Sabarekursioon

Olgu predikaat `app`/3 defineeritud järgnevalt:

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

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 $g_n \equiv \mathtt{q}(t_1, \ldots, t_h)$.

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

Sabarekursioon

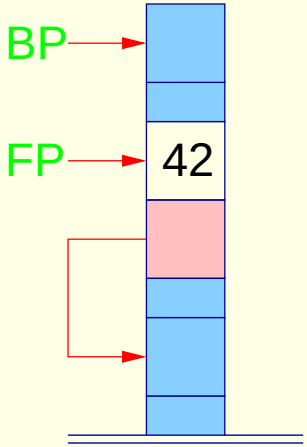
Olgu antud reegel $r \equiv \mathtt{p}(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$, milles on m lokaalset muutujat ning kus $g_n \equiv \mathtt{q}(t_1, \ldots, t_h)$.

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

Sabarekursioon

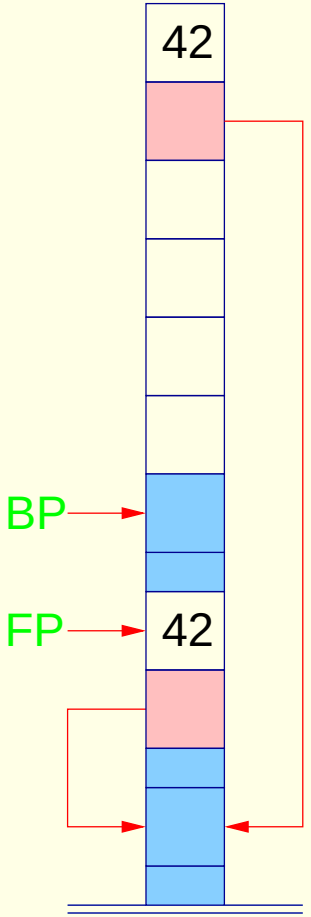
- Kui käesolev reegel ei ole viimane või eesmärgid $g_1, \ldots, g_{n-1}$ on loonud tagasipöördumispunkte, siis FP $\leq$ BP.
- Sel juhul käsk `lastmark` loob uue freimi ning salvestab sinna eelmise freimi viidad.
- Vastasel korral (so. kui FP $>$ BP) ei tehta midagi.

Sabarekursioon



lastmark

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

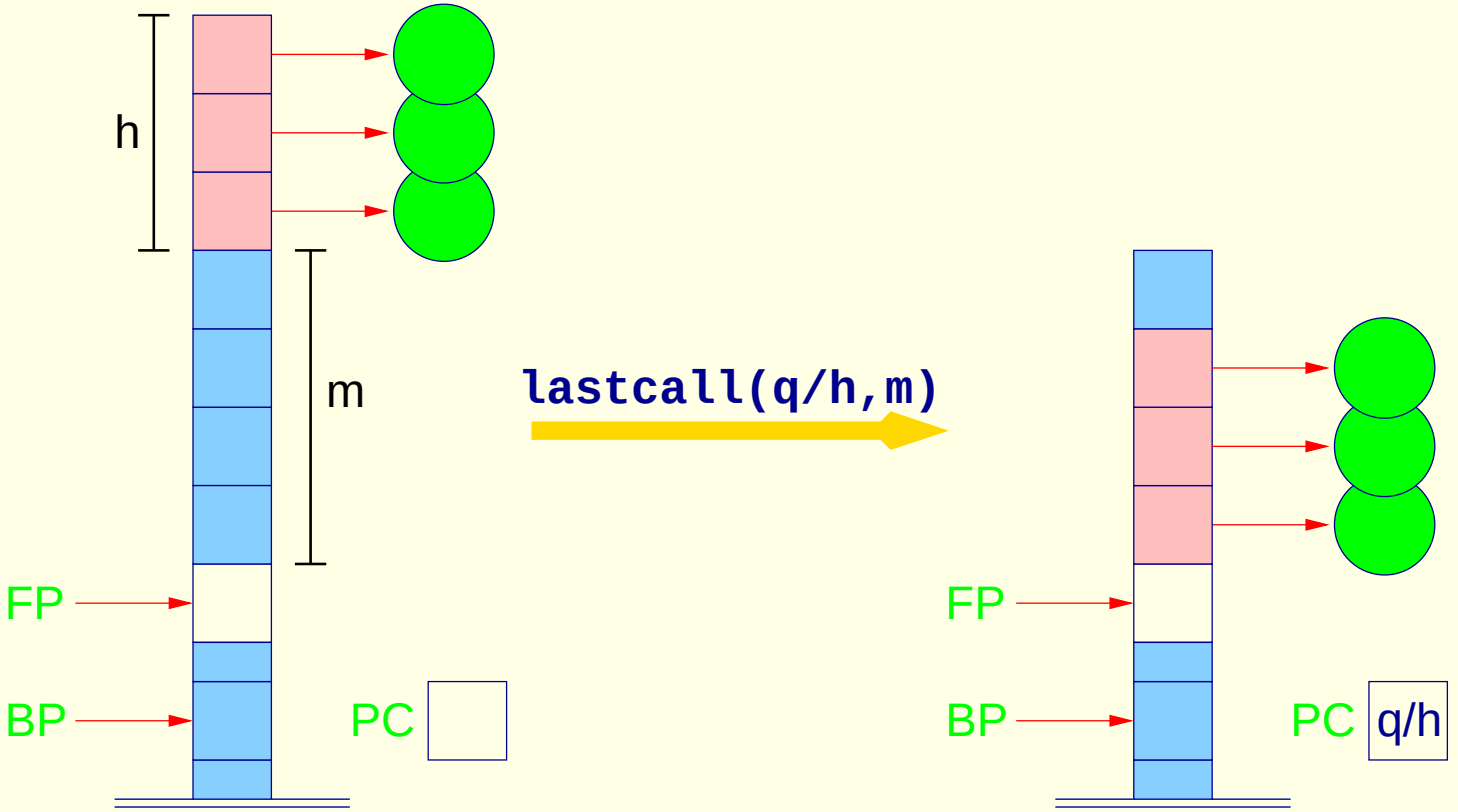


Sabarekursioon

- Kui FP $\leq$ BP, siis käsk `lastcall(q/h,m)` käitub nagu `call q/h`.
- Vastasel korral korduvkasutatakse kehtivat freimi:
  - pesadele $S[FP+1], \ldots, S[FP+h]$ 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

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

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); 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
   ustrct []/2 D
   son 1
   uref 4
   son 2
   uvar 6
   up E
D: check 4
   putref 4
   putvar 6
   putstrct []/2
   bind
E: putref 5
   putref 2
   putref 6
   move(6,3)
   jump app/3
```

Freimi trimmimine

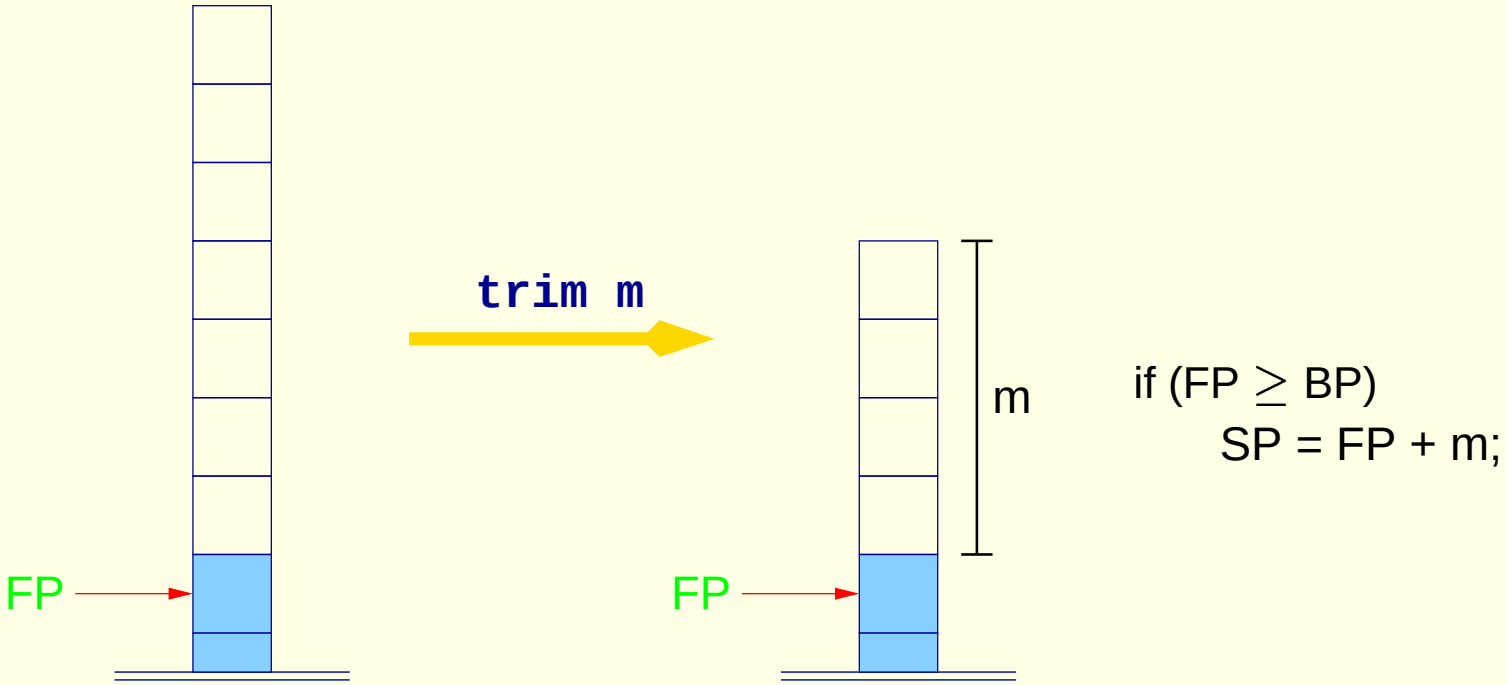
- Järjestame muutujad vastavalt nende *elueale*.
- Võimaluse korral eemaldame *surnud* muutujad.
- Näide:

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

  - pärast eesmärki $\mathtt{p}_2(\overline{X_1}, X_2)$ on muutuja X_1 surnud;
  - pärast eesmärki $\mathtt{p}_3(\overline{X_2}, X_3)$ 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 tagasipöördumispunkte pole loodud.

Freimi trimmimine

Näide:

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

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

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

Reeglite indekseerimine

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

Reeglite indekseerimine

Näide:

$$\texttt{app}(X,Y,Z) \quad \leftarrow \quad X = [\,],\ Y = Z$$
$$\texttt{app}(X,Y,Z) \quad \leftarrow \quad X = [H \mid X'],\ Z = [H \mid Z'],\ \texttt{app}(X',Y,Z')$$

- 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 indekseeritud 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 on A_1 ja A_2. Siis saame järgnevad neli proovimisahelat:

```
VAR:  setbtp      // variables
      try A1
      delbtp
      jump A2

NIL:  jump A1     // atom []

CONS: jump A2     // constructor [|]

ELSE: fail        // default
```

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

```
fail  =  backtrack()
```

Reeglite indekseerimine

Genereerime predikaadi `p/k` jaoks koodi:

$\text{code}_P\ rr$ =

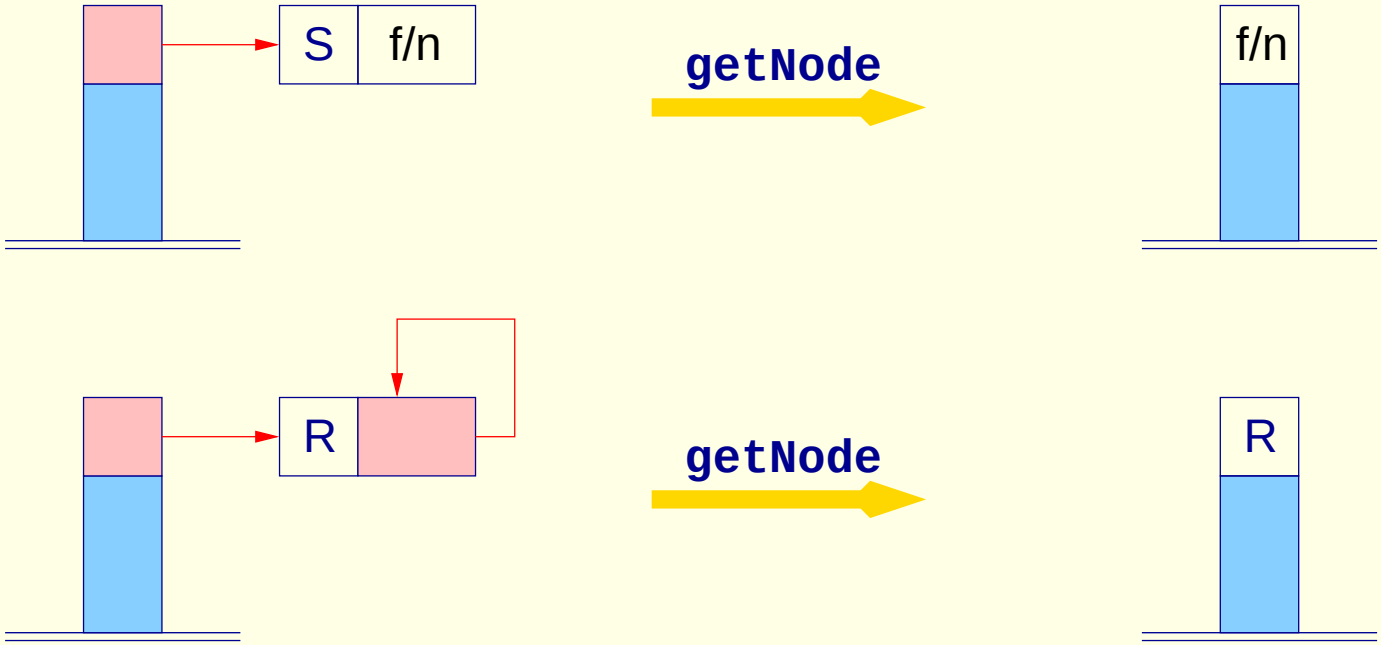
```
        putref 1
        getNode
        index p/k
        tchains rr
```

A_1: $\text{code}_C\ r_1$

...

A_m: $\text{code}_C\ r_m$

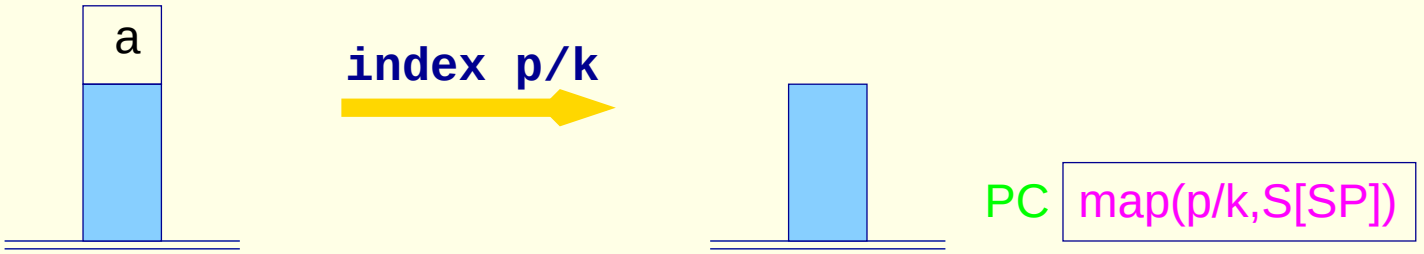
Reeglite indekseerimine




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

Lõikeoperaator

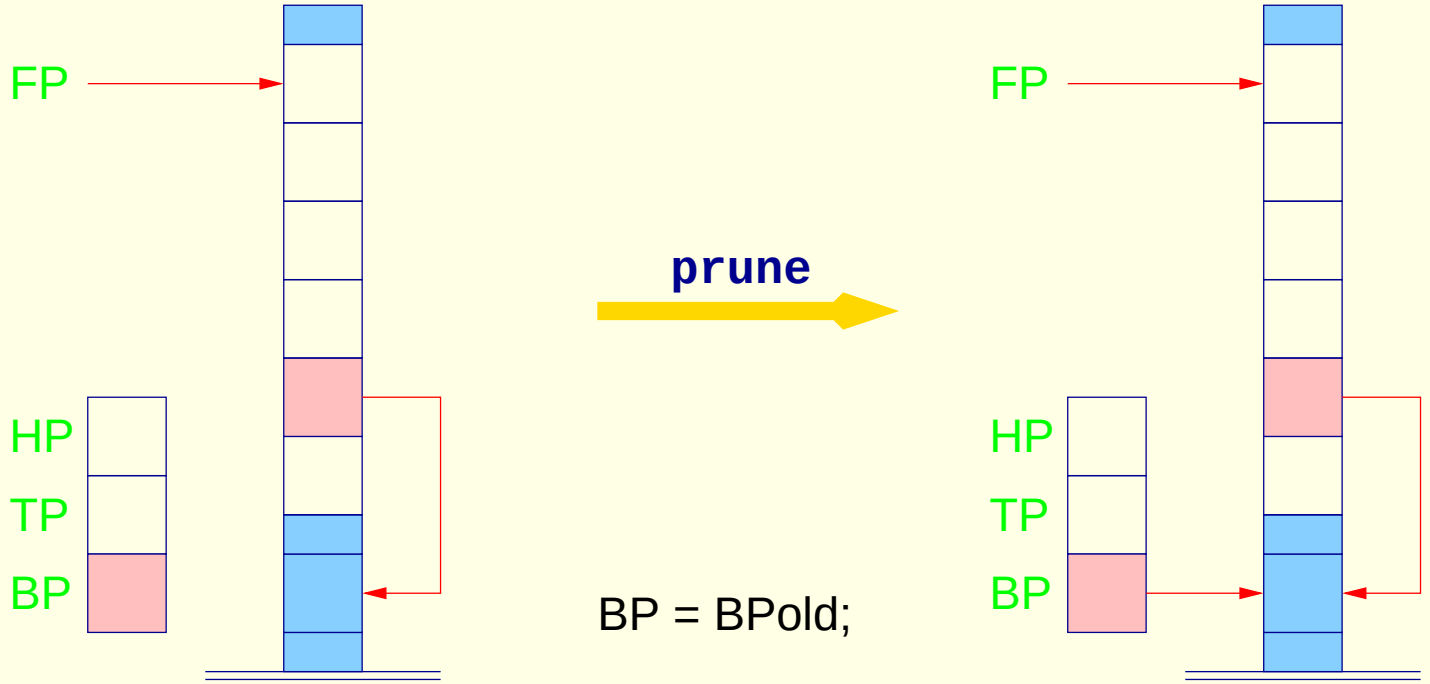
Näide:

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

. . . või kasutades optimeeritud transleerimist:

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

Lõikeoperaator



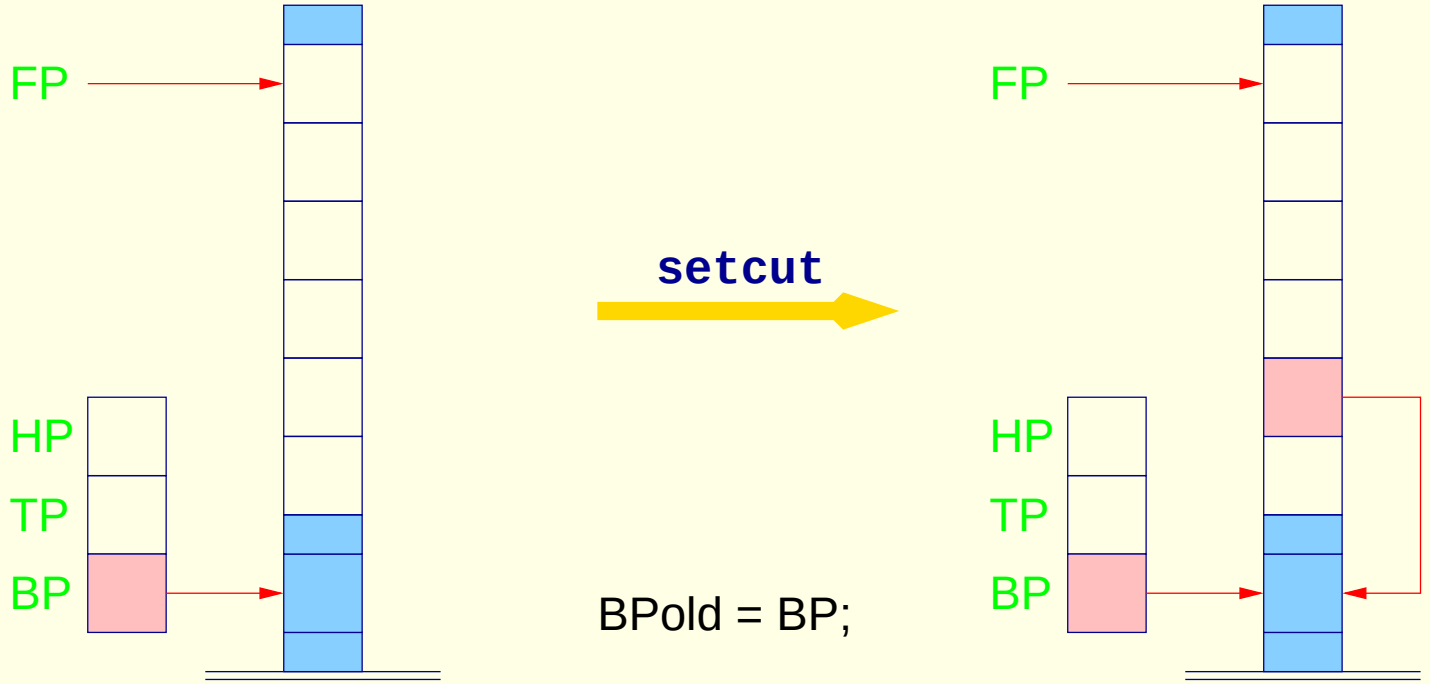
Lõikeoperaator

Probleem:

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

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

Lõikeoperaator



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
```