32.3 Zusammenfügung

Nehmen wir an, das Prädikat q/k sei durch die Klauseln $rr \equiv r_1, \ldots, r_f$ $(f > 1)$ definiert.
Wir benötigen code, um

- den Rücksetz-Punkt zu initialisieren;
- sukzessive die Alternativen zu probieren;
- den Rücksetz-Punkt aufzugeben.

Das bedeutet:

$$
\begin{array}{rcll}
\text{code}_P\ rr & = & \text{q/k}: & \text{setbtp} \\
 & & & \text{try } A_1 \\
 & & & \ldots \\
 & & & \text{try } A_{f-1} \\
 & & & \text{delbtp} \\
 & & & \text{jump } A_f \\
 & & A_1: & \text{code}_C\ r_1 \\
 & & & \ldots \\
 & & A_f: & \text{code}_C\ r_f
\end{array}
$$

Beachte:

- Wir geben den Rücksetz-Punkt vor der letzten Alternative auf :-)
- Wir springen die letzte Alternative direkt an — um nie mehr zu diesem Rahmen zurückzukehren :-))

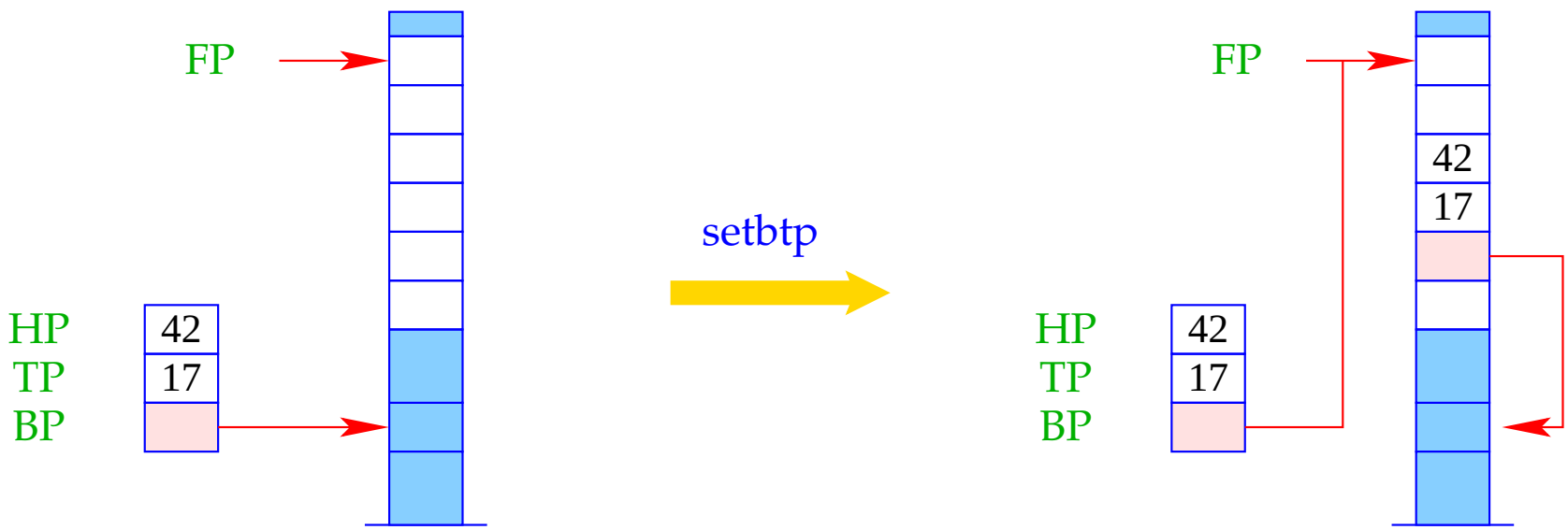
Beispiel:

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

Die Übersetzung des Prädikats s liefert:

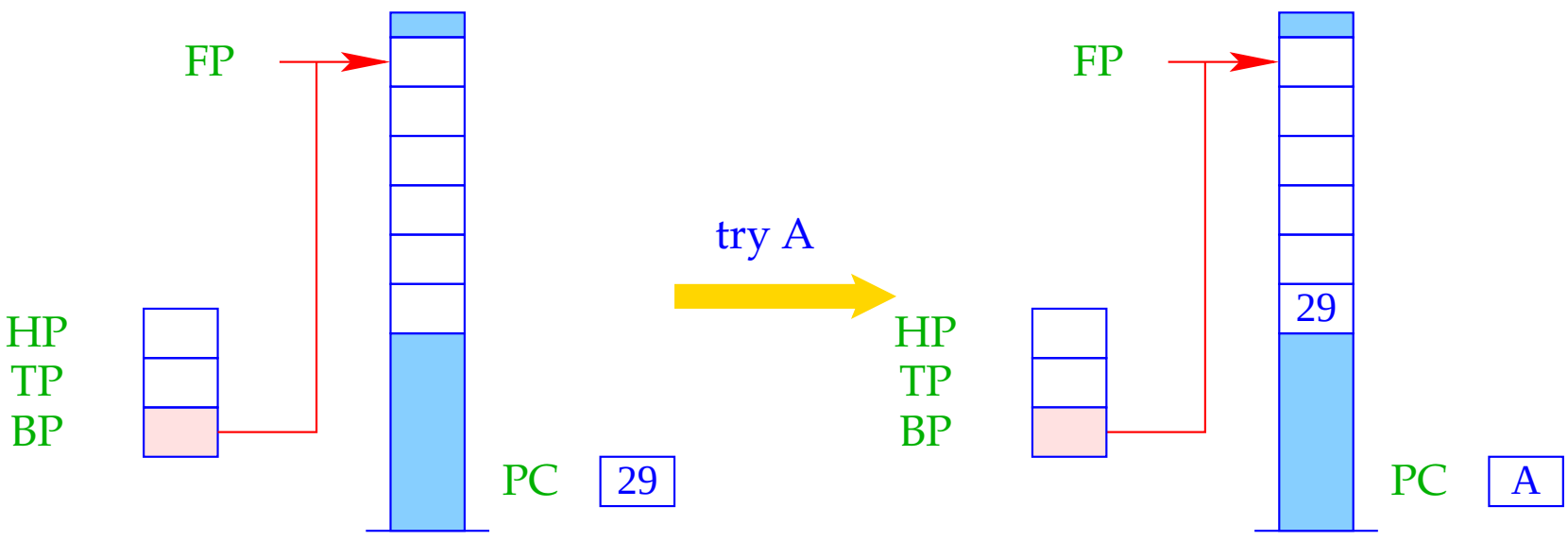
```
s/1:  setbtp     A:  pushenv 1    B:  pushenv 1
      try A          mark C           putref 1
      delbtp         putref 1         uatom a
      jump B         call t/1         popenv
                 C:  popenv
```

Der Befehl setbtp rettet die Register HP, TP, BP:



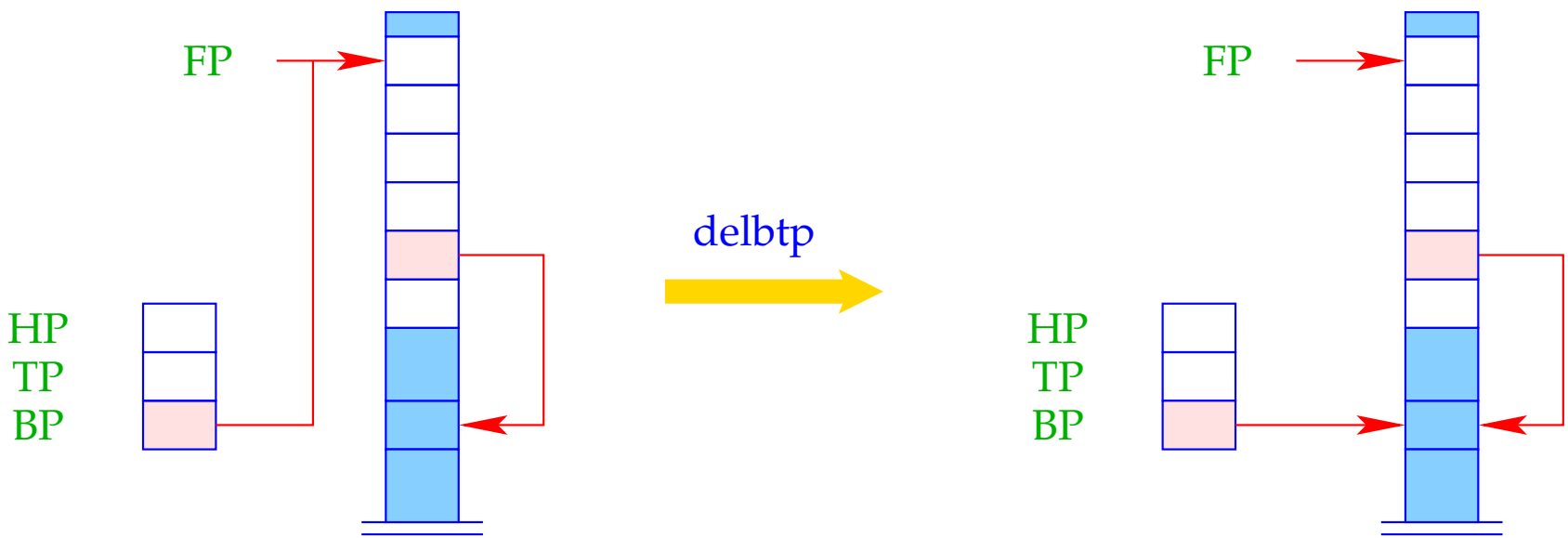
HPold = HP;
TPold = TP;
BPold = BP;
BP = FP;

Der Befehl try A versucht die Alternative an der Adresse A und ersetzt die negative Fortsetzungs-Adress mit dem aktuellen PC:



negForts = PC;
PC = A;

Der Befehl delbtp restauriert den alten Backtrack-Pointer:



BP = S[FP-4];

32.4 Endbehandlung von Klauseln

Erinnern wir uns an die Übersetzung von Klauseln:

$$
\begin{array}{lcl}
\text{code}_C\ r & = & \text{pushenv m} \\
 & & \text{code}_G\ g_1\ \rho \\
 & & \ldots \\
 & & \text{code}_G\ g_n\ \rho \\
 & & \text{popenv}
\end{array}
$$

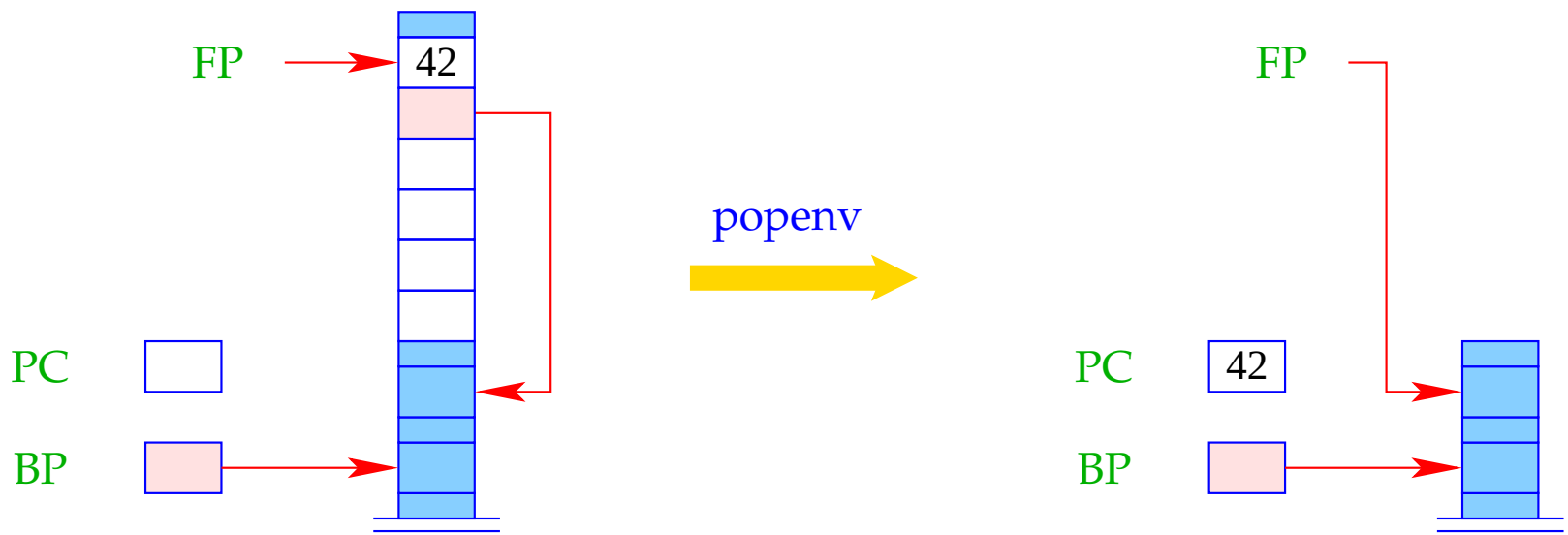
Der aktuelle Kellerrahmen kann aufgegeben werden ...

- falls die aktuelle Klausel die letzte oder einzige ist; und
- falls alle Ziele im Rumpf definitiv beendet sind.

$\Longrightarrow$ der aktuelle Rücksetz-Punkt ist älter :-)

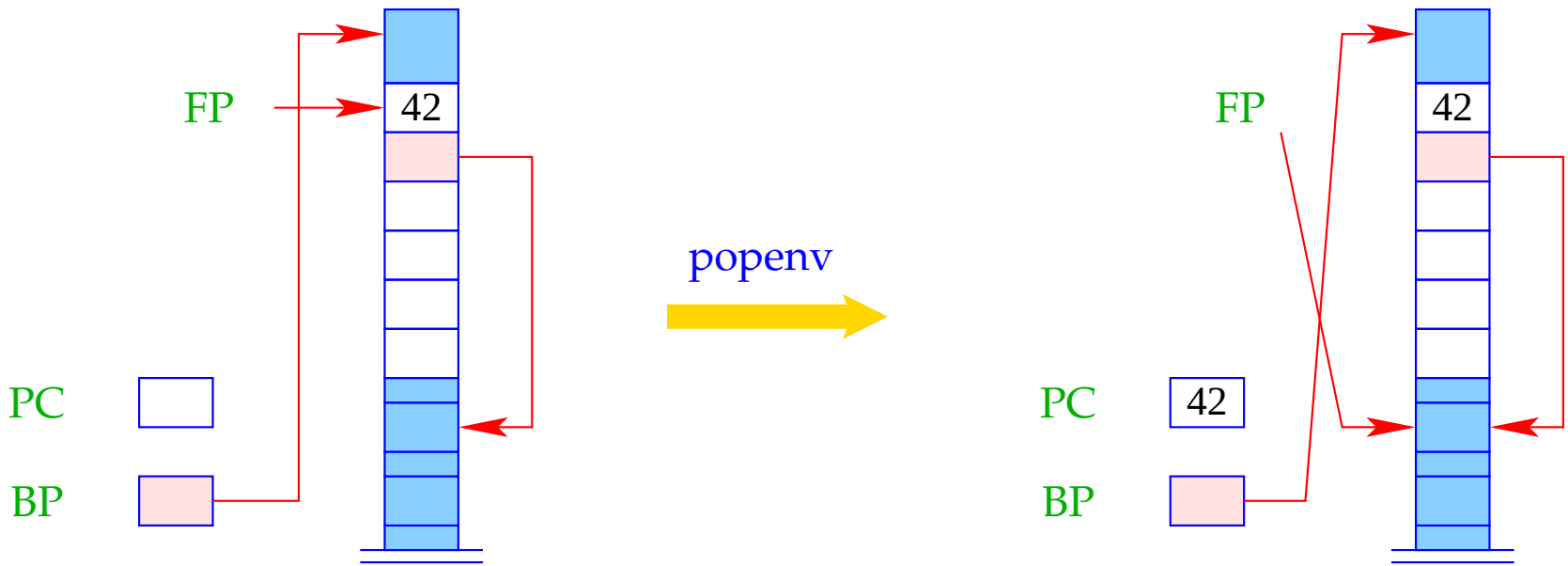
$\Longrightarrow$ FP > BP

Der Befehl popenv restauriert die Register FP und PC und versucht, den Kellerrahmen frei zu geben:




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

Achtung: popenv gibt den Kellerrahmen nicht immer auf :




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

Falls die Freigabe scheitert, werden weitere Daten oben auf dem Keller allokiert. Bei Rückkehr zu dem Rahmen können die lokalen Variablen aber immer noch über den FP adressiert werden :-))

33 Anfragen und Programme

Die Übersetzung des Programms $p \equiv rr_1 \ldots . rr_h \; ?g$ besteht aus:

- Code zur Auswertung der Anfrage $?g$;
- einer Instruktion no für Fehlschlag; sowie
- Code für die Prädikate rr_i.

Vor Auswertung der Anfrage müssen zusätzlich die Register korrekt initialisiert und ein erster Kellerrahmen auf dem Keller anglegt werden.

Danach muss die Ergebnis-Substitution zurück gegeben (oder Fehlschlag gemeldet) werden:

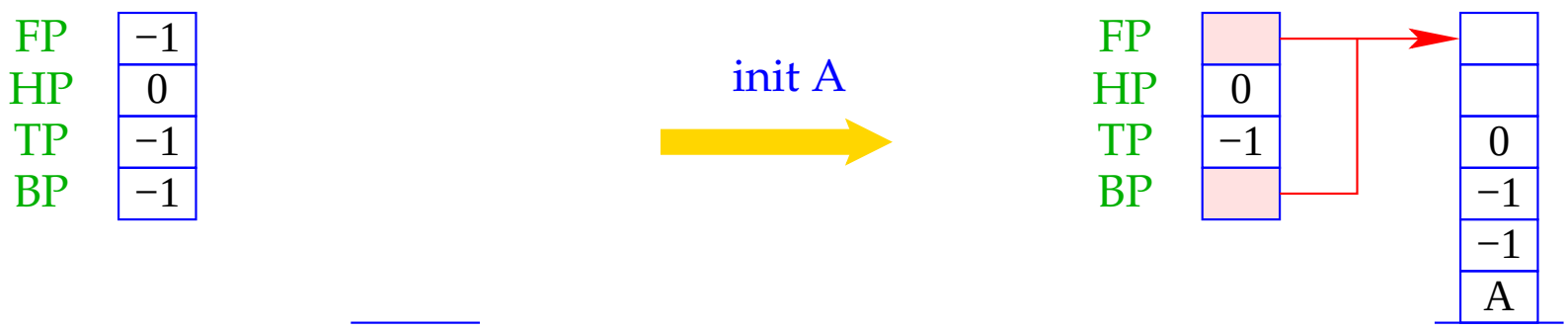
$$
\begin{array}{lcll}
\text{code } p & = & & \text{init A} \\
 & & & \text{pushenv d} \\
 & & & \text{code}_G\ g\ \rho \\
 & & & \text{halt d} \\
 & & \text{A:} & \text{no} \\
 & & & \text{code}_P\ rr_1 \\
 & & & \ldots \\
 & & & \text{code}_P\ rr_h
\end{array}
$$

wobei $free(g) = \{X_1, \ldots, X_d\}$ und die Adress-Umgebung $\rho\ X_i = i$ ist.

Die Instruktion halt d beendet die Programm-Ausführung und stellt die

- ... beendet die Programm-Ausführung;
- ... stellt die Bindungen der d Variablen der Welt zur Verfügung;
- ... löst backtracking aus – sofern von der Benutzerin gefordert :-)

Die Instruktion init A ist definiert durch:



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

An der Adresse "A" für das Fehlschlagen des Ziels haben wir die Instruktion
no untergebracht, welche *no* auf die Standard-Ausgabe schreibt und dann hält.

Das ultimative Beispiel:

$$t(X) \leftarrow \bar{X} = b \qquad q(X) \leftarrow s(\bar{X}) \qquad s(X) \leftarrow \bar{X} = a$$

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

Die Übersetzung liefert:

```
        init N          popenv          q/1:  pushenv 1     E:    pushenv 1
        pushenv 0   p/0: pushenv 1            mark D              mark G
        mark A          mark B                putref 1            putref 1
        call p/0        putvar 1              call s/1            call t/1
A:      halt 0          call q/1        D:    popenv        G:    popenv
N:      no          B:  mark C          s/1:  setbtp        F:    pushenv 1
t/1:    pushenv 1       putref 1              try E               putref 1
        putref 1        call t/1              delbtp              uatom a
        uatom b     C:  popenv                jump F              popenv
```

34 Letzte Ziele

Betrachte das Prädikat app aus der Einleitung:

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

Wir beobachten:

- Der rekursive Aufruf ist das letzte Ziel der Klausel.
- Ein solches Ziel ist ein letzter Aufruf :-)

  $\Longrightarrow$ wir versuchen, es im aktuellen Kellerrahmen auszuwerten !!!

  $\Longrightarrow$ nach (erfolgreicher) Beendigung werden wir nicht mehr zum gegenwärtigen Aufrufer zurückkehren !!!

Betrachten wir die Klausel r: $p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$
mit m lokalen Variablen, wobei $g_n \equiv q(t_1, \ldots, t_h)$. Das Zusammenspiel
zwischen code_C und code_G:

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

Ersetzung:	mark B	$\Longrightarrow$	lastmark
	call q/h; popenv	$\Longrightarrow$	lastcall q/h m

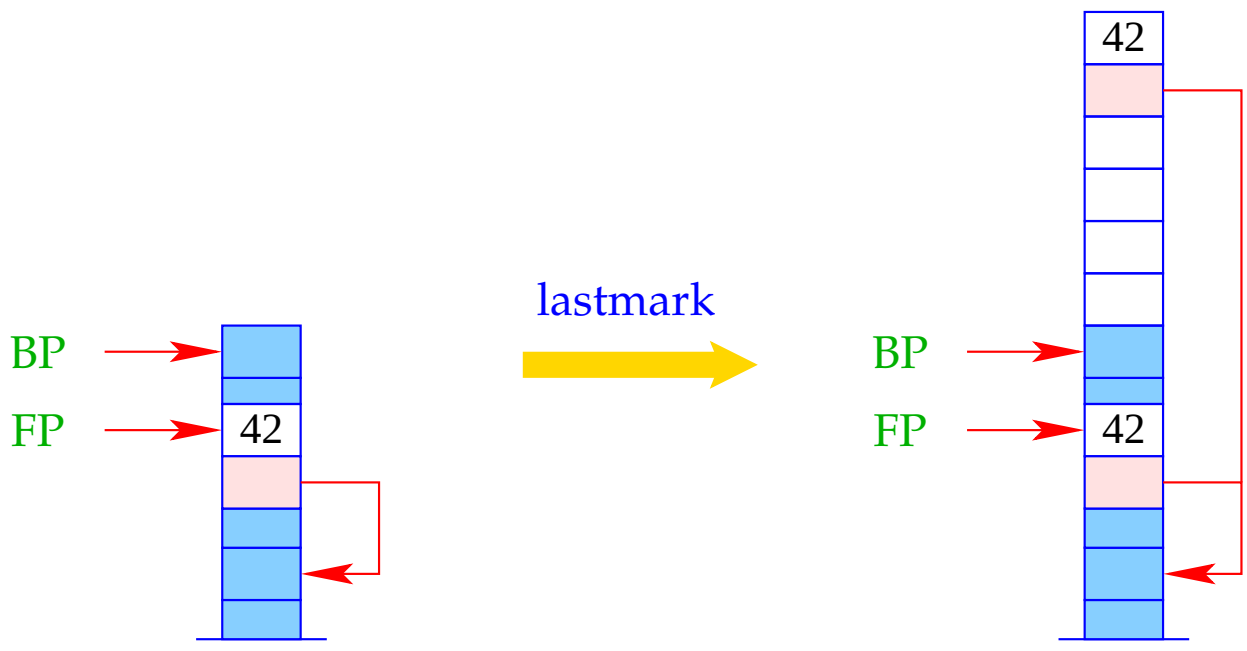
Betrachten wir die Klausel r: $p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$
mit m lokalen Variablen, wobei $g_n \equiv q(t_1, \ldots, t_h)$. Das Zusammenspiel
zwischen code_C und code_G:

$$
\begin{aligned}
\text{code}_C\ r \quad = \quad & \text{pushenv m} \\
& \text{code}_G\ g_1\ \rho \\
& \ldots \\
& \text{code}_G\ g_{n-1}\ \rho \\
& \text{lastmark} \\
& \text{code}_A\ t_1\ \rho \\
& \ldots \\
& \text{code}_A\ t_h\ \rho \\
& \text{lastcall}\ \ \text{q/h}\ \ \text{m}
\end{aligned}
$$

Ersetzung:	mark B	$\Longrightarrow$	lastmark
	call q/h; popenv	$\Longrightarrow$	lastcall q/h m

Falls die gegenwärtige Klausel nicht letzt ist oder die $g_1, \ldots, g_{n-1}$ Rücksetz-Punkte erzeugt haben, ist FP $\leq$ BP :-)

lastmark legt einen neuen Rahmen an mit einer Referenz auf den Vorgänger:




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

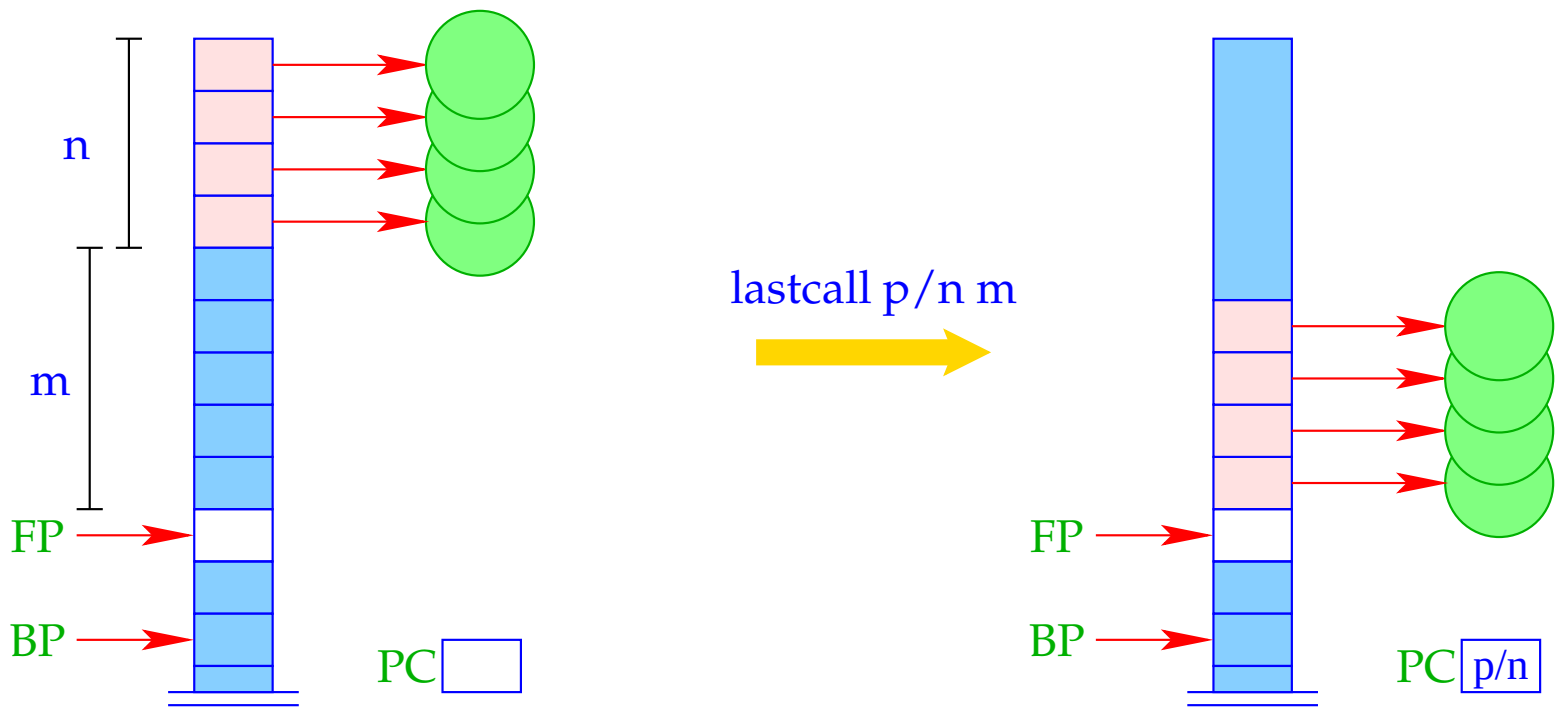
Falls FP $>$ BP dann tut lastmark nichts :-)

Falls $FP \leq BP$, führt lastcall p/h m ein normales call p/h aus.

Andernfalls werden die aktuellen Parameter in den Zellen S[FP+1], S[FP+2], . . ., S[FP+h] ausgetauscht und p/h angesprungen.

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

Die Differenz zwischen den alten und neuen Adressen der verschobenen Parameter $m = SP - h - FP$ ist gerade gleich der Anzahl der lokalen Variablen im Kellerrahmen.


n
m
FP
BP
PC
lastcall p/n m
FP
BP
PC
p/n

Beispiel:

Betrachten wir die Klausel r, von eben:

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

Die Optimierung letzter Ziele liefert für $\text{code}_C\ r$:

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

Beachte: Ist das letzte Literal der letzten Klausel gleichzeitig das einzige in dieser Klausel, können wir auf lastmark verzichten und lastcall p/h m durch die Folge move m h; jump p/h ersetzen.

Beispiel:

Betrachte die letzte Klausel des Prädikats app:

$$\mathsf{app}(X, Y, Z) \quad \leftarrow \quad \bar{X} = [H|X'],\ \bar{Z} = [\bar{H}|Z'],\ \mathsf{app}(\bar{X}', \bar{Y}, \bar{Z}')$$

Hier ist das letzte Ziel das einzige :-) Folglich erhalten wir:

```
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

D:  uref 4
    son 2
    uvar 6
    up E
    check 4
    putref 4
    putvar 6
    putstruct [|]/2

E:  bind
    putref 5
    putref 2
    putref 6
    move 6 3
    jump app/3
```

35 Verkleinerung von Kellerrahmen

Idee:

- Ordne lokale Variablen gemäß ihrer Lebensdauer;
- Beseitige tote Variablen — falls möglich :-}

35 Verkleinerung von Kellerrahmen

Idee:

- Ordne lokale Variablen gemäß ihrer Lebensdauer;
- Beseitige tote Variablen — falls möglich :-}

Beispiel:

Betrachte die Klausel:

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

35 Verkleinerung von Kellerrahmen

Idee:

- Ordne lokale Variablen gemäß ihrer Lebensdauer;
- Beseitige tote Variablen — falls möglich :-}

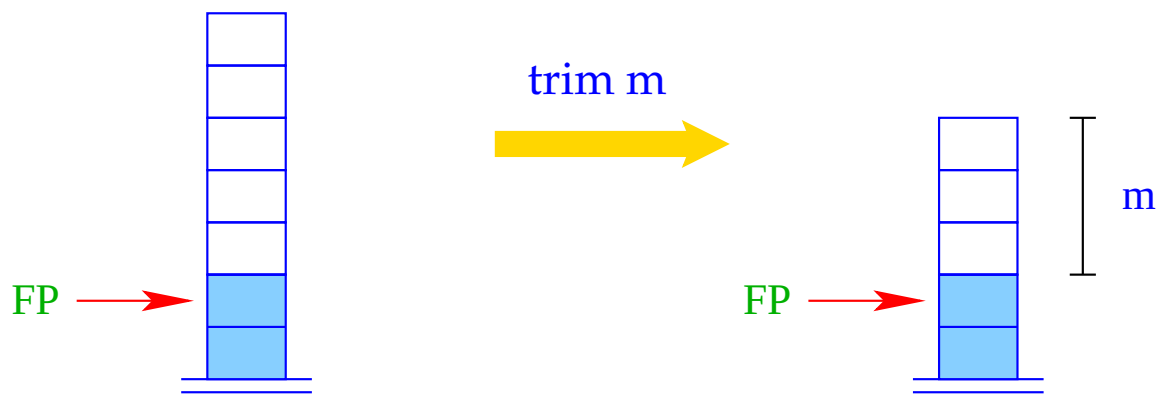
Beispiel:

Betrachte die Klausel:

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

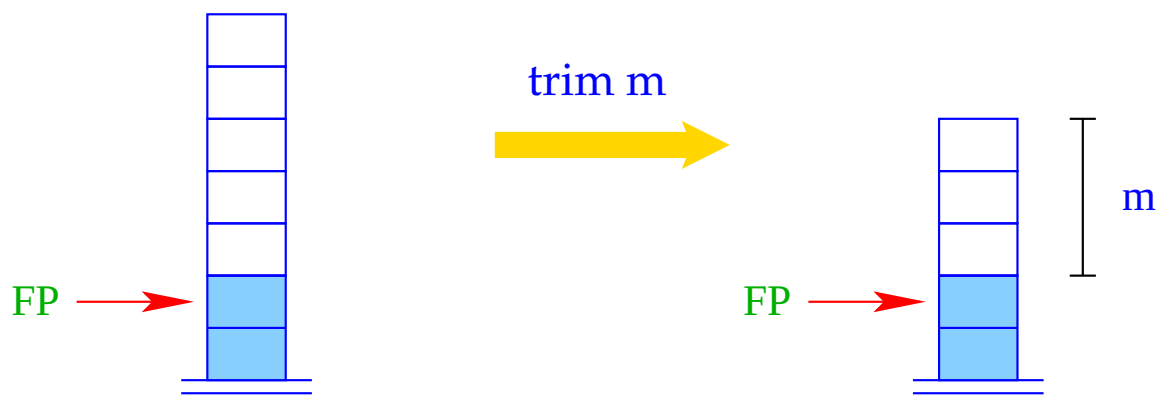
Nach der Anfrage $\mathsf{p}_2(\bar{X}_1, X_2)$ ist die Variable X_1 tot.
Nach der Anfrage $\mathsf{p}_3(\bar{X}_2, X_3)$ ist die Variable X_2 tot :-)

Nach jedem nicht-letzten Ziel mit toten Variablen fügen wir Instruktionen
trim m ein:



if (FP $\geq$ BP)
    SP = FP + m;

Nach jedem nicht-letzten Ziel mit toten Variablen fügen wir die Instruktion trim ein:



if (FP $\geq$ BP)
SP = FP + m;

Die toten lokalen Variablen dürfen nur eliminiert werden, wenn keine Rücksetz-Punkte angelegt wurden :-)

Beispiel (Forts.):

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

Anordnung der Variablen:

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

Der resultierende Code:

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 $\mathsf{p}_2/2$		call $\mathsf{p}_3/2$	lastcall $\mathsf{p}_4/2$ 3
call $\mathsf{p}_1/2$	B:	trim 4	C:	trim 3	

36 Klausel-Indizierung

Beobachtung:

Oft werden Prädikate durch Fallunterscheidung nach dem ersten Argument definiert.

$\Longrightarrow$ Berücksichtigung des ersten Arguments kann viele Alternativen ausschließen :-)

$\Longrightarrow$ Fehlschlag wird früher entdeckt :-)

$\Longrightarrow$ Rücksetz-Punkte werden früher beseitigt :-))

$\Longrightarrow$ Kellerrahmen werden früher gepoppt :-)))

Beispiel: Das app-Prädikat:

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

- Falls der Wurzel-Konstruktor $[\,]$ ist, ist nur die erste Klausel anwendbar.
- Falls der Wurzel-Konstruktor $[|]$ ist, ist nur die zweite Klausel anwendbar.
- Jeder andere Wurzel-Konstruktor sollte fehlschlagen !!
- Nur wenn das erste Argument eine ungebundene Variable ist, müssen beide Alternativen probiert werden ;-)

Idee:

- Führe getrennte Try-Ketten für jeden möglichen Konstruktor ein.
- Besichtige den Wurzelknoten des ersten Arguments.
- Abhängig vom Ergebnis, führe einen indizierten Sprung zu der entsprechenden Kette durch.

Angenommen, das Prädikat p/k sei durch die Folge rr von Klauseln $r_1 \ldots r_m$ definiert.

Sei tchains rr die Folge der Try-Ketten entsprechend den Wurzel-Konstruktoren in Unifikationen $X_1 = t$.

Beispiel:

Betrachten wir erneut das app-Prädikat und nehmen an, der Code der beiden Klauseln beginne an den Adressen A_1 und A_2.

Dann erhalten wir die folgenden vier Ketten:

```
VAR:    setbtp      // Variablen     NIL:        jump A1   // Atom [ ]
        try A1
        delbtp                        CONS:       jump A2   // Konstruktor [|]
        jump A2
                                      DEFAULT:    fail      // Default
```

Beispiel:

Betrachten wir erneut das app-Prädikat und nehmen an, der Code der beiden Klauseln beginne an den Adressen A_1 und A_2.

Dann erhalten wir die folgenden vier Ketten:

VAR:	setbtp	// Variablen	NIL:	jump A_1	// Atom [ ]
	try A_1				
	delbtp		CONS:	jump A_2	// Konstruktor [|]
	jump A_2				
			DEFAULT:	fail	// Default

Die neue Instruktion fail ist für alle Konstruktoren außer [ ] und [|] zuständig ...

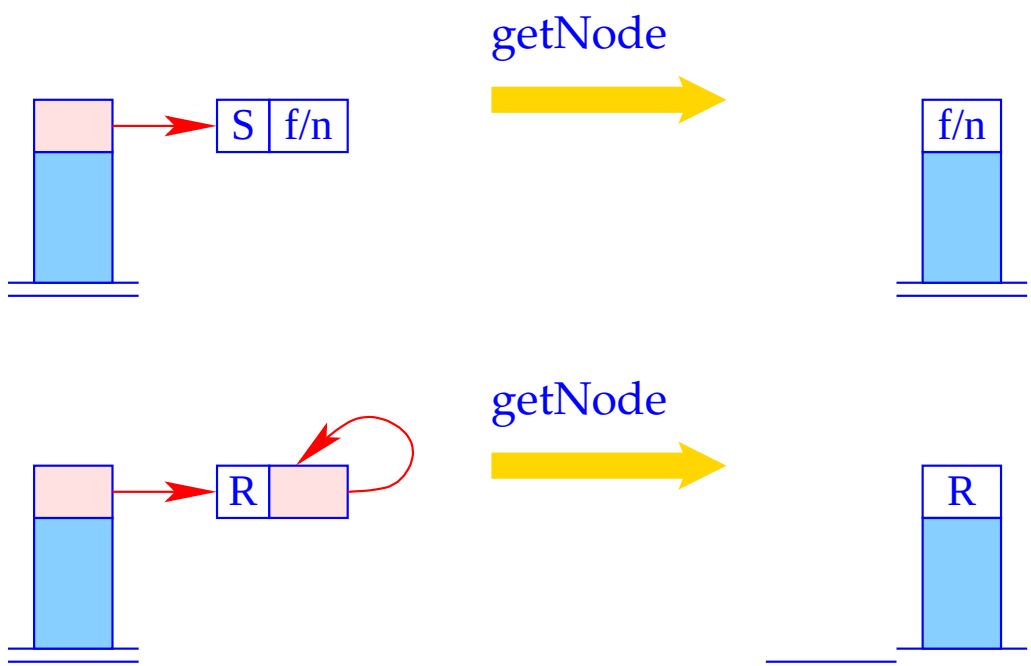
fail = `backtrack()`

... löst direkt backtracking aus :-)

Dann erzeugen wir für das Prädikat p/k:

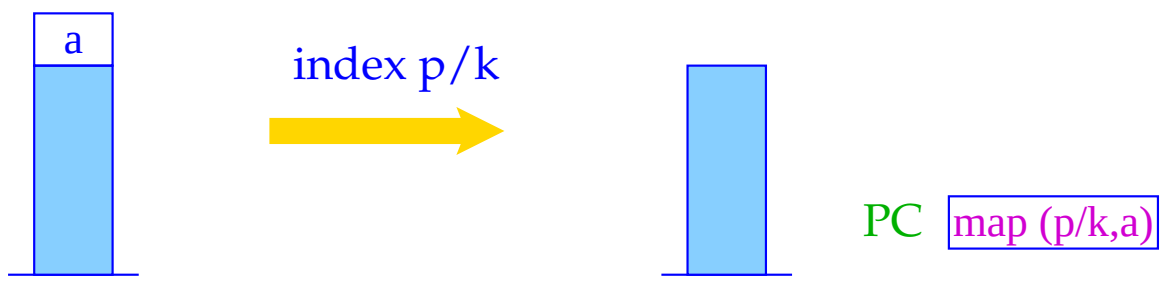
$$
\begin{array}{llll}
\text{code}_P\ rr & = & & \text{putref } 1 \\
 & & & \text{getNode} \qquad \text{// extrahiert die Wurzel-Beschriftung} \\
 & & & \text{index p/k} \qquad \text{// springe zum Try-Block} \\
 & & & \text{tchains } rr \\
 & & A_1 : & \text{code}_C\ r_1 \\
 & & & \ldots \\
 & & A_m : & \text{code}_C\ r_m
\end{array}
$$

Der Befehl getNode liefert “R”, falls der Verweis oben auf dem Keller auf eine ungebundene Variable zeigt. Andernfalls liefert er den Inhalt des Heap-Objekts:



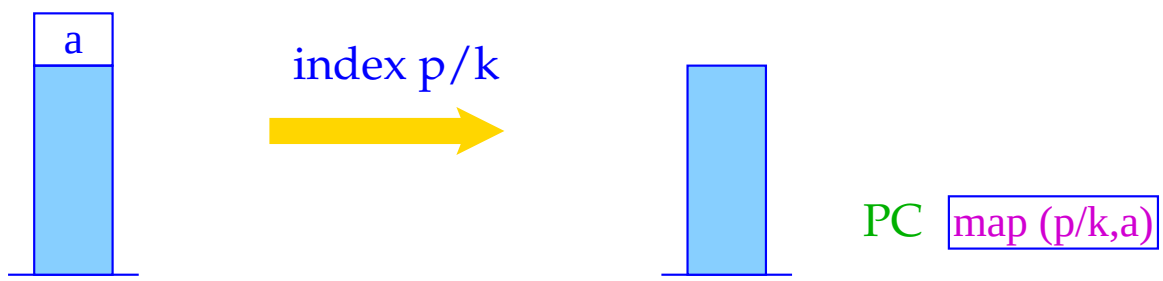

```
switch (H[S[SP]]) {
case (S, f/n):     S[SP] = f/n; break;
case (A,a):        S[SP] = a; break;
case (R,_) :       S[SP] = R;
}
```

Der Befehl index p/k führt einen indizierten Sprung an die entsprechende Try-Kette durch:



PC = map (p/k,S[SP]);
SP– –;

Der Befehl index p/k führt einen indizierten Sprung an die entsprechende Try-Kette durch:



PC = map (p/k,S[SP]);
SP– –;

Die Funktion map() liefert zu gegebenem Prädikat und Knoten-Inhalt die Start-Adresse der entsprechenden Try-Kette :-)

Sie wird typischerweise mit einer Hash-Tabelle implementiert :-)