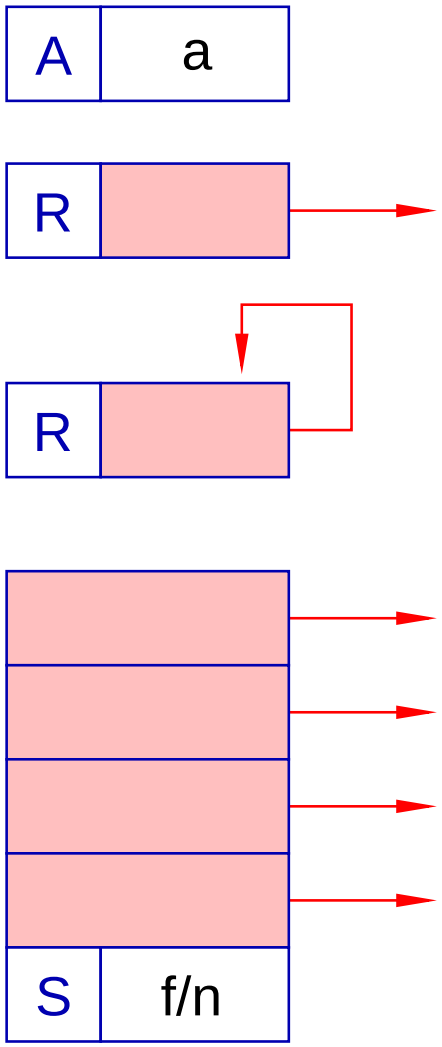
H = Heap — mälupiirkond dünaamiliste andmete hoidmiseks;

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.

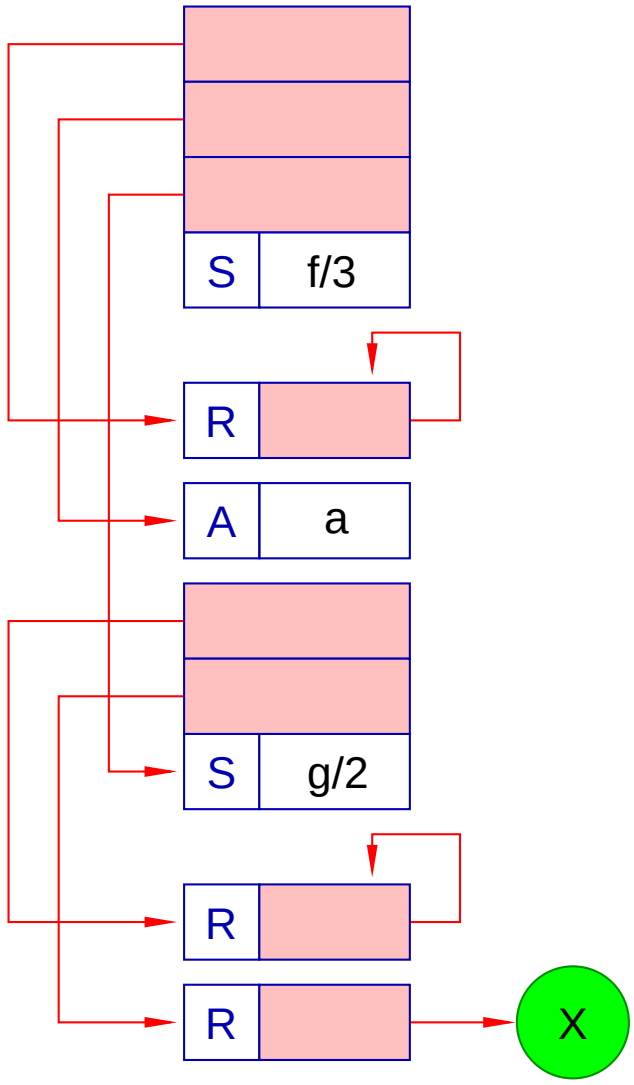
Aadresskeskkond ρ seab igale muutujale X vastavusse aadressi magasinis (FP suhtes).

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 initsialiseeritud muutuja ja Y ning Z on veel initsialiseerimata.



Termide konstrueerimine

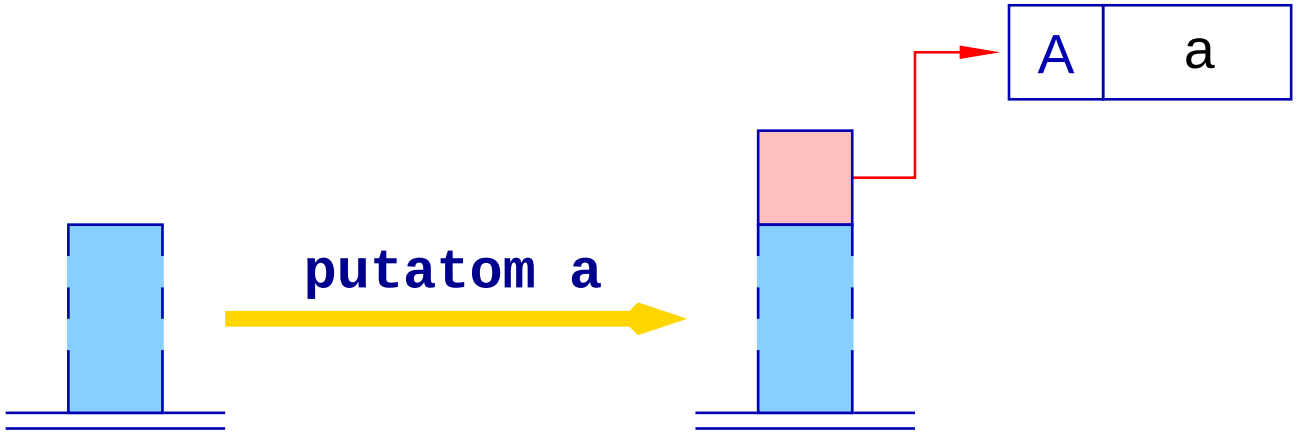
$$
\begin{array}{lcl}
\text{code}_A\ a\ \rho & = & \texttt{putatom a} \\
\text{code}_A\ X\ \rho & = & \texttt{putvar}\ (\rho\ X) \\
\text{code}_A\ \bar{X}\ \rho & = & \texttt{putref}\ (\rho\ X) \\
\text{code}_A\ \_\ \rho & = & \texttt{putanon}
\end{array}
\qquad
\begin{array}{lcl}
\text{code}_A\ f(t_1,\ldots,t_n)\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

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

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

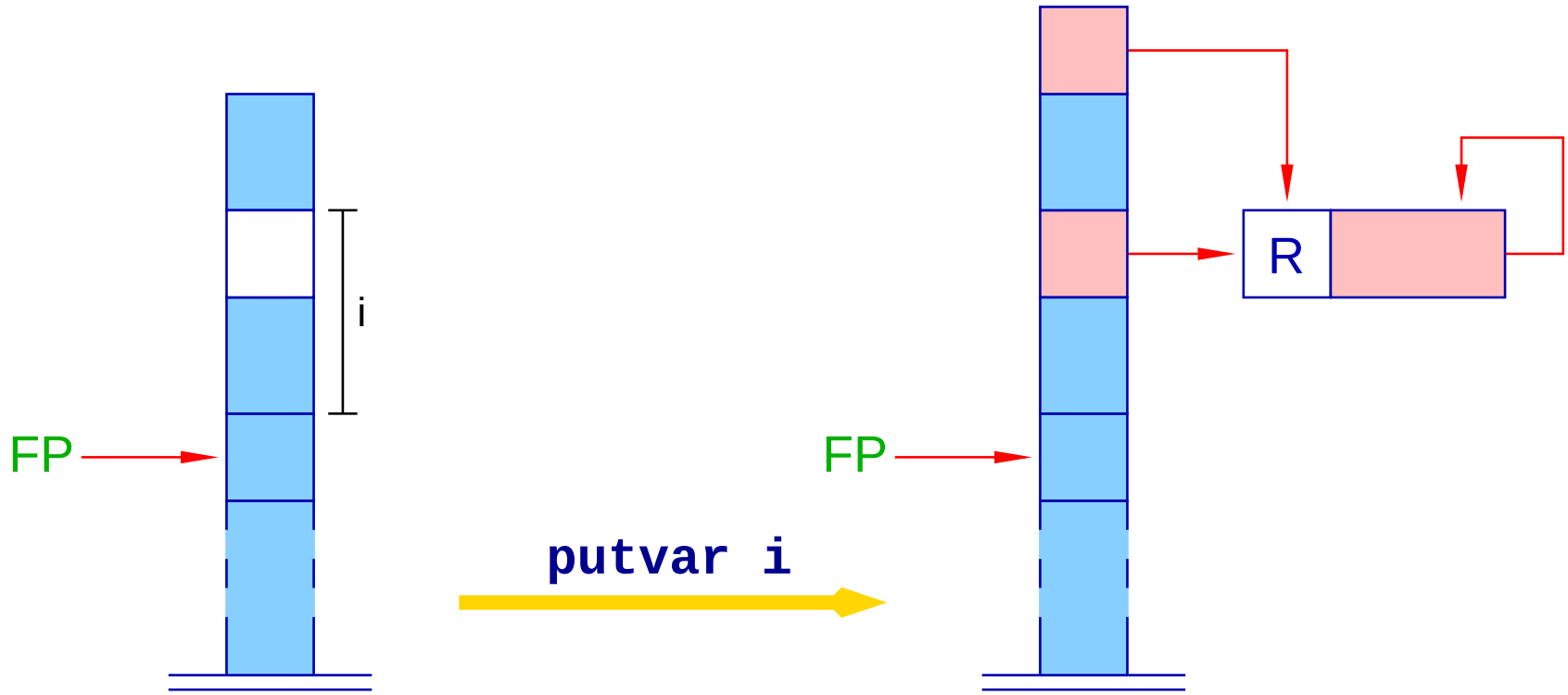
```
putref 1          putatom a
putvar 2          putvar 3
putstruct g/2     putstruct f/3
```

Termide konstrueerimine



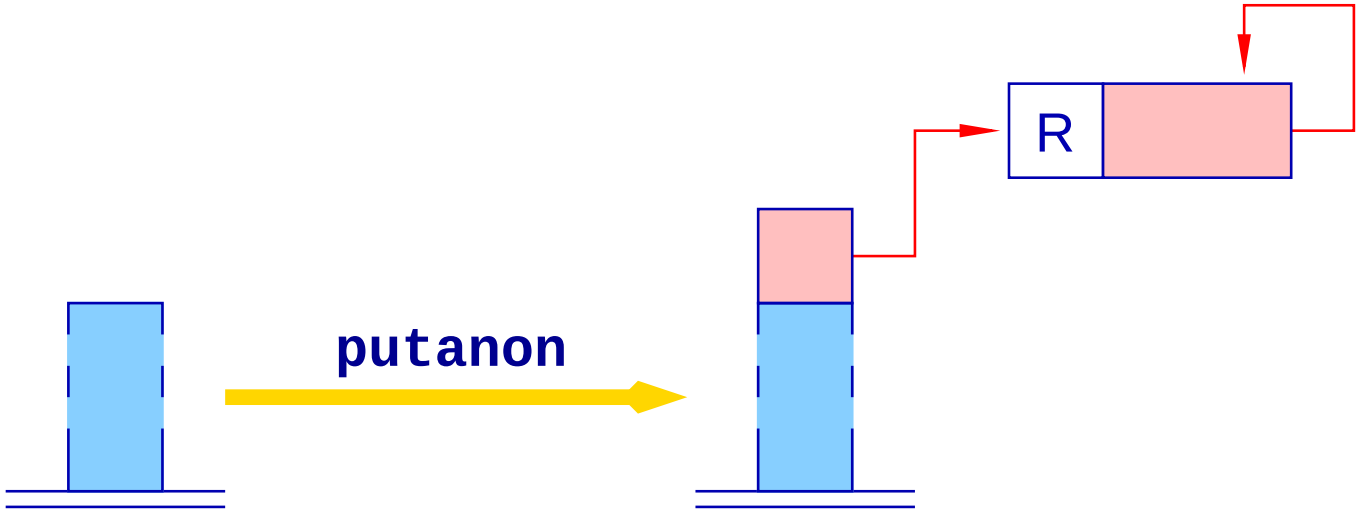

```
SP++;
S[SP] = new (A,a);
```

Termide konstrueerimine



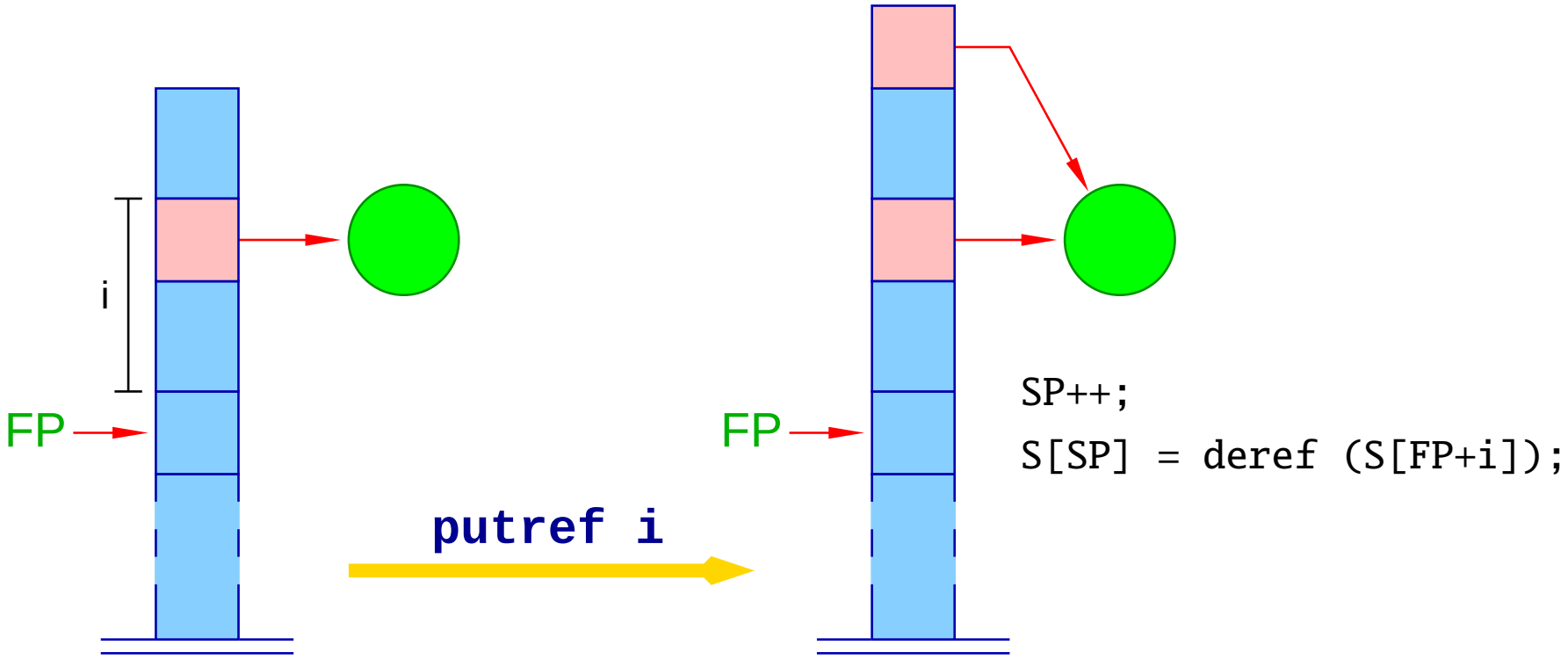

```
SP++;
S[SP] = new (R,HP);
S[FP+i] = S[SP];
```

Termide konstrueerimine




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

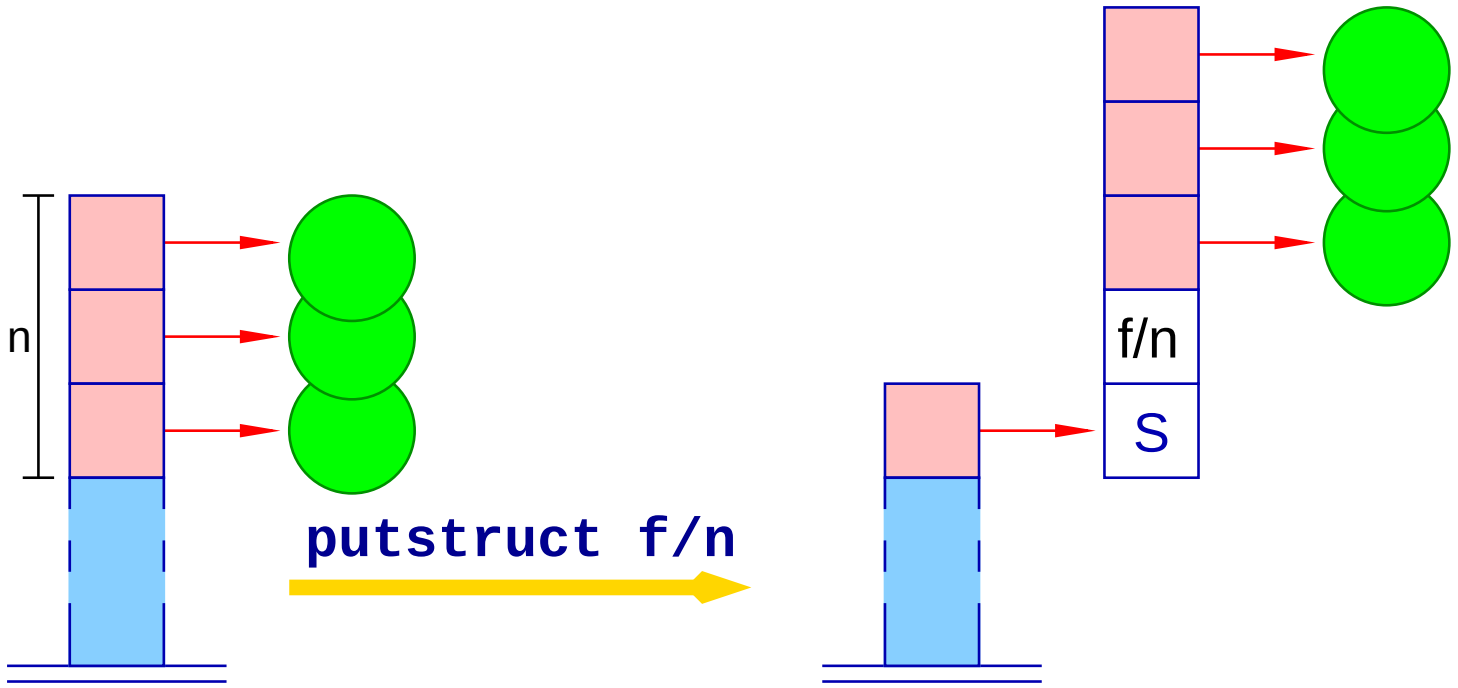
Termide konstrueerimine



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] = SP[SP+i-1];
S[SP] = v;
```

Termide konstrueerimine

Märkused:

- Käsk `putref 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, et sama kehtib ka paljudel muudel juhtudel, pole see üldjuhul 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

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

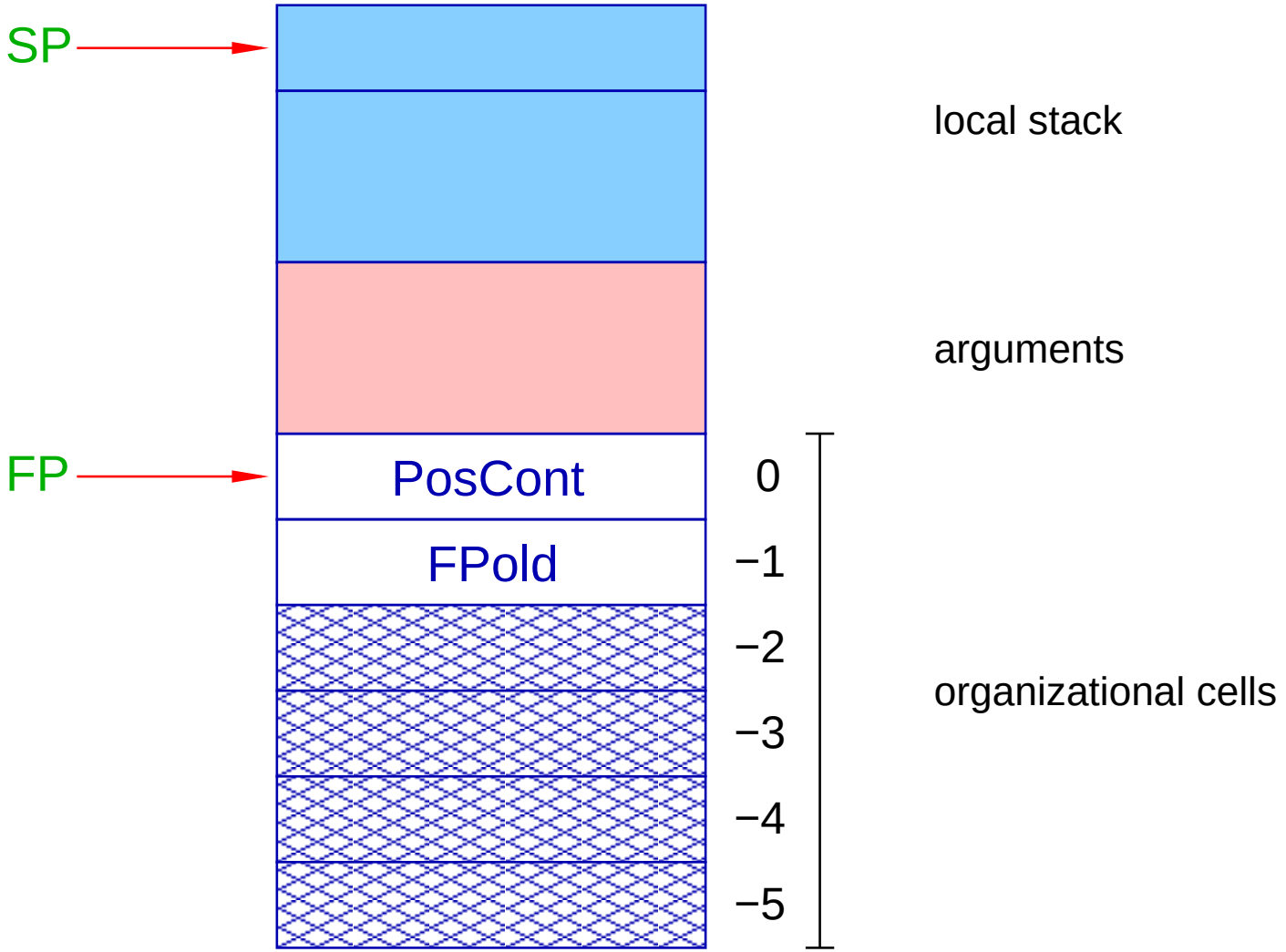
```
    mark A
    code_A t_1 ρ
    ...
    code_A t_k ρ
    call p/k
A:  ...
```

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

Eesmärkide transleerimine

Freimi struktuur:

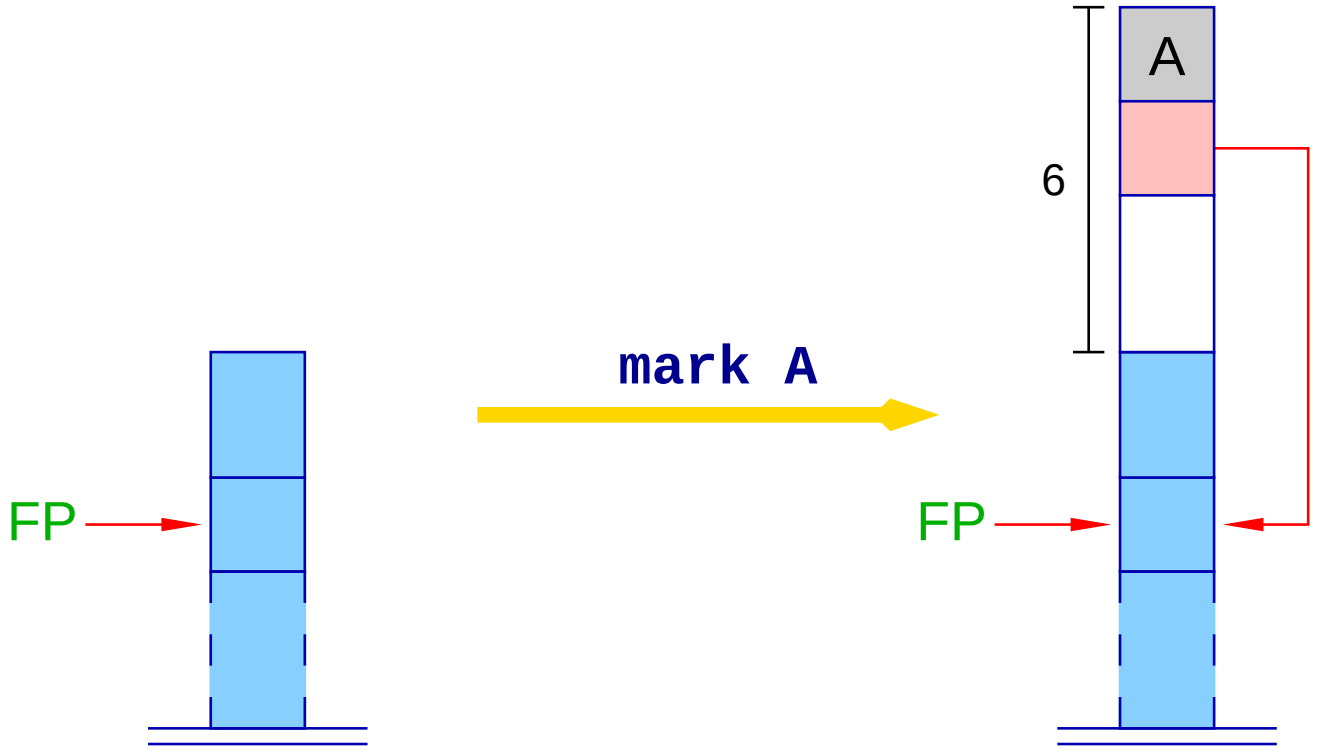


Eesmärkide transleerimine

Märkused:

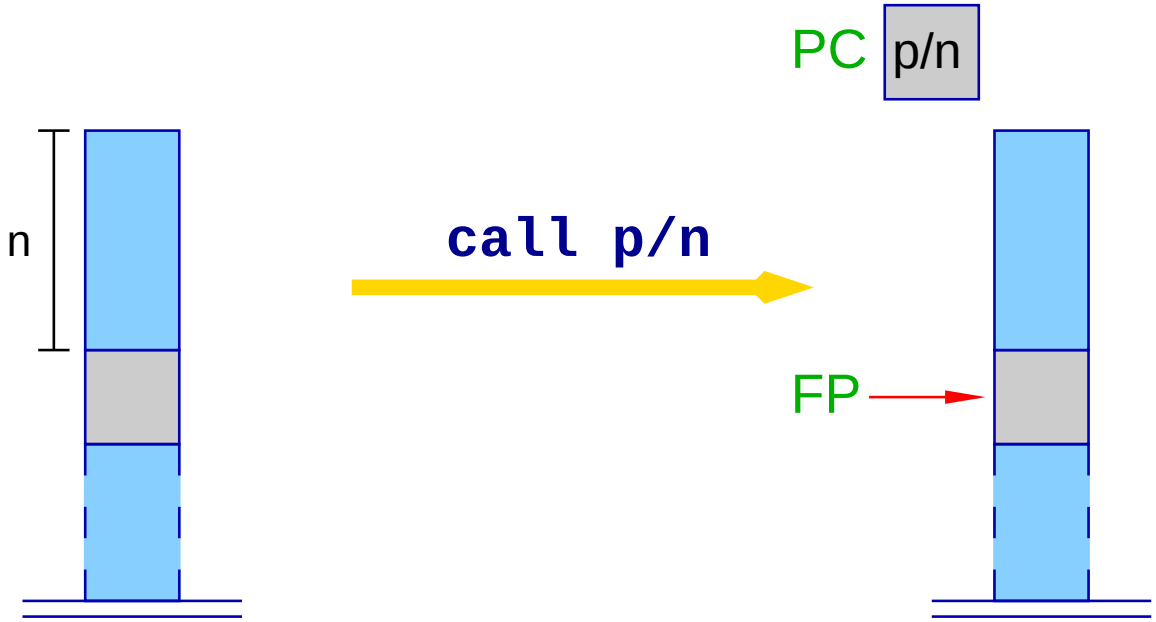
- Positiivse jätku aadress `PosCont` viitab koodile, kust jätkata kui eesmärgi täitmine oli edukas.
- Ülejäänud organisatoorsed pesad on vajalikud ebaõnnestumisel tagasipöördumiseks (backtracking).

Eesmärkide transleerimine




```
SP = SP + 6;
S[SP] = A;
S[SP-1] = FP;
```

Eesmärkide transleerimine




```
FP = SP - n;
PC = p/n;
```

Unifitseerimine

- Muutuja X esinemisi tähistame $\tilde{X}$.
- Transleeritakse erinevalt sõltuvalt sellest, kas on initsialiseeritud või mitte.
- Kasutame makrot put $\tilde{X}$ ρ:

$$\begin{aligned} \text{put } X\ \rho &= \texttt{putvar}\ (\rho X) \\ \text{put } \bar{X}\ \rho &= \texttt{putref}\ (\rho X) \\ \text{put } \_\ \rho &= \texttt{putanon} \end{aligned}$$

Unifitseerimine

Unifitseerimise $\tilde{X} = t$ transleerimine:

- lisame magasini viida muutujale X;
- konstrueerime kuhja termi t esituse;
- toome sisse uue käsu, mis realiseerib unifitseerimise.

$$\begin{array}{lcl} \text{code}_G\ (\tilde{X} = t)\ \rho & = & \text{put}\ \tilde{X}\ \rho \\ & & \text{code}_A\ t\ \rho \\ & & \texttt{unify} \end{array}$$

Unifitseerimine

Näide: olgu antud võrrand

$$\bar{U} = f(g(\bar{X}, Y), a, Z)$$

Siis keskkonna

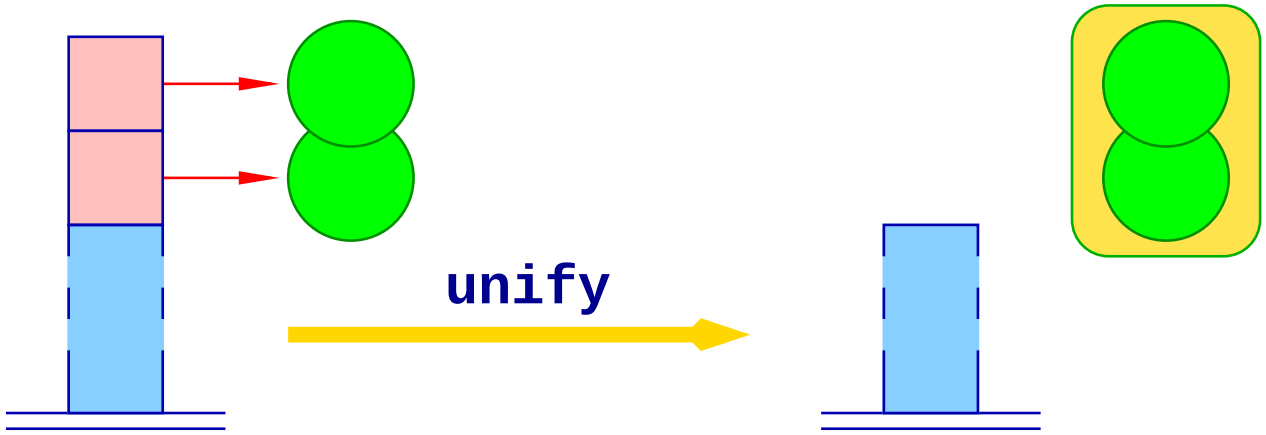
$$\rho = \{X \mapsto 1,\ Y \mapsto 2,\ Z \mapsto 3,\ U \mapsto 4\}$$

korral emiteeritakse kood:

```
putref 4          putatom a
putref 1          putvar 3
putvar 2          putstruct f/3
putstruct g/2     unify
```

Unifitseerimine

Käsk `unify` rakendab täitmisaegset funktsiooni `unify()` kahele ülemisele viidale:




```
unify (S[SP-1], S[SP-2]);
SP = SP - 2;
```

Unifitseerimine

Funktsioon `unify()`

- ... võtab kaks kuhja aadressit. Iga väljakutse jaoks garanteerime, et viidaahelad oleksid elimineeritud.
- ... kontrollib, kas need aadressid on juba identsed. Sel juhul ei tee midagi ja unifitseerimine õnnestus.
- ... seob nooremad muutujad (suuremad aadressid) vanemate muutujatega (väiksemate aadressitega).
- ... muutuja sidumisel termiga kontrollib, kas muutuja esineb termis või mitte (occur-check).
- ... salvestab loodud seosed.
- ... võib ebaõnnestude (fail), misjuhul toimub tagasipöördumine (backtracking).

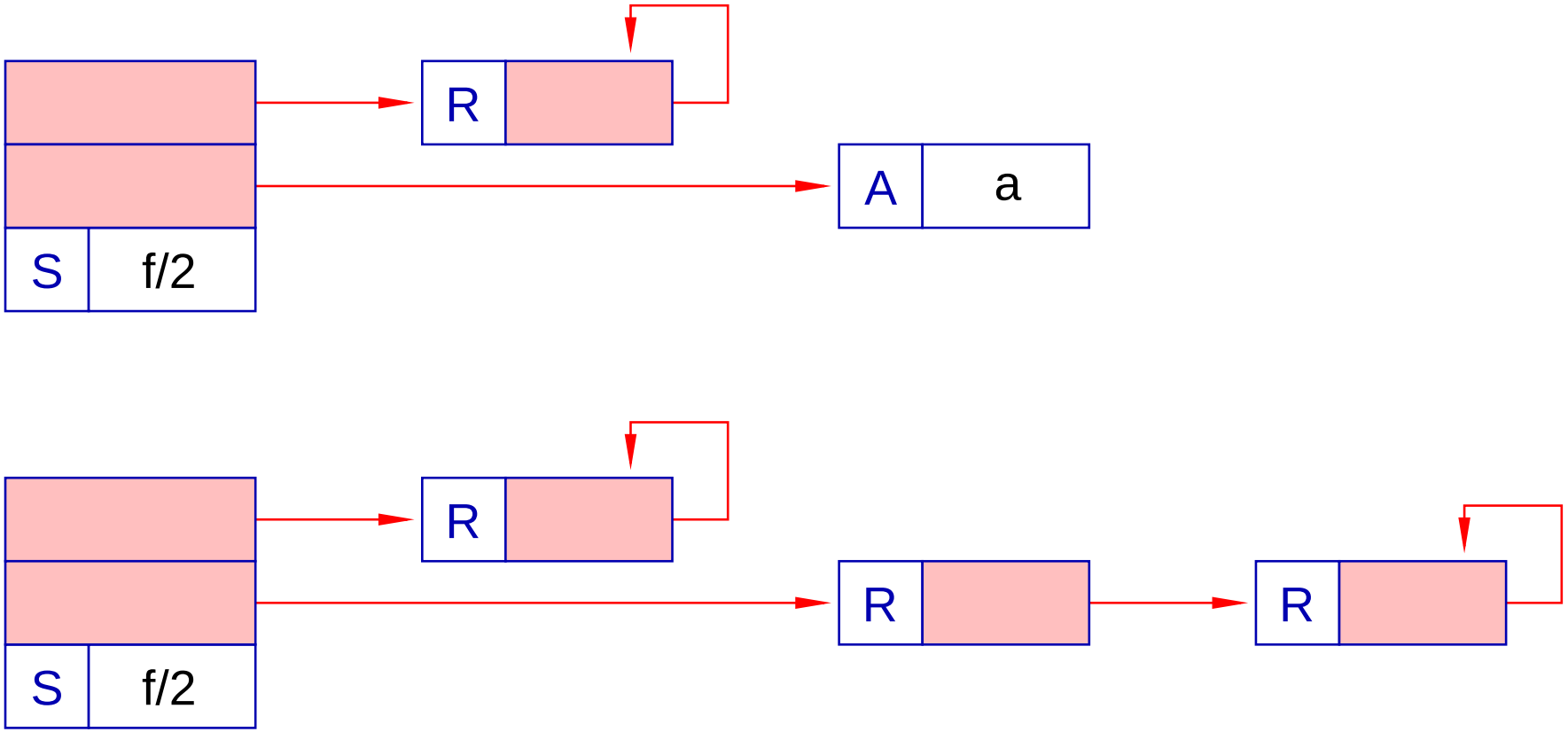
Unifitseerimine

```
bool unify (ref u, ref v) {
  if (u == v) return true;
  if (H[u] == (R,_)) {
    if (H[v] == (R,_)) {
      if (u > v) {
        H[u] = (R,v); trail(u); return true;
      } else {
        H[v] = (R,u); trail(v); return true;
      }
    } else if (check (u,v)) {
      H[u] = (R,v); trail(u); return true;
    } else { backtrack(); return false; }
  }
...
```

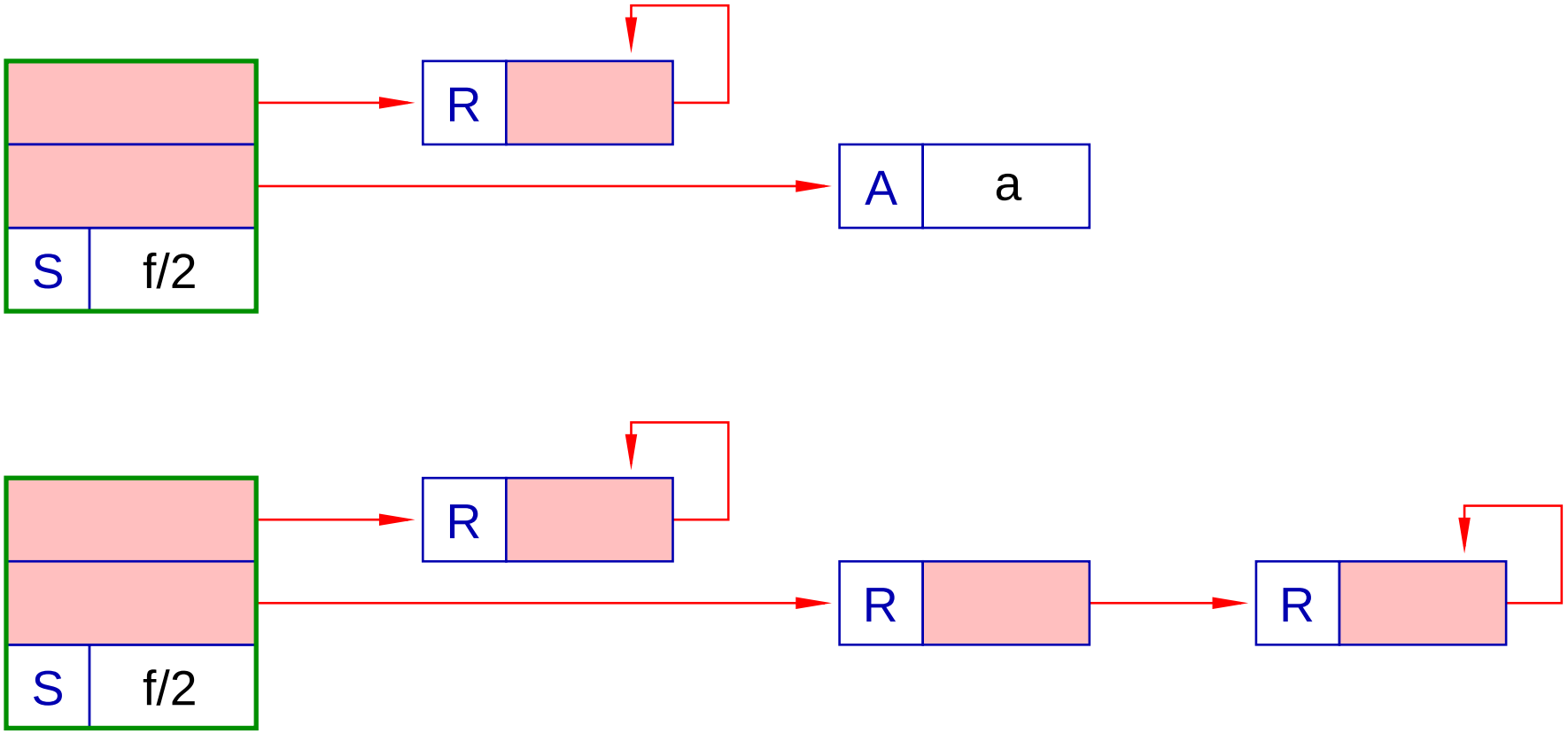
Unifitseerimine

```
...
  if (H[v] == (R,_)) {
    if (check (v,u)) {
      H[v] = (R,u); trail(v); return true;
    } else { backtrack(); return false; }
  }
  if (H[u] == (A,a) && H[v] == (A,a)) return true;
  if (H[u] == (S,f/n) && H[v] == (S,f/n)) {
    for (int i=1; i<=n; i++)
      if (!unify (deref(H[u+i]), deref(H[v+i]))) return false;
    return true;
  }
  backtrack(); return false;
}
```

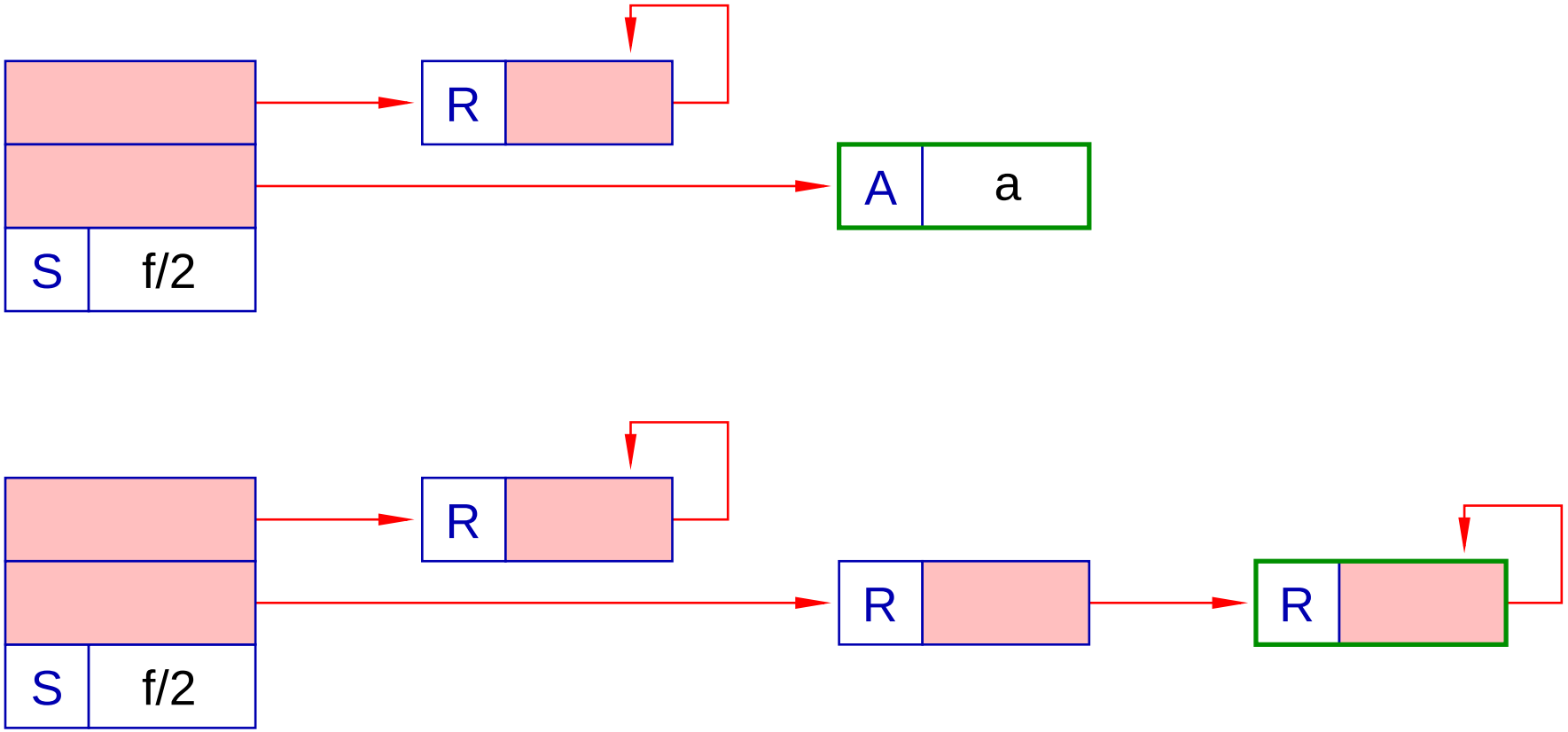
Unifitseerimine



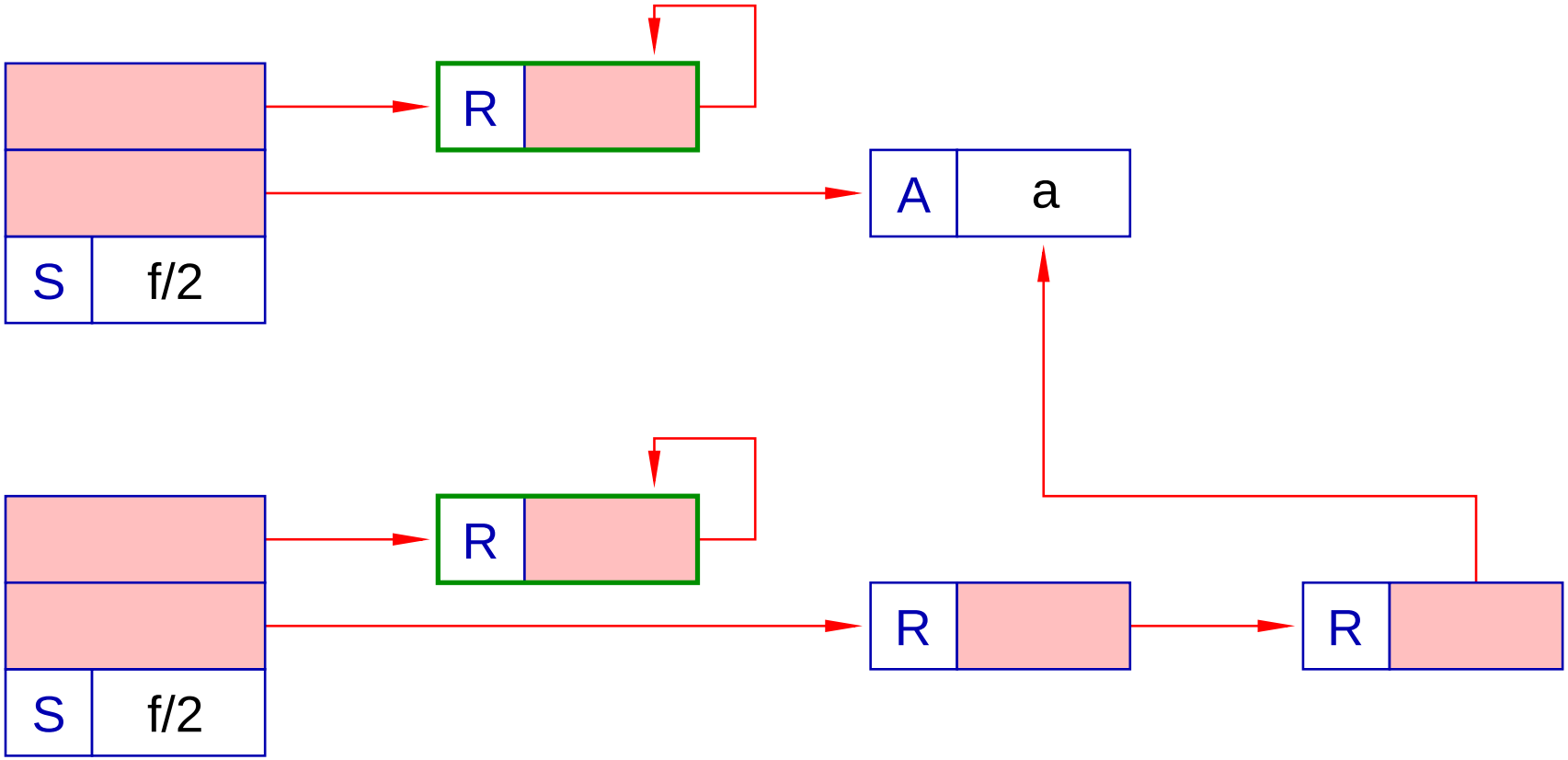
Unifitseerimine



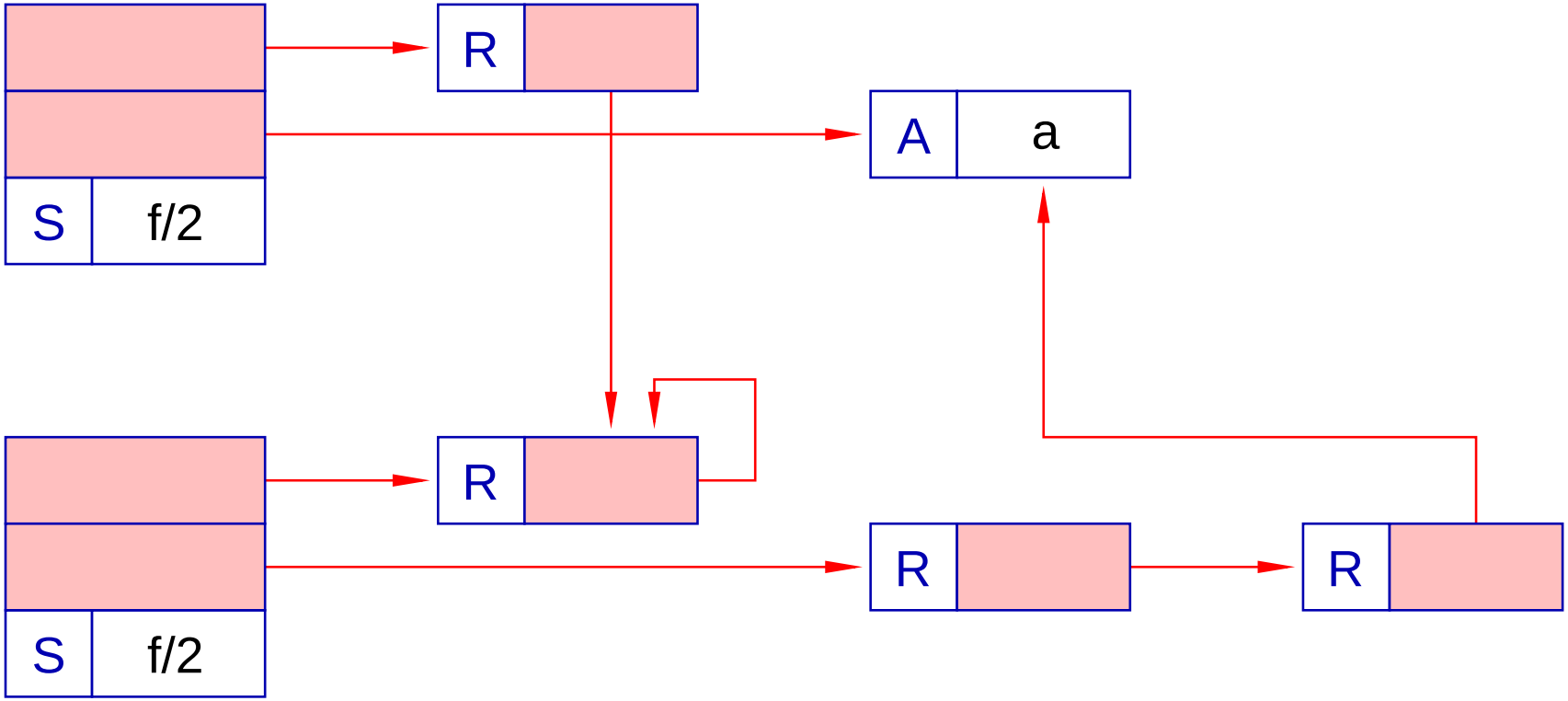
Unifitseerimine



Unifitseerimine



Unifitseerimine



Unifitseerimine

- Funktsioon `trail()` salvestab uued seosed.
- Funktsioon `backtrack()` teostab tagasipöördumise.
- Funktsioon `check()` kontrollib, kas muutuja (tema esimene argument) esineb termis (tema teine argument).
- Tihti seda kontrolli ei teostata:

```
bool check (ref u, ref v) {
    return true;
}
```

Unifitseerimine

Kui soovime kontrolli teostada, siis saame funktsiooni `check()` realiseerida järgnevalt:

```
bool check (ref u, ref v) {
  if (u == v) return false;
  if (H[v] == (S,f/n))
    for (int i=1; i<=n; i++)
      if (!check (u, deref (H[v+i])))
        return false;
  return true;
}
```

Unifitseerimine

- Võrrandi $\tilde{X} = t$ transleerimine on väga lihtne,
- aga tihti muutuvad konstrueeritud objektid koheselt prügiks.
- Idee:
  - Lisame muutujale $\tilde{X}$ vastava seose magasini.
  - Väldime termi t alamtermide konstrueerimist kui võimalik.
  - Selle asemel transleerime iga t tipu käsuks, mis teostab selle tipuga unifitseerimise!

$$\begin{aligned} \text{code}_G\ (\tilde{X} = t)\ \rho \quad &= \quad \text{put}\ \tilde{X}\ \rho \\ &\qquad\quad \text{code}_U\ t\ \rho \end{aligned}$$

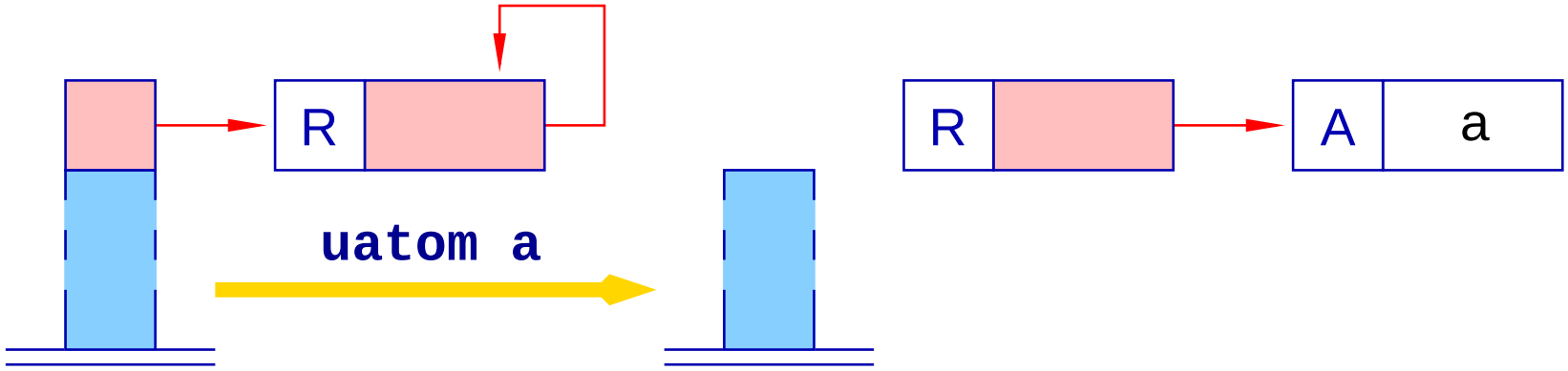
Unifitseerimine

Lihtsate termidega unifitseerimine:

$$
\begin{array}{lcl}
\text{code}_U\ a\ \rho & = & \texttt{uatom a} \\
\text{code}_U\ X\ \rho & = & \texttt{uvar}\ (\rho X) \\
\text{code}_U\ \bar{X}\ \rho & = & \texttt{uref}\ (\rho X) \\
\text{code}_U\ \_\ \rho & = & \texttt{pop}
\end{array}
$$

Unifitseerimine

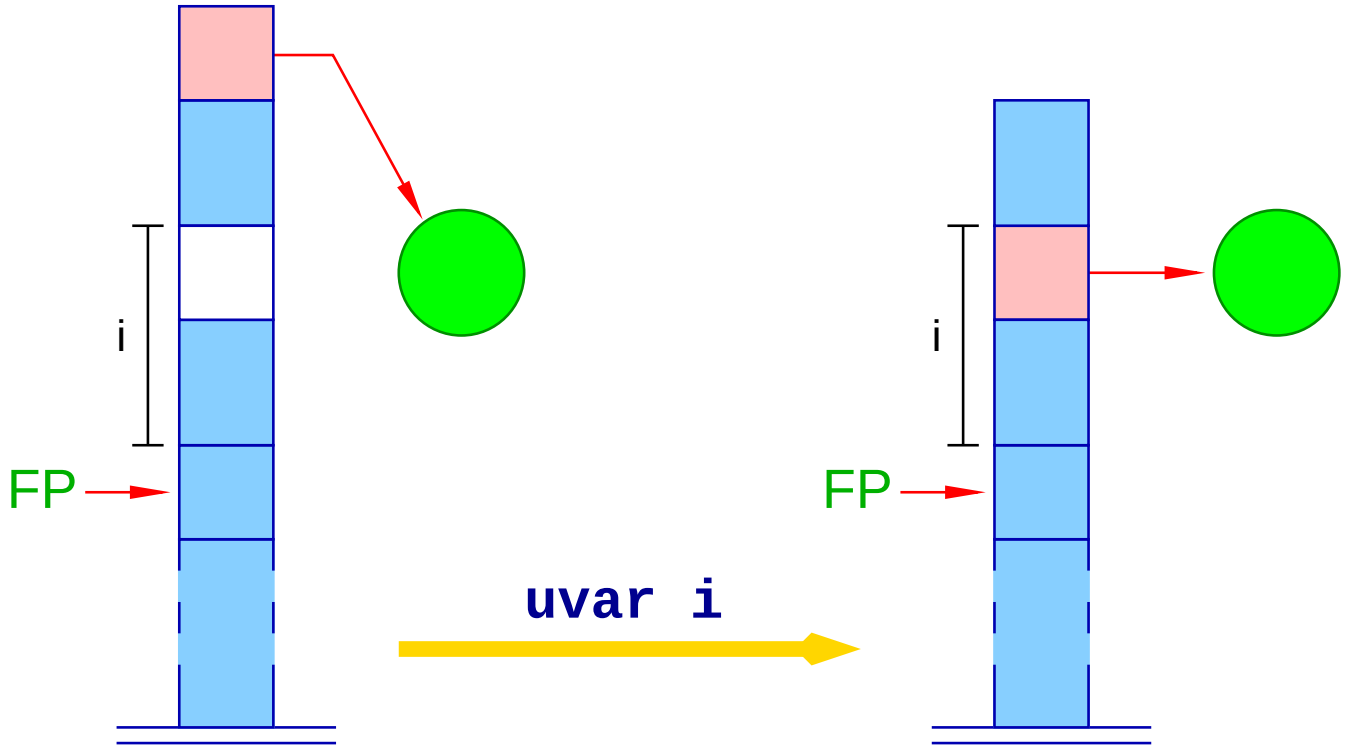
Käsk `uatom a` realiseerib unifitseerimise aatomiga:




```
v = S[SP]; SP--;
switch (H[v]) {
  case (A,a): break;
  case (R,_): H[v] = (R, new(A,a));
              trail(v); break;
  default:    backtrack();
}
```

Unifitseerimine

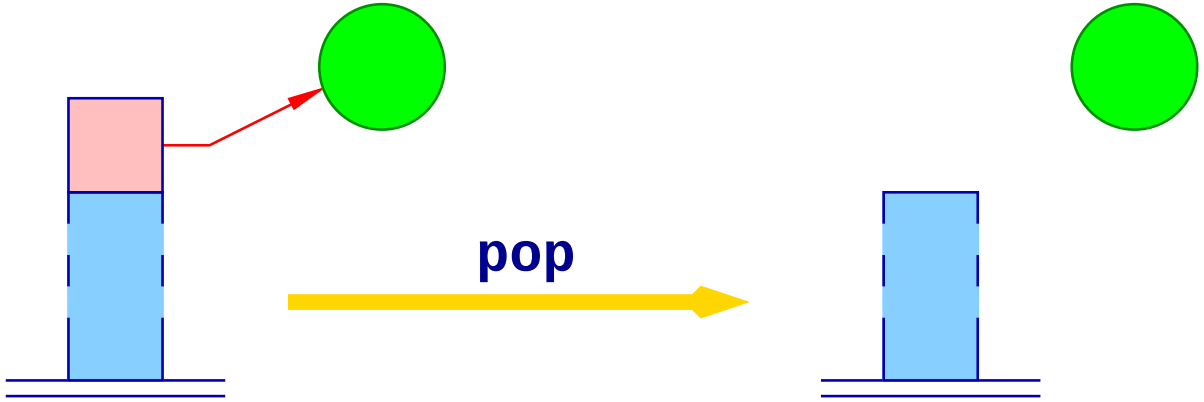
Käsk `uvar i` realiseerib unifitseerimise initsialiseerimata muutujaga:




```
S[FP+i] = S[SP];
SP--;
```

Unifitseerimine

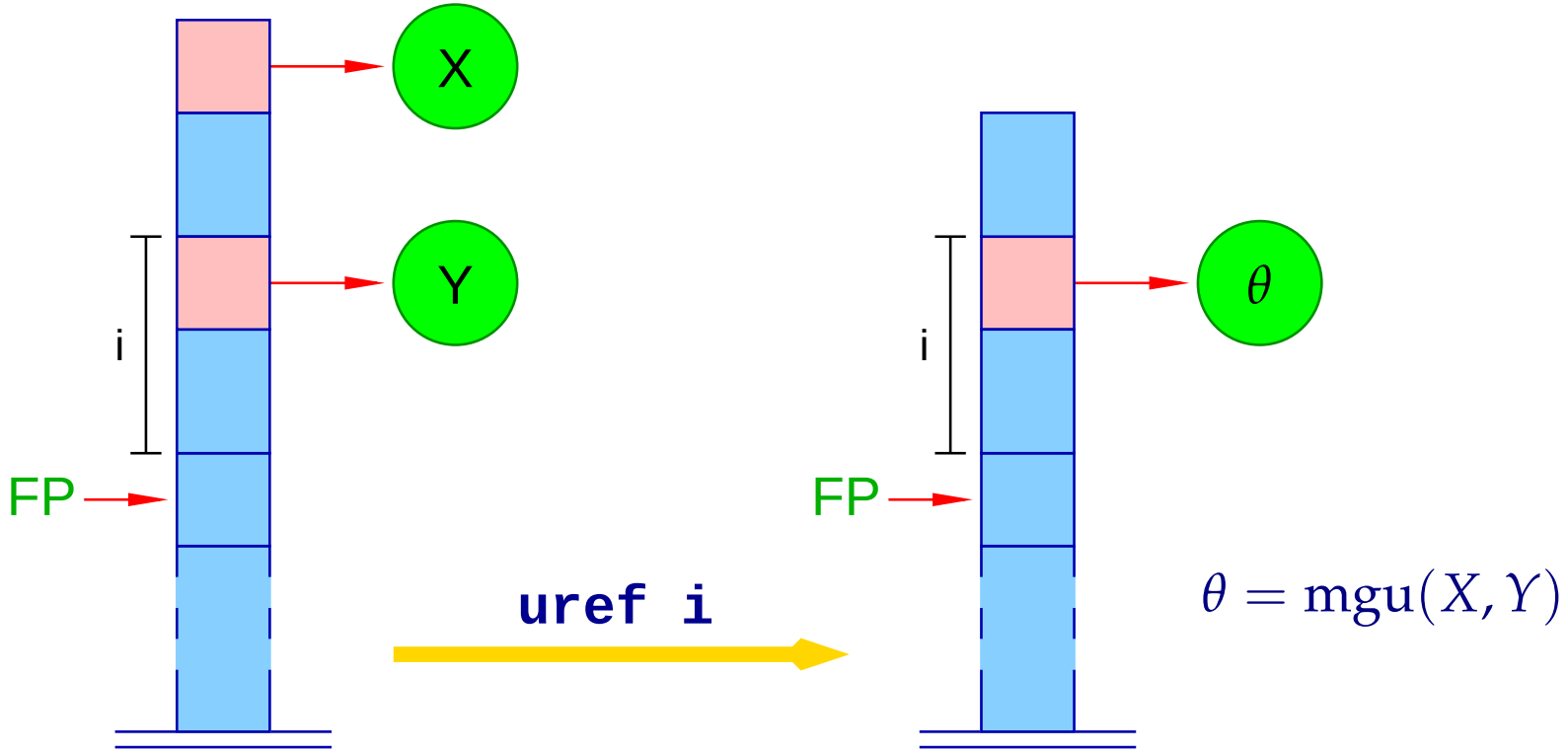
Käsk pop realiseerib unifitseerimise anonüümse muutujaga:




```
SP--;
```

Unifitseerimine

Käsk `uref i` realiseerib unifitseerimise initsialiseeritud muutujaga:




```
unify (S[SP], deref(S[FP+i]));
SP--;
```

Ainus koht, kus täitmisaegset funktsiooni `unify()` kasutatakse!

Unifitseerimine

Konstruktoraplikatsioonide unifitseerimine:

- Termiga t unifitseerimine toimub termi läbimisel eeljärjekorras.
- Kõigepealt kontrollime, kas tipmine viit on unifitseeruv term.
- Kui ta on sama konstruktorsümboli rakendus, siis rekursiivselt kontrollime alamterme.
- Initsialiseerimata muutuja korral lülitume kontrollimiselt ümber termi konstrueerimisele.

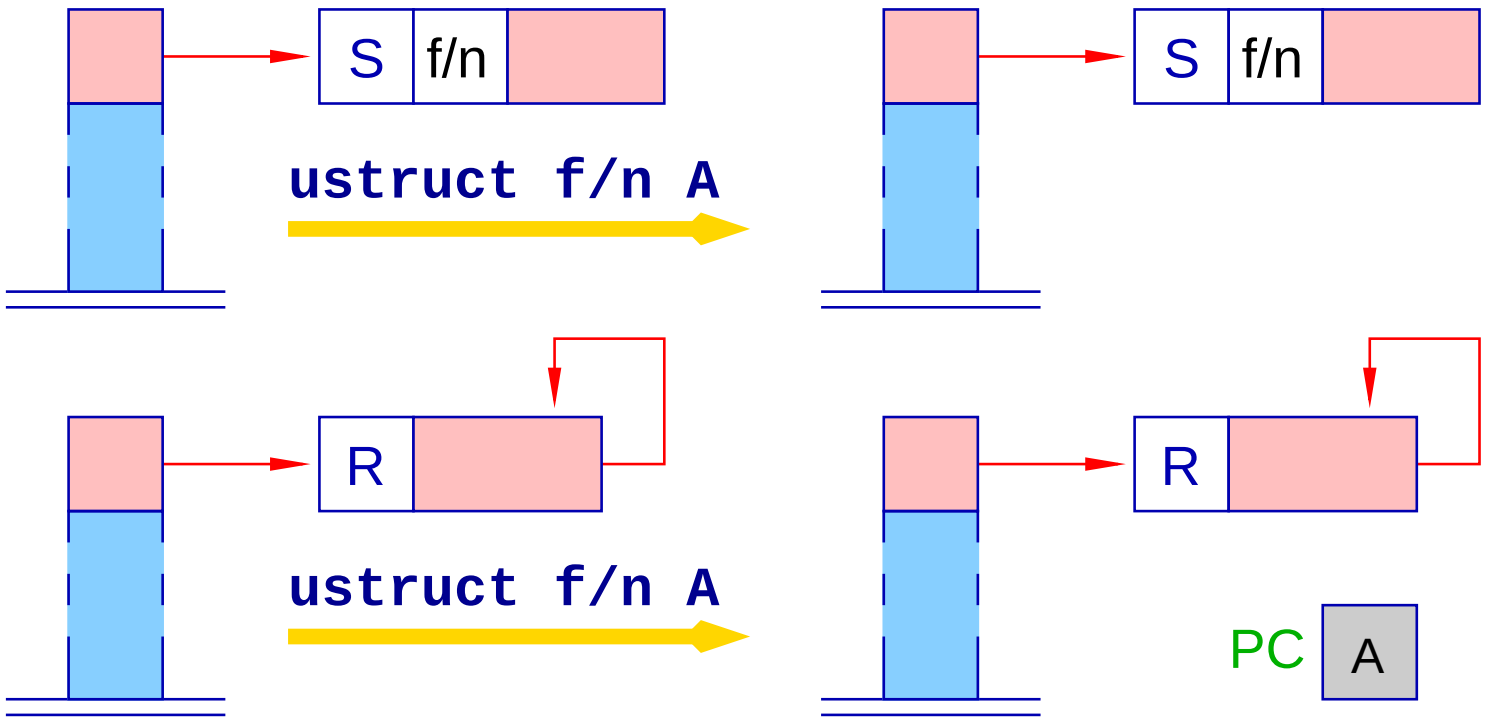
Unifitseerimine

Konstruktoraplikatsioonide unifitseerimine:

$\text{code}_U\ (f(t_1, \ldots, t_n))\ \rho$ =

```
    ustruct f/n A
    son 1
    code_U t_1 ρ
    ...
    son n
    code_U t_n ρ
    up B
A:  check ivars(f(t_1, ..., t_n)) ρ
    code_A (f(t_1, ..., t_n)) ρ
    bind
B:  ...
```

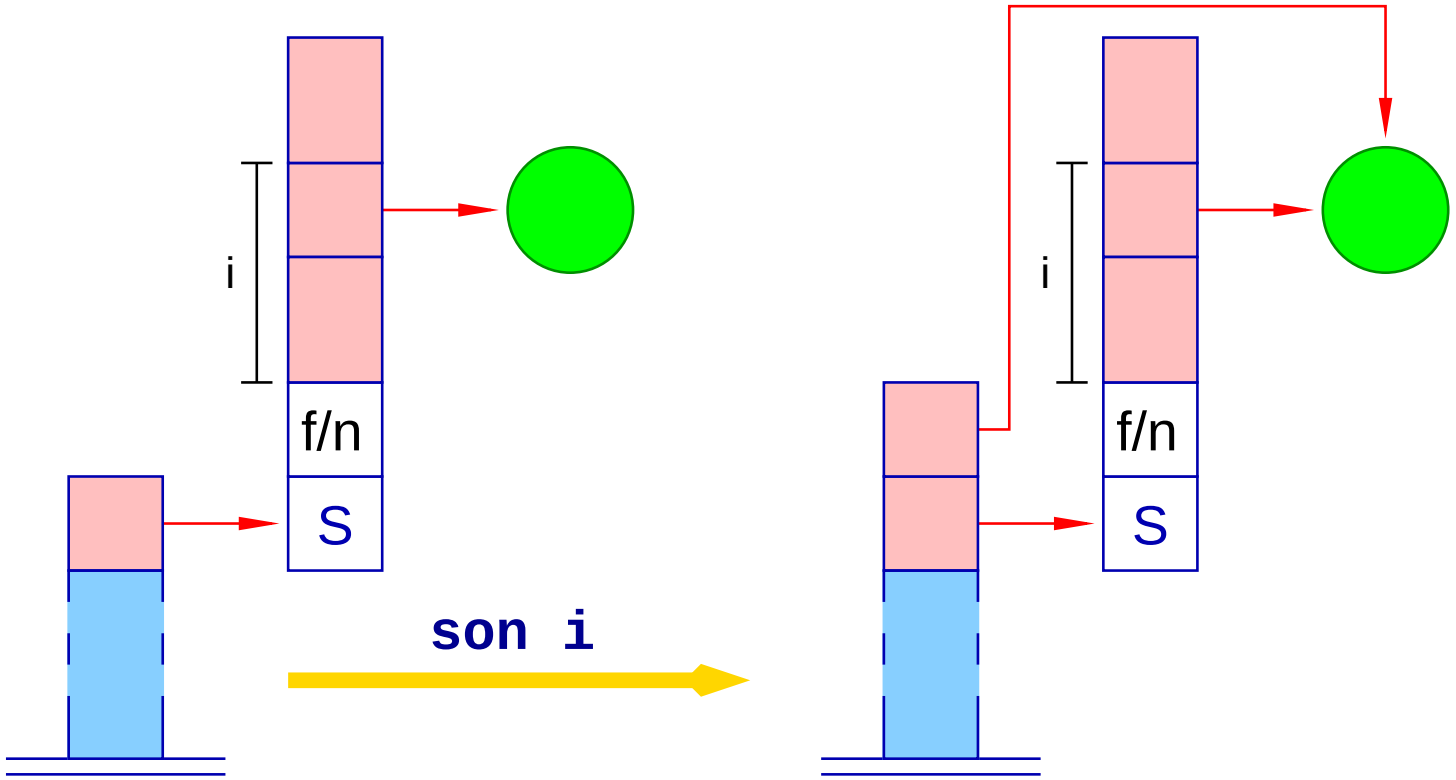
Unifitseerimine




```
switch (H[S[SP]]) {
  case (S,f/n): break;
  case (R,_):   PC = A; break;
  default:      backtrack();
}
```

Unifitseerimine

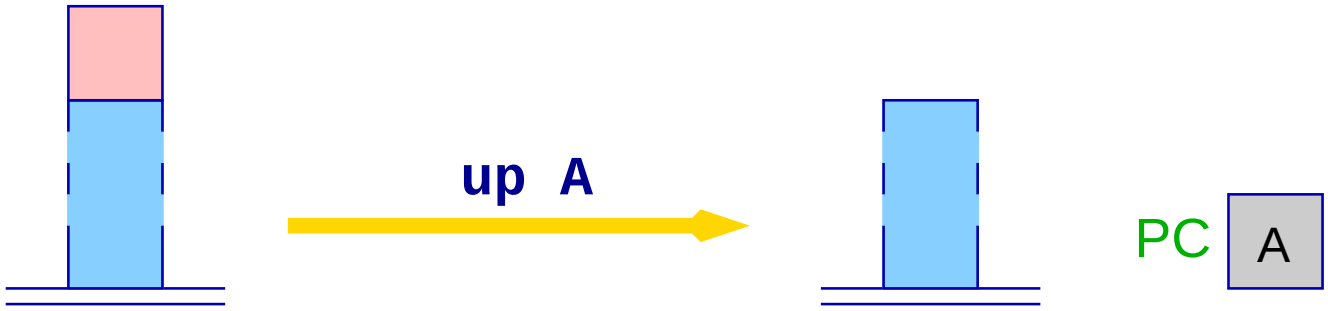
Käsk `son i` lisab i-nda alamtermi aadressi magasini tippu:




```
S[SP+1] = deref (H[S[SP]]+i);
SP++;
```

Unifitseerimine

Käsk `up A` eemaldab magasinist ülemise pesa ja hüppab unifitseerimisele järgnevale koodile:




```
SP--;
PC = A;
```

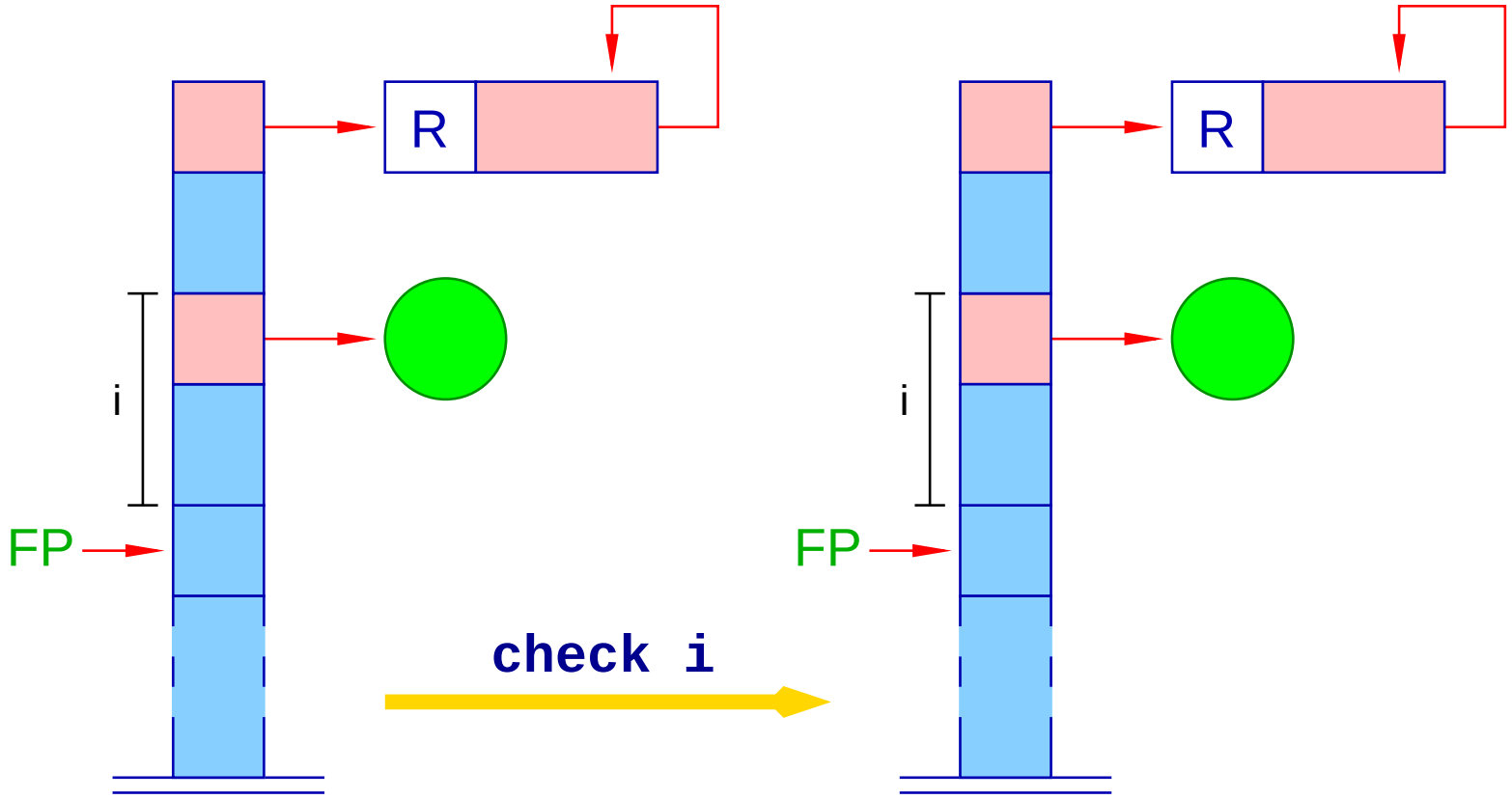
Unifitseerimine

- Initsialiseerimata muutja korral lülitume kontrollimiselt ümber termi konstrueerimisele.
- Enne uue termi konstrueerimist peame veenduma, et ta ei sisalda magasini tipus olevat muutujat:
  - funktsioon $\mathrm{ivars}(t)$ väljastab termis t initsialiseeritud muutujate hulga;
  - makro check $\{Y_1, \ldots, Y_d\}\ \rho$ genereerib vajalikud testid:

$$\begin{aligned} \text{check}\ \{Y_1, \ldots, Y_d\}\ \rho \quad = \quad & \textsf{check}\ (\rho\ Y_1) \\ & \ldots \\ & \textsf{check}\ (\rho\ Y_d) \end{aligned}$$

Unifitseerimine

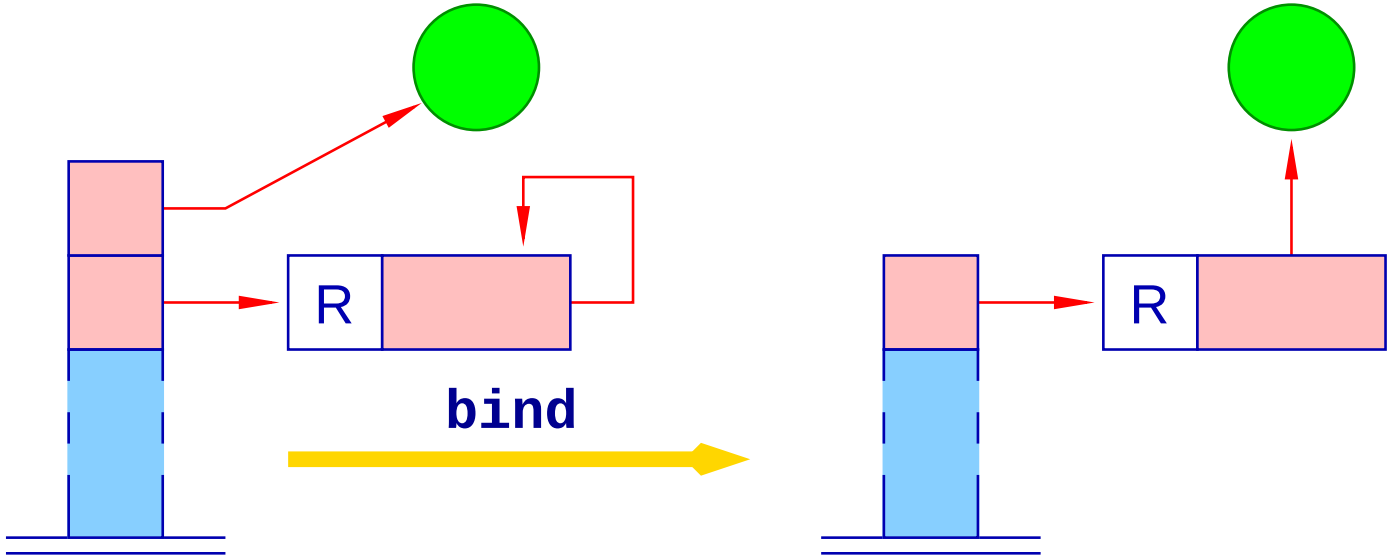
Käsk `check i` kontrollib, kas magasini tipus olev (initsialiseerimata) muutuja esineb i-nda muutujaga seotud termis:




```
if (!check (S[SP], deref(S[FP+i])))
    backtrack();
```

Unifitseerimine

Käsk `bind` seob (initsialiseerimata) muutuja konstrueeritud termiga:




```
H[S[SP-1]] = (R, S[SP]);
trail (S[SP-1]);
SP = SP - 2;
```

Unifitseerimine

Näide: olgu antud term $t \equiv f(g(\bar{X}, Y), a, Z)$ ja keskkond $\rho = \{X \mapsto 1,\ Y \mapsto 2,\ Z \mapsto 3\}$. Siis $\text{code}_U\ t\ \rho$ emiteerib koodi:

```
    ustruct f/3 A1
    son 1
    ustruct g/2 A2
    son 1
    uref 1
    son 2
    uvar 2
    up B2
A2: check 1
    putref 1
    putvar 2
    putstruct g/2
    bind
B2: son 2
    uatom a
    son 3
    uvar 3
    up B1
A1: check 1
    putref 1
    putvar 2
    putstruct g/2
    putatom a
    putvar 3
    putstruct f/3
    bind
B1: ...
```

Reeglid

Reeglile vastav kood:

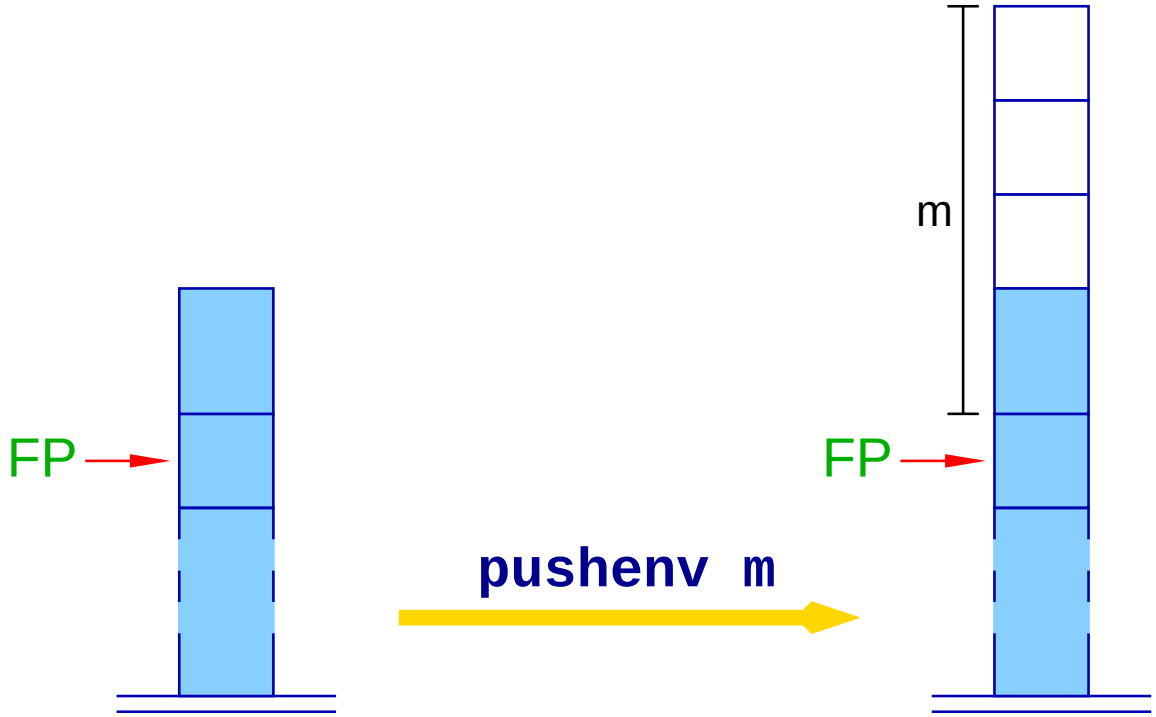
- reserveerib magasinis mälu lokaalsetele muutjuatele;
- väärtustab reegli keha;
- vabastab magasini freimi (kui võimalik).

Reeglis sisalduvaid lokaalseid muutujaid tähistame $\{X_1, \ldots, X_m\}$.
NB! Esimesed k lokaalset muutujat on formaalsed parameetrid.

$$\begin{aligned} \text{code}_C\ (p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n) \quad = \quad & \texttt{pushenv m} \\ & \text{code}_G\ g_1\ \rho \\ & \ldots \\ & \text{code}_G\ g_n\ \rho \\ & \texttt{popenv} \end{aligned}$$

Reeglid

Käsk `pushenv m` reserveerib mälu lokaalsetele muutujatele




```
SP = FP + m;
```

Reeglid

Näide: olgu antud reegel

$$r \equiv \mathtt{a}(X, Y) \leftarrow \mathtt{f}(\bar{X}, X_1),\ \mathtt{a}(\bar{X}_1, \bar{Y})$$

Siis $\mathrm{code}_C\ r$ emiteerib koodi:

```
pushenv 3          call f/2          putref 2
mark A          A: mark B            call a/2
putref 1           putref 3       B: popenv
putvar 3
```

Predikaadid

- Predikaat q/k on defineeritud reeglite jadana $rr \equiv r_1 \ldots r_f$.
- Predikaatide transleerimine toimub funktsiooni code_P abil.
- Juhul, kui predikaat on defineeritud ühe reegliga (so. $f = 1$), siis:

$$\text{code}_P\ r \quad = \quad \text{code}_C\ r$$

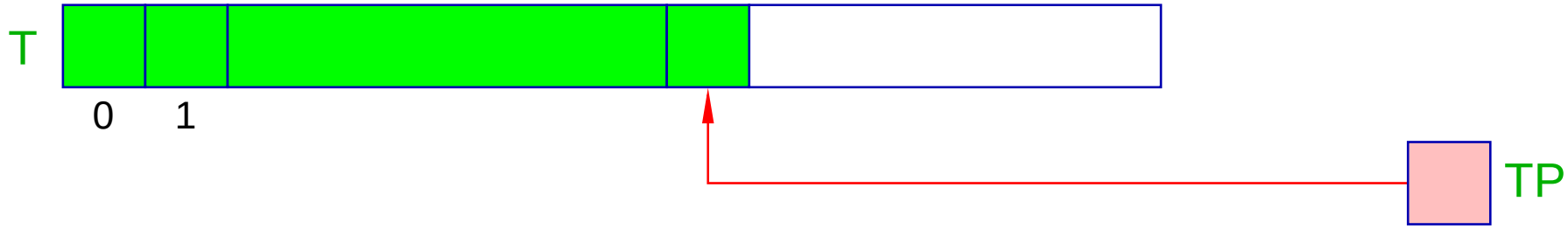
- Kui predikaat on defineeritud mitme reegli abil, siis:
  - tuleb "proovida" esimest reeglit;
  - kui see ebaõnnestub, tuleb proovida järgmist reeglit; jne.

Predikaadid

- Kui unifitseerimine ebaõnnestub, kutsutakse välja täitmisaegne funktsioon `backtrack()`.
- Eesmärk on kerida kogu arvutuskäik tagasi kuni nn. tagasipöördumispunktini (backtrack point); so. (dünaamiliselt) viimase eesmärgini, kus saab järgmist reeglit "proovida".
- Selleks, et taastada varem kehtinud muutujate seosed, kasutati uute seoste loomiseks funktsiooni `trail()`.

Predikaadid

Jälg:

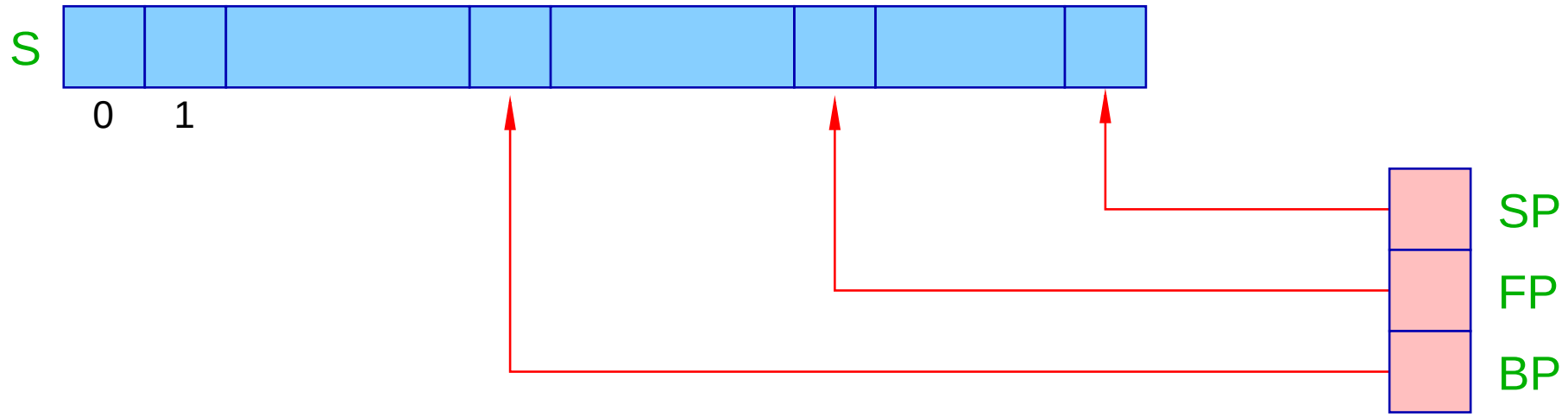


T = Trail — mälupiirkond loodud seoste hoidmiseks;

TP = Tail-Pointer — viitab viimasele kasutatud pesale.

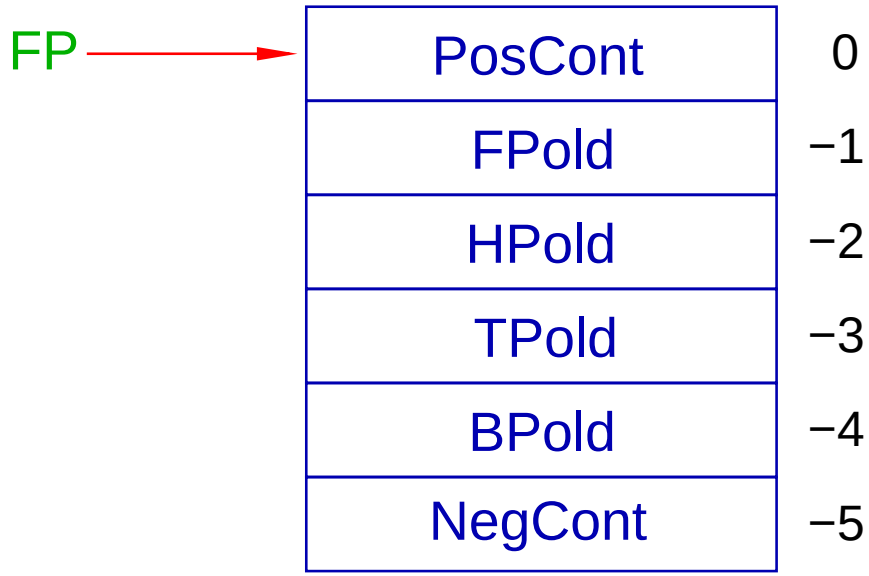
Predikaadid

Hetkel kehtivale tagasipöördumispunktile viitab register BP.



Predikaadid

Tagasipöördumispunkti esitab magasinis freim:



Parema loetavuse huvides kasutame edaspidi makrosid:

PosCont	≡	S[FP]	TPold	≡	S[FP−3]
FPold	≡	S[FP−1]	BPold	≡	S[FP−4]
HPold	≡	S[FP−2]	NegCont	≡	S[FP−5]

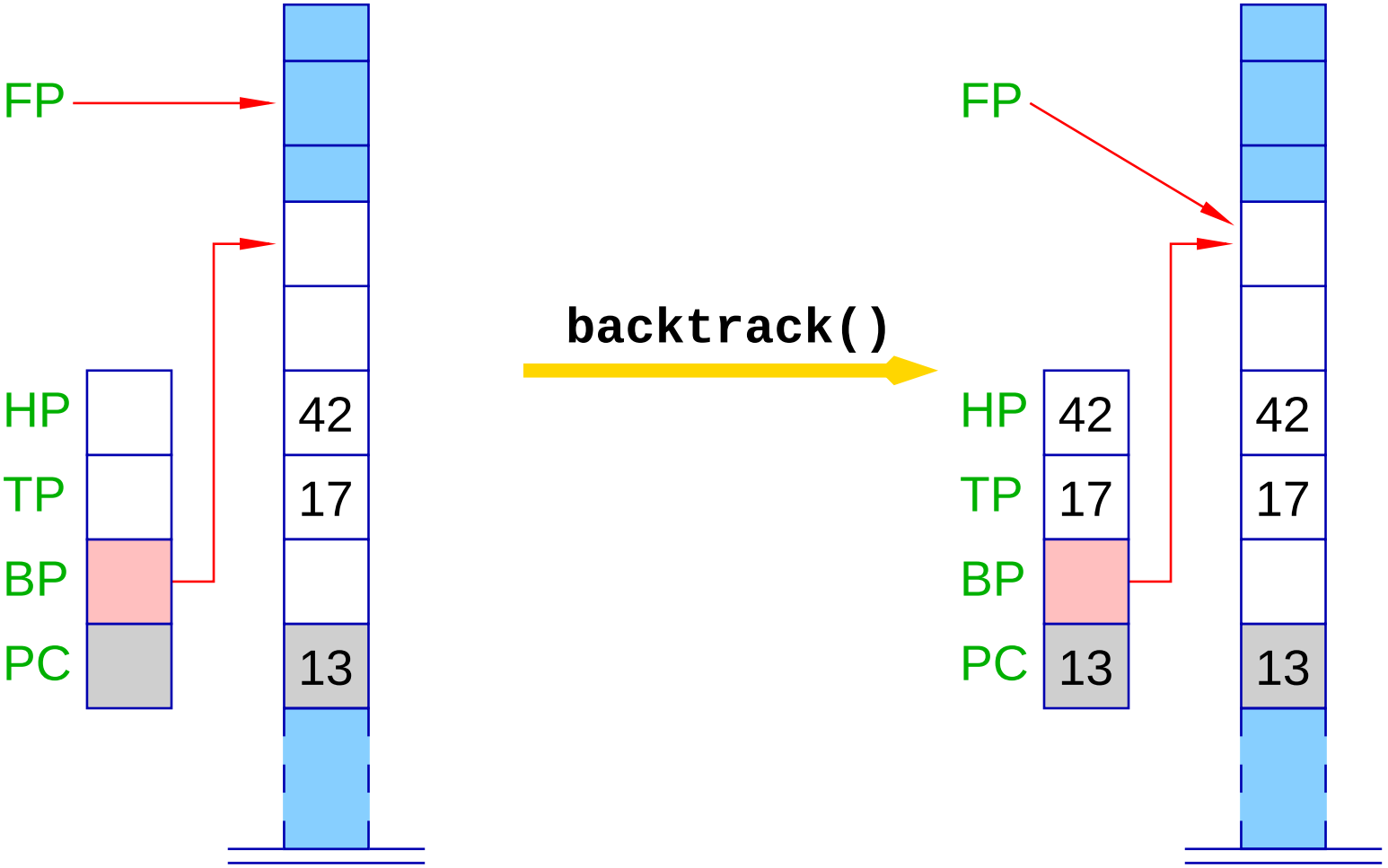
Predikaadid

Täitmisaegne funktsioon `backtrack()` taastab registrite seisu vastavalt tagasipöösrdumispunkti freimile:

```
void backtrack() {
    FP = BP;
    HP = HPold;
    reset (TPold,TP);
    TP = TPold;
    PC = NegCont;
}
```

Funktsioon `reset()` taastab muutujate seosed.

Predikaadid



Predikaadid

- Pärast viimast tagasipöördumispunkti loodud muutujad ja seosed saab eemaldada lihtsalt taastades registri HP vana väärtuse.
- See töötab juhul kui nooremad muutujad alati viitavad vanematele objektidele.
- Seosed, kus vanemad muutujad viitavad noorematele objektidele, tuleb eemaldada "käsitsi".
- Need seosed on salvestatud "jäljes" (trail).

Predikaadid

Funktsioon `trail()` lisab uue seose juhul kui argument viitab nooremale objektile:

```
void trail (ref u) {
  if (u < S[BP-2]) {
    TP = TP+1;
    T[TP] = u;
  }
}
```

Pesa `S[BP-2]` sisaldab registrit HP enne tagasipöördumispunkti loomist.

Funktsioon `reset()` eemaldab kõik pärast viimast tagasipöördumispunkti lisatud seosed:

```
void reset (ref x, ref y) {
  for (ref u=y; x<u; u--)
    H[T[u]] = (R,T[u]);
}
```

Predikaadid

Predikaadi q/k, mis on defineeritud reeglitega $r_1, \ldots, r_f \quad (f > 1)$ transleerimisel genereeritakse kood, mis:

- loob tagasipöördumispunkti;
- üksteise järel "proovib" erinevaid reegleid;
- kustutab tagasipöördumispunkti.

Predikaadid

$$\begin{array}{lll} \text{code}_P\ (r_1,\ldots,r_f) = & \text{q/k: } \texttt{setbtp} & \texttt{jump } \texttt{A}_f \\ & \texttt{try } \texttt{A}_1 & \texttt{A}_1\text{: } \text{code}_C\ r_1 \\ & \ldots & \ldots \\ & \texttt{try } \texttt{A}_{f-1} & \texttt{A}_f\text{: } \text{code}_C\ r_f \\ & \texttt{delbtp} & \end{array}$$

NB!

- Tagasipöördumispunkt kustutatakse enne viimase reegli "proovimist".
- Viimasele reeglile hüpatakse ja kehtivale freimile enam tagasi ei pöörduta.

Predikaadid

Näide:

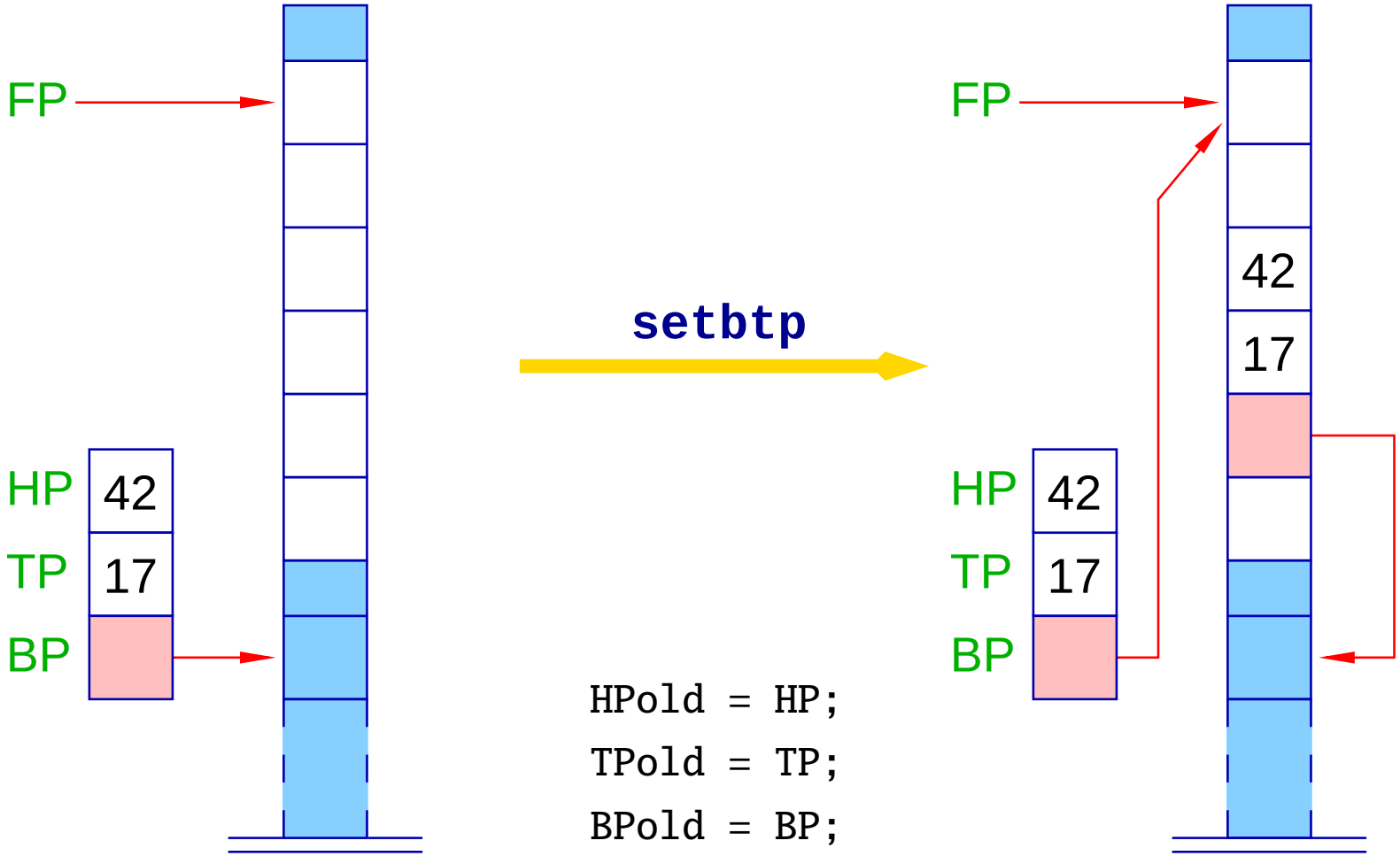
$$\begin{aligned} \mathtt{s}(X) &\leftarrow \mathtt{t}(\bar{X}) \\ \mathtt{s}(X) &\leftarrow \bar{X} = a \end{aligned}$$

Predikaadi `s/1` transleerimisel emiteeritakse kood:

```
s/1: setbtp
     try A
     delbtp
     jump B
A:   pushenv 1
     mark C
     putref 1
     call t/1
C:   popenv
B:   pushenv 1
     putref 1
     uatom a
     popenv
```

Predikaadid

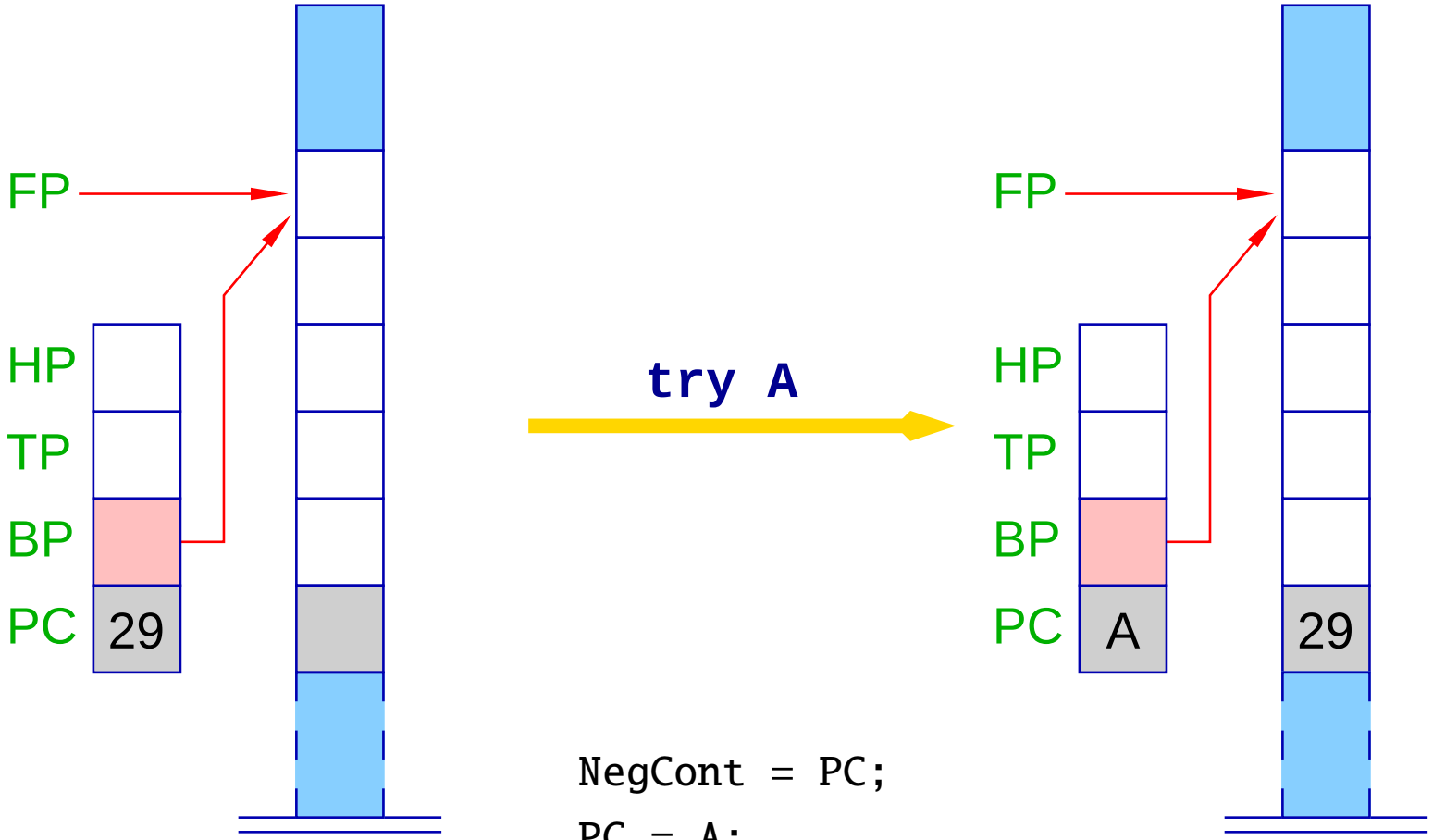
Käsk `setbtp` salvestab registrid HP, TP, BP:




```
HPold = HP;
TPold = TP;
BPold = BP;
BP = FP;
```

Predikaadid

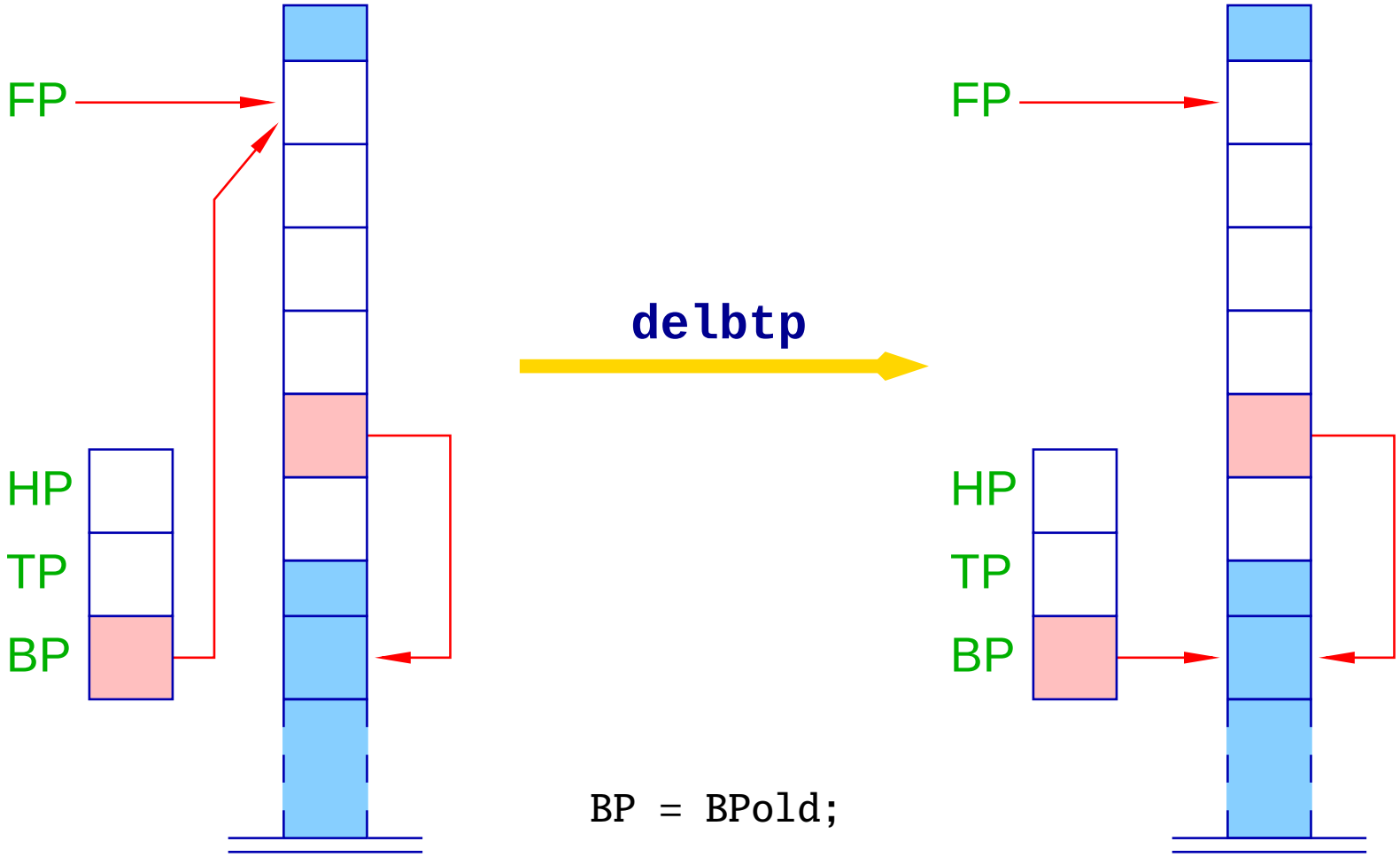
Käsk `try A` salvestab "proovitava" reegli ebaõnnestumisel kasutatava jätku aadressi ja seejärel hüppab reeglile aadrssil A:




```
NegCont = PC;
PC = A;
```

Predikaadid

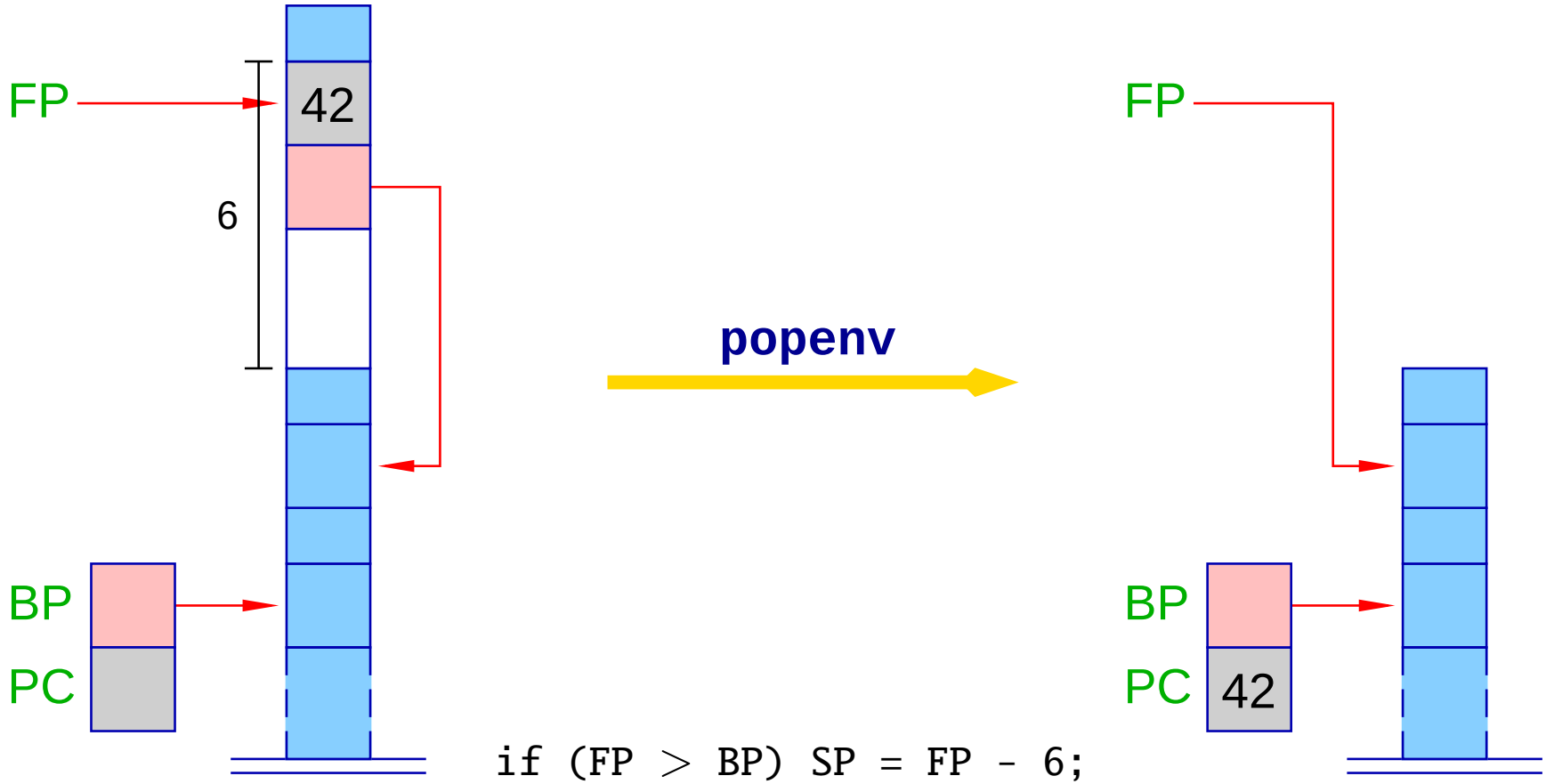
Käsk `delbtp` taastab registri BP väärtuse:




```
BP = BPold;
```

Predikaadid

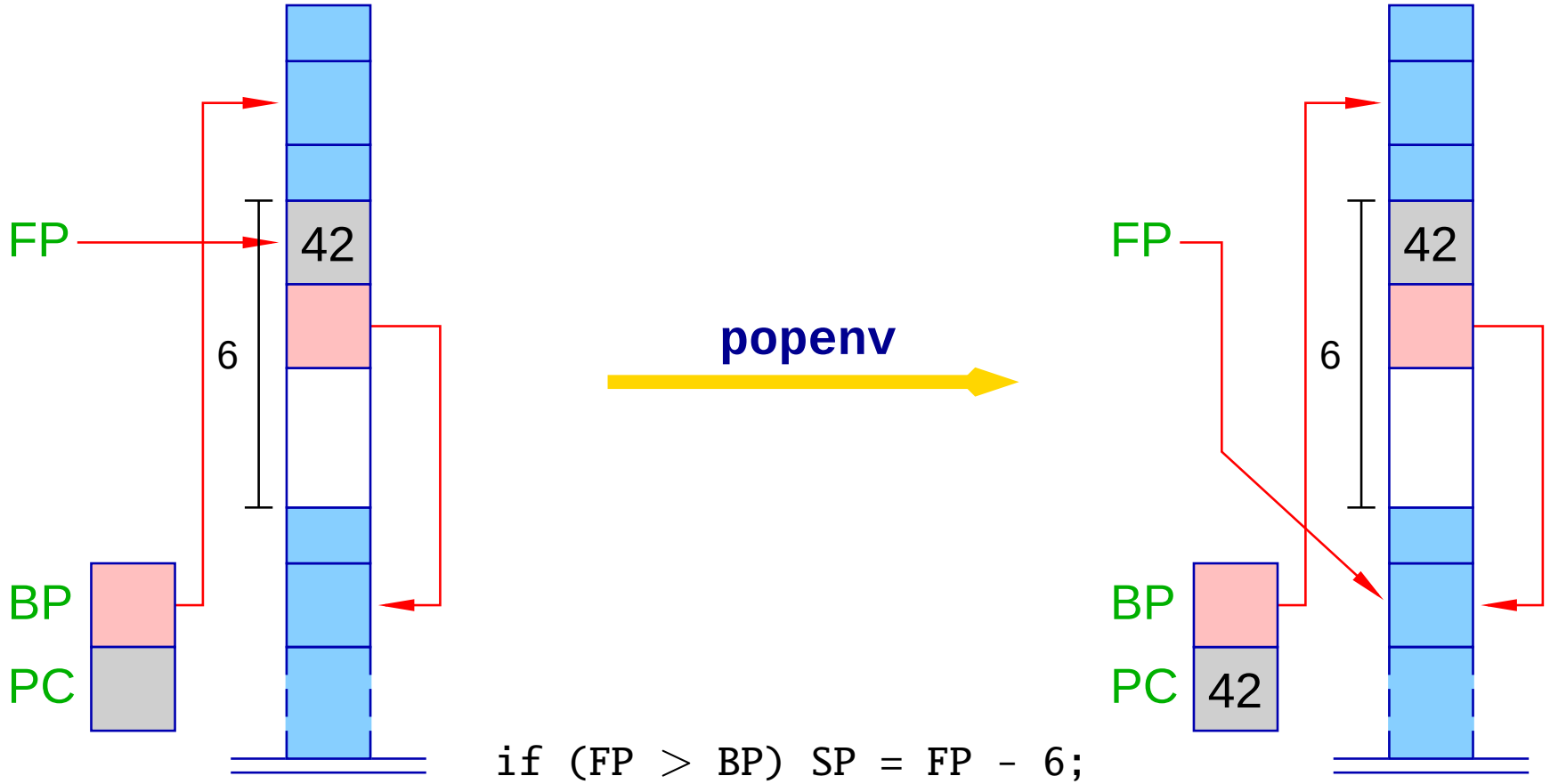
Käsk popenv taastab registrid FP ja PC ning võimalusel vabastab freimi:




```
if (FP > BP) SP = FP - 6;
PC = PosCont;
FP = FPold;
```

Predikaadid

Kui FP $\leq$ BP siis freimi ei vabastata:




```
if (FP > BP) SP = FP - 6;
PC = PosCont;
FP = FPold;
```

Päringud ja programmid

- Programmi $p \equiv rr_1 \ldots rr_h?g$ transleerimisel genereeritakse:
  - päringu g väärtustamisele vastav kood;
  - predikaatide rr_i definitsioonidele vastav kood.
- Päringu väärtustamisele eelneb:
  - registrite initsialiseerimine;
  - globaalsetele muutujatele mälu reserveerimine.
- Päringu väärtustamisele järgneb:
  - globaalsete muutujate väärtuste väljastamine.

Päringud ja programmid

$$
\begin{array}{lll}
\text{code}\ (rr_1 \ldots rr_h\ ?g) = & \texttt{init} & \text{code}_P\ rr_1 \\
& \texttt{pushenv d} & \ldots \\
& \text{code}_G\ g\ \rho & \text{code}_P\ rr_h \\
& \texttt{halt d} &
\end{array}
$$

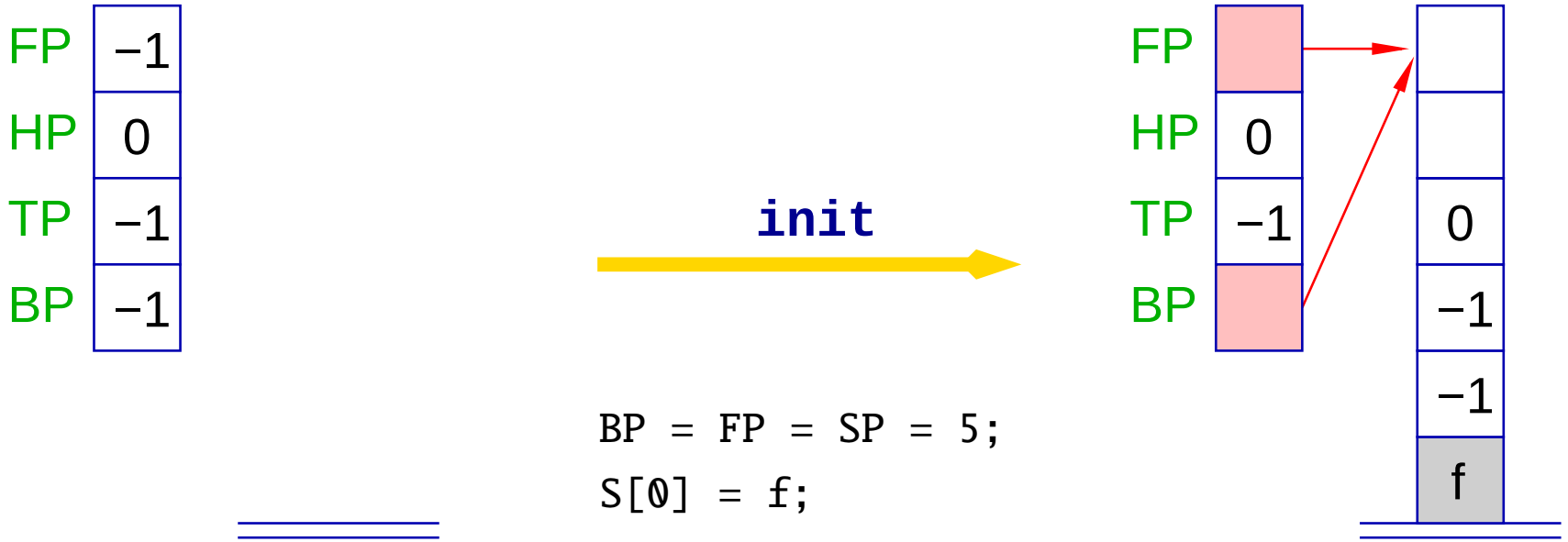
kus $free(g) = \{X_1, \ldots, X_d\}$ ja $\rho = \{X_i \mapsto i \mid i = 1 \ldots d\}$.

Käsk `halt d` ...

- ... lõpetab programmi töö;
- ... väljastab globaalsete muutujate väärtused;
- ... kasutaja nõudel teostab tagasipöördumise.

Päringud ja programmid

Käsk `init` loob algse tagasipöördumispunkti:




```
BP = FP = SP = 5;
S[0] = f;
S[1] = S[2] = -1;
S[3] = 0;
BP = FP;
```

Päringu g ebaõnnestumisel täidetav kood (aadressil f) võib näiteks välja trükkida teate ebaõnnestumisest.

Päringud ja programmid

Näide: $\mathtt{t}(X) \leftarrow \bar{X} = b$ $\quad$ $\mathtt{q}(X) \leftarrow \mathtt{s}(\bar{X})$ $\quad$ $\mathtt{s}(X) \leftarrow \bar{X} = a$

$\mathtt{p} \leftarrow \mathtt{q}(X), \mathtt{t}(\bar{X})$ $\quad$ $\mathtt{s}(X) \leftarrow \mathtt{t}(\bar{X})$ $\quad$ $? \mathtt{p}$

```
    init          p/0: pushenv 1    q/1: pushenv 1    E: pushenv 1
    pushenv 0          mark B            mark D           mark G
    mark A             putvar 1          putref 1         putref 1
    call p/0           call q/1          call s/1         call t/1
  A: halt 0          B: mark C         D: popenv        G: popenv
t/1: pushenv 1           putref 1     s/1: setbtp        F: pushenv 1
    putref 1           call t/1          try E            putref 1
    uatom b          C: popenv           delbtp           uatom a
    popenv                               jump F           popenv
```

Sabarekursioon

Olgu predikaat app/3 defineeritud järgnevalt:

$$\begin{aligned} \mathsf{app}(X, Y, Z) &\leftarrow X = [\,],\ Y = Z \\ \mathsf{app}(X, Y, Z) &\leftarrow X = [H \mid X'],\ Z = [H \mid Z'],\ \mathsf{app}(X', Y, Z') \end{aligned}$$

Reegli viimane eesmärk on predikaadi rekursiivne väljakutse:

- võib püüda teda väärtustada kehtivas freimis;
- pärast väljakutse (õnnestunud) lõpetamist võib kehtivasse freimi tuleku vahele jätta ja minna otse tagasi "ülemisse" freimi.

Sabarekursioon

Olgu antud reegel $r \equiv \mathtt{p}(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$, milles on m lokaalset muutujat ning kus $g_n \equiv \mathtt{q}(t_1, \ldots, t_h)$.

$$
\begin{array}{lll}
\mathrm{code}_C\ r = & \mathtt{pushenv\ m} & \mathrm{code}_A\ t_1\ \rho \\
& \mathrm{code}_G\ g_1\ \rho & \ldots \\
& \ldots & \mathrm{code}_A\ t_h\ \rho \\
& \mathrm{code}_G\ g_{n-1}\ \rho & \mathtt{call\ q/h} \\
& \mathtt{mark\ B} & \mathrm{B:}\ \mathtt{popenv}
\end{array}
$$

Sabarekursioon

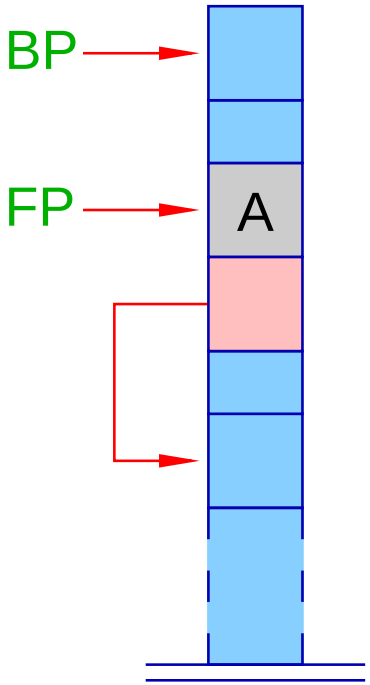
Olgu antud reegel $r \equiv \mathtt{p}(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$, milles on m lokaalset muutujat ning kus $g_n \equiv \mathtt{q}(t_1, \ldots, t_h)$.

$$
\begin{array}{lll}
\mathrm{code}_C\ r = & \texttt{pushenv m} & \mathrm{code}_A\ t_1\ \rho \\
& \mathrm{code}_G\ g_1\ \rho & \ldots \\
& \ldots & \mathrm{code}_A\ t_h\ \rho \\
& \mathrm{code}_G\ g_{n-1}\ \rho & \texttt{lastcall (q/h,m)} \\
& \texttt{lastmark} &
\end{array}
$$

Sabarekursioon

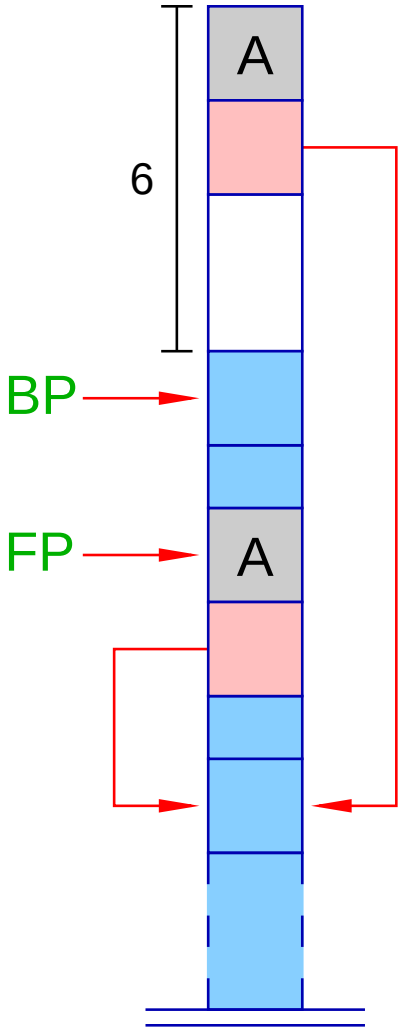
- Kui käesolev reegel ei ole viimane või eesmärgid $g_1, \ldots, g_{n-1}$ on loonud tagasipöördumispunkte, siis FP $\leq$ BP.
- Sel juhul käsk `lasmark` loob uue freimi ning salvestab sinna eelmise freimi viidad.
- Vastasel korral (so. kui FP $>$ BP) ei tehta midagi.

Sabarekursioon



lastmark

```
if (FP ≤ BP) {
  SP = SP + 6;
  S[SP] = PosCont;
  S[SP-1] = FPold;
}
```

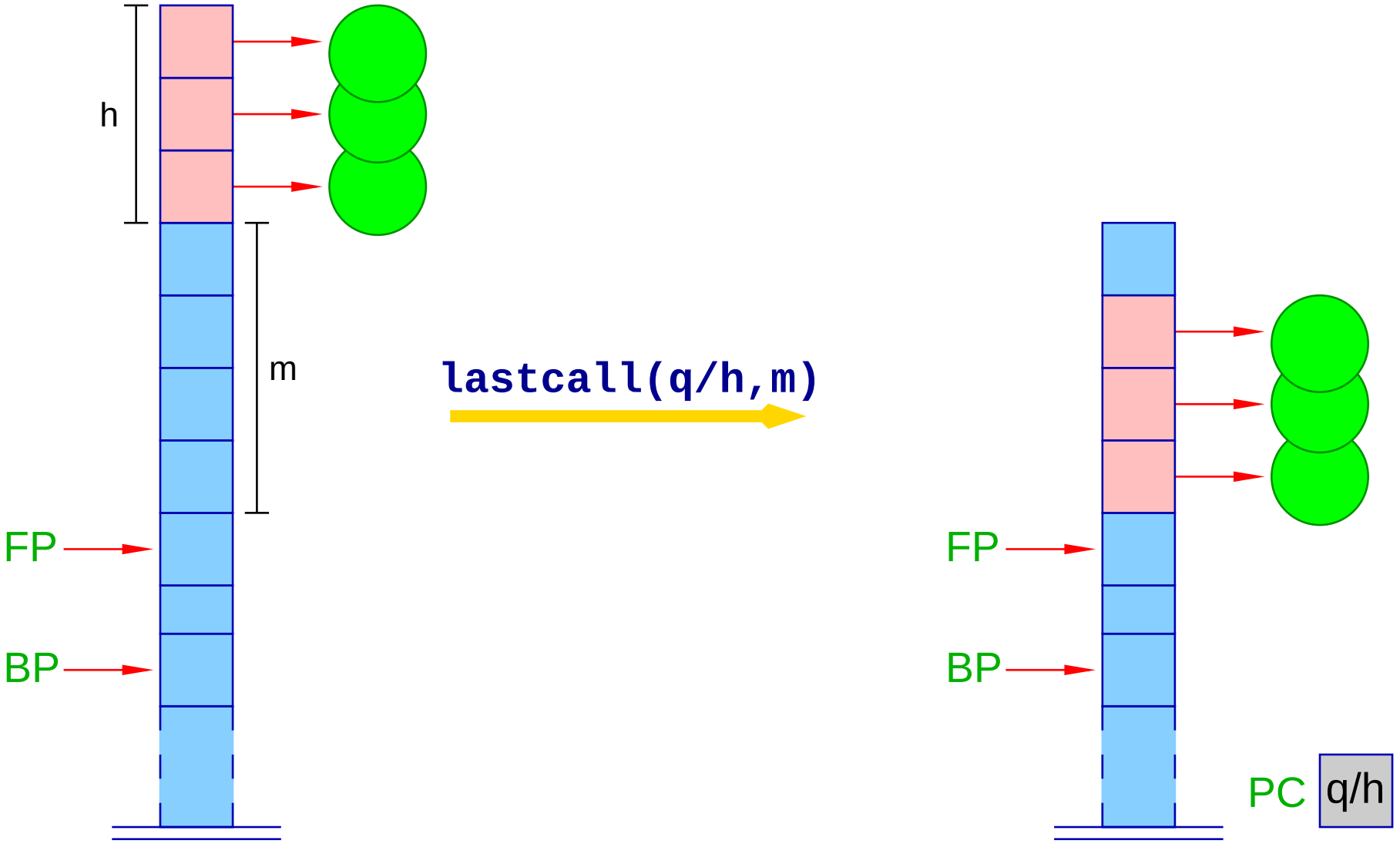


Sabarekursioon

- Kui FP $\leq$ BP, siis käsk `lastcall (q/h,m)` käitub nagu `call q/h`.
- Vastasel korral korduvkasutatakse freimi:
  - pesadele `S[FP+1]`, ..., `S[FP+h]` antakse uued väärtused;
  - hüpatakse predikaadile `q/h` vastavale koodile.

```
lastcall (q/h,m)  =  if (FP ≤ BP) call q/h;
                     else {
                         move (m,h);
                         jump q/h;
                     }
```

Sabarekursioon



Sabarekursioon

Olgu antud reegel

$$\mathtt{a}(X, Y) \leftarrow \mathtt{f}(\bar{X}, X_1),\ \mathtt{a}(\bar{X}_1, \bar{Y})$$

Sabarekursiooni optimeerimisel saame:

```
pushenv 3        putvar 3     putref 3
mark A           call f/2     putref 2
putref 1      A: lastmark     lastcall(a/2,3)
```

NB! Kui tegemist on viimase reegliga ja viimane eesmärk on tema kehas ainuke predikaadi väljakutse, siis võime käsu `lastmark` ära jätta ning käsu `lastcall(q/h,m)` asendada käskudega `move(m,h)` ja `jump q/h`.

Sabarekursioon

Predikaadi app/3 teise reegli jaoks saame:

```
A: pushenv 6
   putref 1
   ustruct [|]/2 B
   son 1
   uvar 4
   son 2
   uvar 5
   up C
B: putvar 4
   putvar 5
   putstruct [|]/2
   bind
C: putref 3
   ustruct [|]/2 D
   son 1
   uref 4
   son 2
   uvar 6
   up E
D: check 4
   putref 4
   putvar 6
   putstruct [|]/2
   bind
E: putref 5
   putref 2
   putref 6
   move(6,3)
   jump app/3
```

Freimi trimmimine

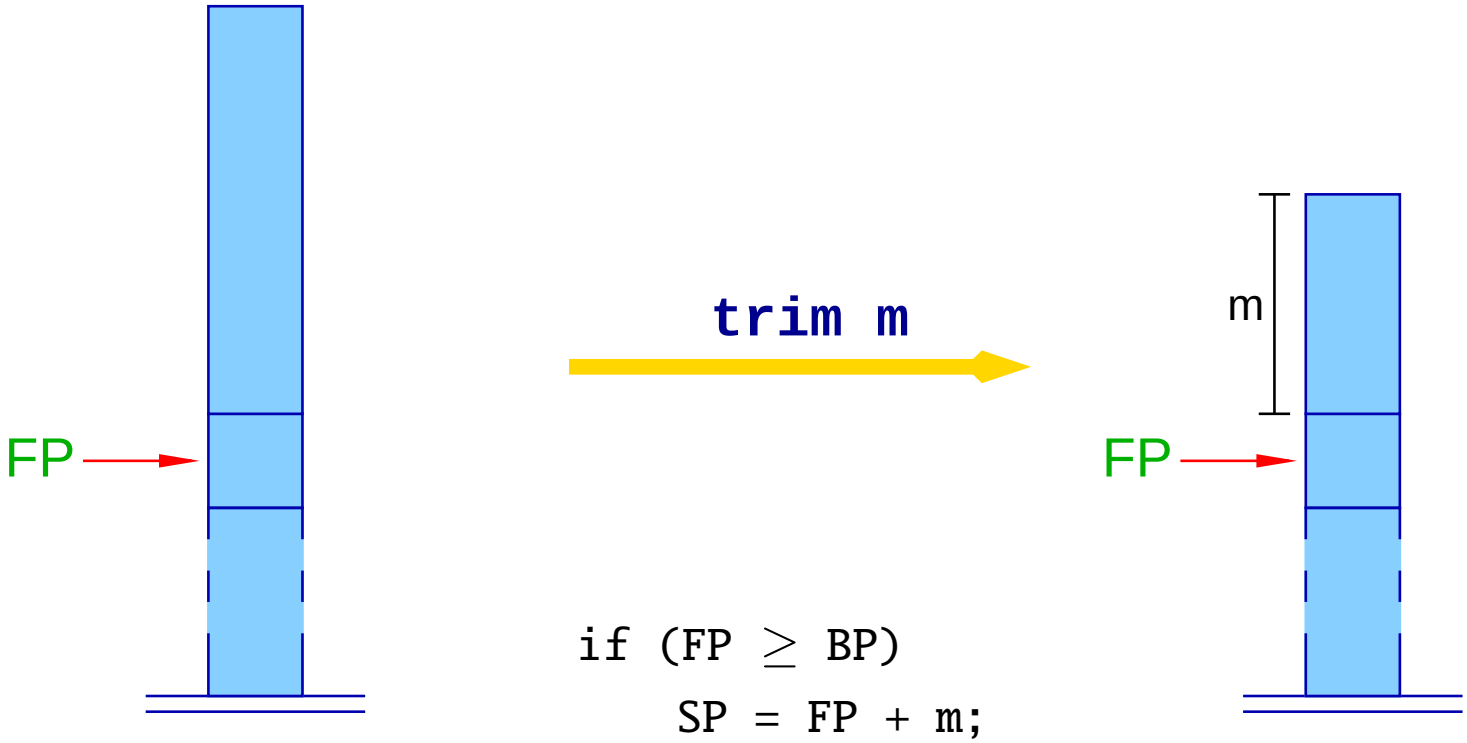
- Järjestame muutujad vastavalt nende elueale.
- Võimaluse korral eemaldame surnud muutujad.
- Näide:

$$\mathtt{a}(X, Z) \leftarrow \mathtt{p}_1(\bar{X}, X_1), \mathtt{p}_2(\bar{X}_1, X_2), \mathtt{p}_3(\bar{X}_2, X_3), \mathtt{p}_4(\bar{X}_3, Z)$$

  - pärast eesmärki $\mathtt{p}_2(\bar{X}_1, X_2)$ on muutuja X_1 surnud;
  - pärast eesmärki $\mathtt{p}_3(\bar{X}_2, X_3)$ on muutuja X_2 surnud.

Freimi trimmimine

Pärast iga surnud muutujatega (mitte-viimast) eesmärki lisame käsu `trim`:



NB! Surnud muutujad võime eemaldada ainult juhul, kui uusi tagasipöördumisi pole loodud.

Freimi trimmimine

Näide:

$$\mathtt{a}(X, Z) \leftarrow \mathtt{p}_1(\bar{X}, X_1), \mathtt{p}_2(\bar{X}_1, X_2), \mathtt{p}_3(\bar{X}_2, X_3), \mathtt{p}_4(\bar{X}_3, Z)$$

Järjestades muutujad $\rho = \{X \mapsto 1, Z \mapsto 2, X_3 \mapsto 3, X_2 \mapsto 4, X_1 \mapsto 5\}$, saame:

```
    pushenv 5        putvar 4     C: trim 3
    mark A           call p2/2        lastmark
    putref 1     B: trim 4            putref 3
    putvar 5         mark C           putref 2
    call p1/2        putref 4         lastcall (p4/2,3)
A:  mark B           putvar 3
    putref 5         call p3/2
```

Reeglite indekseerimine

- Predikaadid on tihti defineeritud oma esimese argumendi variantide eristamise abil.
- Esimest argumenti inspekteerides saame mitmed alternatiivid koheselt välistada.
- Ebaõnnestumise saab varem kindlaks teha.
- Tagasipöördumispunktid saab varem eemaldada.
- Freime saab varem eemaldada.

Reeglite indekseerimine

Näide:

$$\begin{aligned} \mathsf{app}(X, Y, Z) &\leftarrow X = [\,],\ Y = Z \\ \mathsf{app}(X, Y, Z) &\leftarrow X = [H \mid X'],\ Z = [H \mid Z'],\ \mathsf{app}(X', Y, Z') \end{aligned}$$

- Kui esimese argumendi konstruktor on `[]`, siis on ainult esimene reegel kasutatav.
- Kui esimese argumendi konstruktor on `[|]`, siis on ainult teine reegel kasutatav.
- Iga teise konstruktori korral predikaat `app/3` ebaõnnestub.
- Mõlemaid reegleid tuleb proovida ainult juhul, kui esimene argument on initsialiseerimata muutuja.

Reeglite indekseerimine

- Iga konstruktori jaoks toome sisse eraldi proovimisahelad.
- Kontrollime esimese argumendi juurmist tippu.
- Tulemusest sõltuvalt teostame indekseerimitud hüppe vastavale proovimisahelale.

Olgu predikaat `p/k` defineeritud reeglite jadana $rr \equiv r_1 \ldots r_m$. Tähistame makroga tchains rr proovimisahelaid, mis vastavad unifitseerimiste $X_1 = t$ juurmistele konstruktoritele.

Reeglite indekseerimine

Näide:

Vaatame predikaati app/3 ja eeldame, et tema kahele reeglile vastavad algusaadressid A_1 ja A_2. Siis saame järgnevad neli proovimisahelat:

```
VAR: setbtb   // variables        NIL: jump A1   // atom []
     try A1                     CONS: jump A2   // constructor [|]
     delbtp                      ELSE: fail     // default
     jump A2
```

Uus käsk `fail` "hoolitseb kõigi konstruktorite eest peale `[]` ja `[|]`

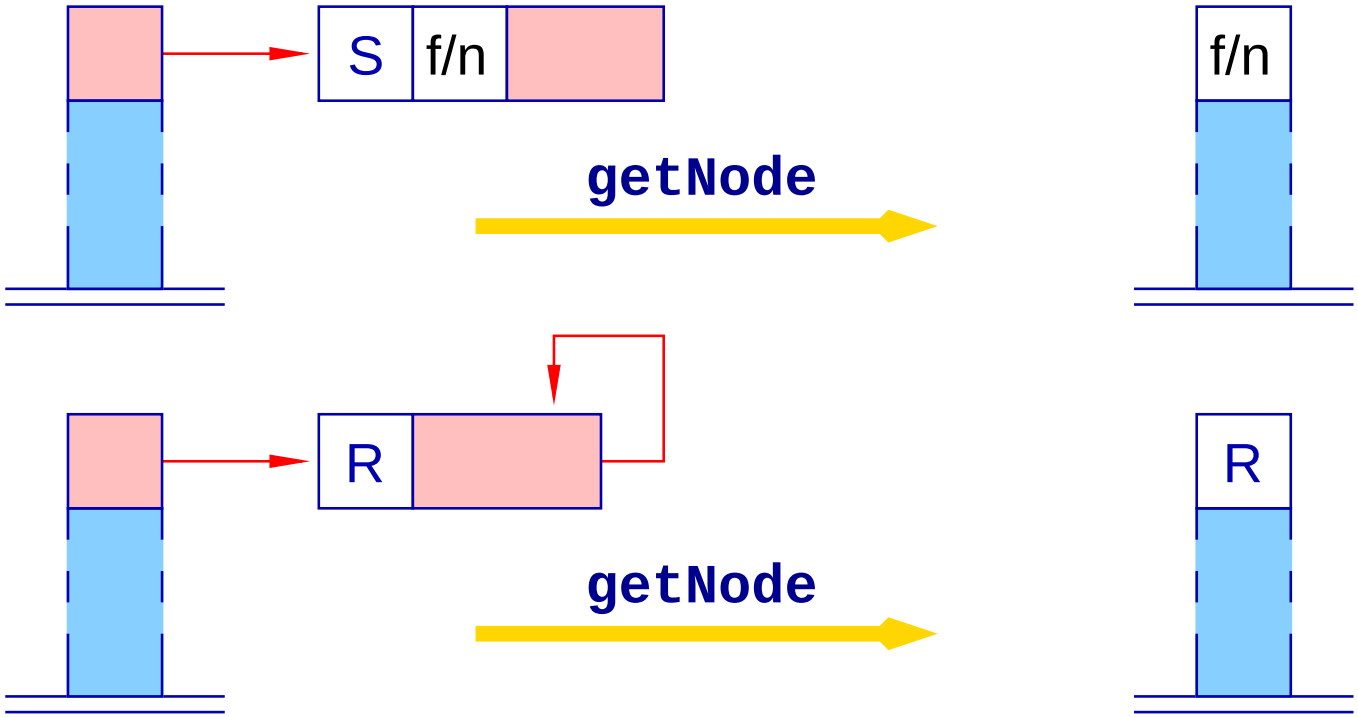
```
fail  =  backtrack()
```

Reeglite indekseerimine

Genereerime predikaadi `p/k` jaoks koodi:

$$
\begin{array}{rcl}
\text{code}_P\ rr & = & \texttt{putref 1} \\
 & & \texttt{getNode} \\
 & & \texttt{index p/k} \\
 & & \text{tchains } rr \\
 & & \mathrm{A}_1\text{: } \text{code}_C\ r_1 \\
 & & \ldots \\
 & & \mathrm{A}_m\text{: } \text{code}_C\ r_m
\end{array}
$$

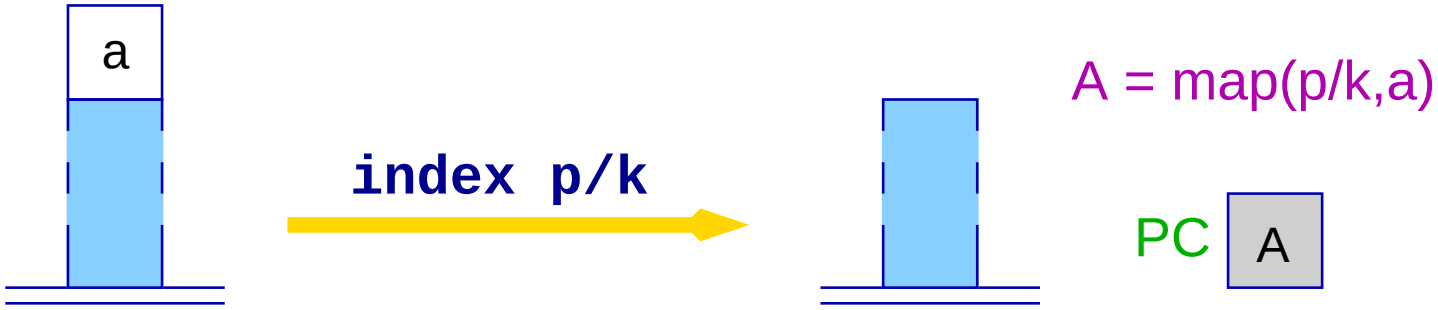
Reeglite indekseerimine




```
switch (H[S[SP]]) {
    case (S,f/n):   S[SP] = f/n; break;
    case (A,a):     S[SP] = a; break;
    case (R,_):     S[SP] = R;
}
```

Reeglite indekseerimine

Käsk `index p/k` teostab indekseeritud hüppe vastavale proovimisahelale:




```
PC = map(p/k,S[SP]);
SP = SP - 1;
```

Funktsioon `map()` väljastab vastava proovimisahela algusaadressi. Tavaliselt on defineeritud läbi mingi paisktabeli.

Lõikeoperaator

Laiendame keelt Proll lõikeoperaatoriga "!" (cut), mis võimaldab ilmutatult ära lõigata tagasipöördumisel proovitavaid harusid.

Näide:

$$\begin{aligned} \texttt{branch}(X,Y) &\leftarrow \texttt{p}(X),\ !,\ \texttt{q}_1(X,Y) \\ \texttt{branch}(X,Y) &\leftarrow \texttt{q}_2(X,Y) \end{aligned}$$

Kui kõik eesmärgid enne lõiget on õnnestunud, siis valik kinnitatakse: tagasipöördumine saab minna ainult neisse tagasipöördumispunktidesse, mis eelnesid predikaadi väljakutsele.

Lõikeoperaator

Lõikeoperaatori transleerimisel:

- taastame registri BP, andes talle kehtivast freimist uueks väärtuseks BPold;
- eemaldame kõik freimid, mis on lokaalsete muutujate peal.

Vastavalt transleerime lõikeoperaatori käskude jadaks:

```
prune
pushenv m
```

kus m on reeglis esinevate (veel elavate) lokaalsete muutujate arv.

Lõikeoperaator

Näide:

$$\texttt{branch}(X,Y) \leftarrow \texttt{p}(X),\ !,\ \texttt{q}_1(X,Y)$$
$$\texttt{branch}(X,Y) \leftarrow \texttt{q}_2(X,Y)$$

Transleerimine annab koodi:

```
setbtp      A: pushenv 2     C: prune          B: pushenv 2
try A          mark C           pushenv 2         putref 1
delbtp         putref 1         lastmark          putref 2
jump B         call p/1         putref 1          move(2,2)
                                putref 2          jump q2/2
                                lastcall(q1/2,2)
```

Lõikeoperaator

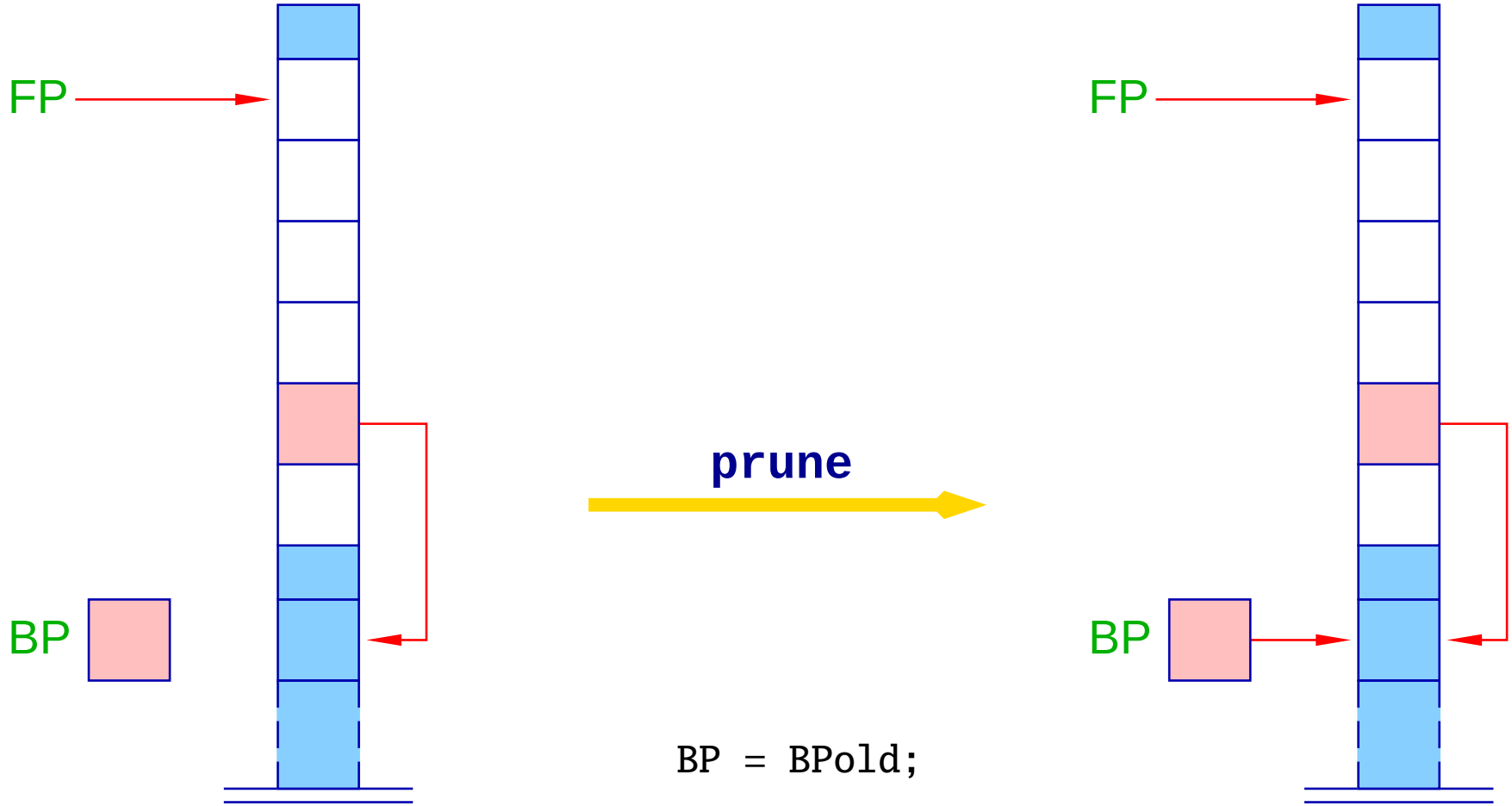
Näide:

$$\texttt{branch}(X,Y) \leftarrow \texttt{p}(X),\ !,\ \texttt{q}_1(X,Y)$$
$$\texttt{branch}(X,Y) \leftarrow \texttt{q}_2(X,Y)$$

. . . või kasutades optimeeritud transleerimist:

```
setbtp      A: pushenv 2     C: prune          B: pushenv 2
try A          mark C           pushenv 2         putref 1
delbtp         putref 1         putref 1          putref 2
jump B         call p/1         putref 2          move(2,2)
                                move(2,2)         jump q2/2
                                jump q1/2
```

Lõikeoperaator



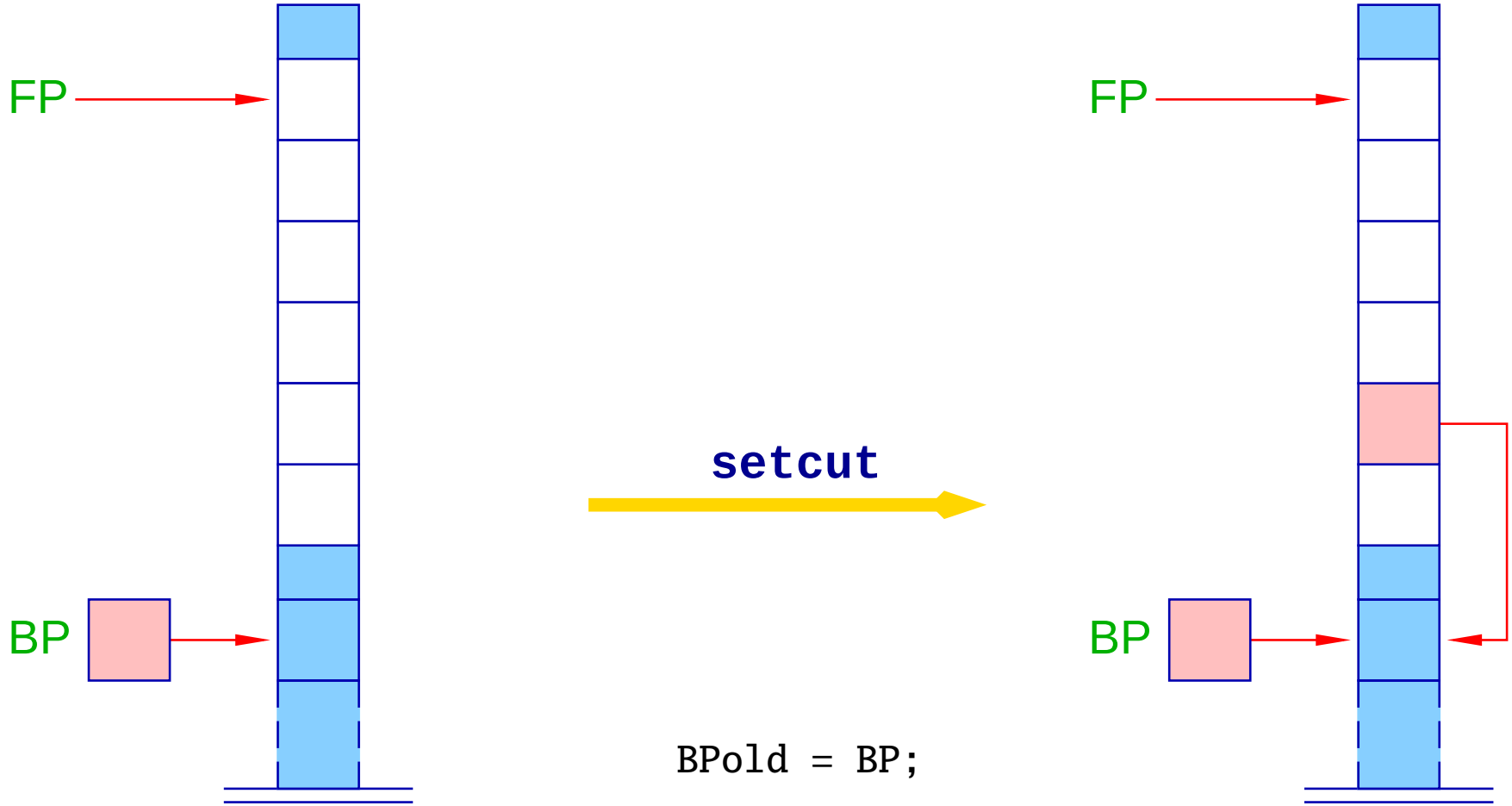
Lõikeoperaator

Probleem:

Kui predikaat on defineeritud ühe reegli abil, siis pole vana BP väärtust freimi salvestatud.

Selleks, et lõikeoperaator töötaks ka ühereegliliste predikaatide ja pikkusega üks proovimisahelate jaoks, lisame iga sellise reegli algusse (või enne hüpet) käsu `setcut`.

Lõikeoperaator




```
BPold = BP;
```

Lõikeoperaator

Lõpetav näide: predikaat `notP` on edukas, kui `p` ebaõnnestub ja vastupidi:

$$\texttt{notP}(X) \leftarrow \texttt{p}(X), \texttt{!}, \texttt{fail}$$
$$\texttt{notP}(X) \leftarrow$$

kus `fail` ebaõnnestub alati. Siis saame:

```
setbtp     A: pushenv 1     C: prune          B: pushenv 1
try A         mark C           pushenv 1         popenv
delbtp        putref 1         fail
jump B        call p/1         popenv
```