Script **generated by TTT**

Title: Petter: Virtual Machines (04.06.2019)

Date: Tue Jun 04 10:16:01 CEST 2019

Duration: 76:01 min

Pages: 13

For more comprehensible notation, we thus introduce the macros:

$$\begin{aligned} \text{posCont} &\equiv S[\text{FP}] \\ \text{FPold} &\equiv S[\text{FP}-1] \\ \text{HPold} &\equiv S[\text{FP}-2] \\ \text{TPold} &\equiv S[\text{FP}-3] \\ \text{BPold} &\equiv S[\text{FP}-4] \\ \text{negCont} &\equiv S[\text{FP}-5] \end{aligned}$$

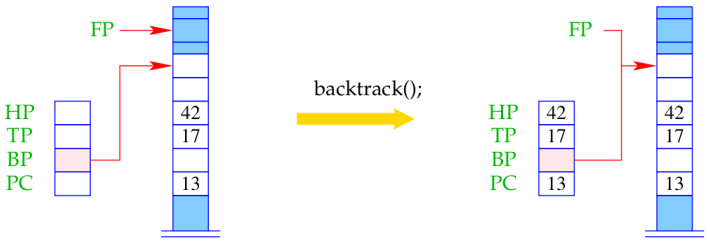
for the corresponding addresses.

Remark

Occurrence on the left ⟹ saving the register

Occurrence on the right ⟹ restoring the register

Calling the run-time function `void backtrack()` yields:




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

where the run-time function `reset()` undoes the bindings of variables established since the backtrack point.

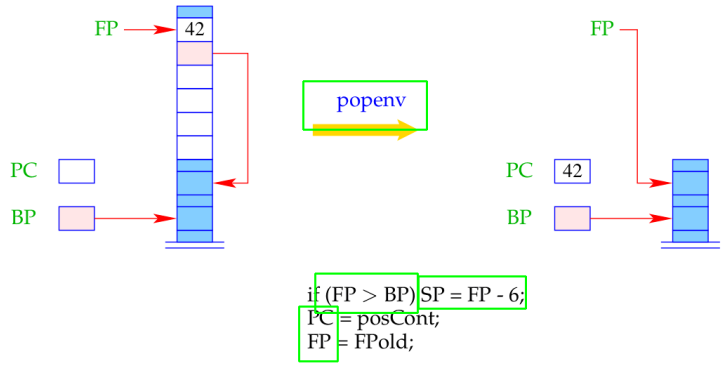
Functions `void trail(ref u)` and `void reset (ref y, ref x)` can thus be implemented as:

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

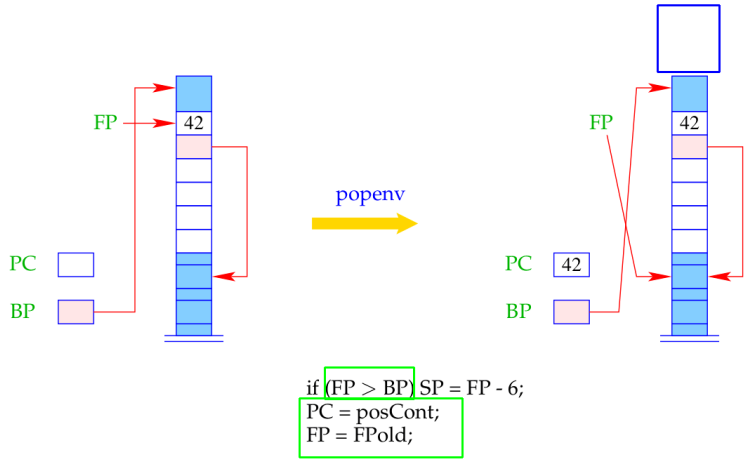
```
void reset (ref x, ref y) {
   for (ref u=y; x<u; u--)
      H[T[u]] = (R,T[u]);
}
```

Here, `S[BP-2]` represents the heap pointer when creating the last backtrack point.

The instruction popenv restores the registers FP and PC and possibly pops the stack frame:

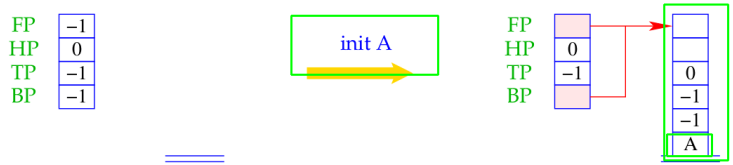


Caveat popenv may fail to de-allocate the frame !!!



If popping the stack frame fails, new data are allocated on top of the stack. When returning to the frame, the locals still can be accessed through the FP!

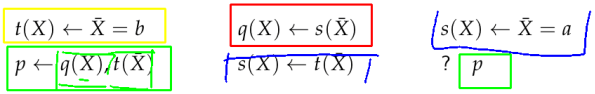
The instruction init A is defined by:



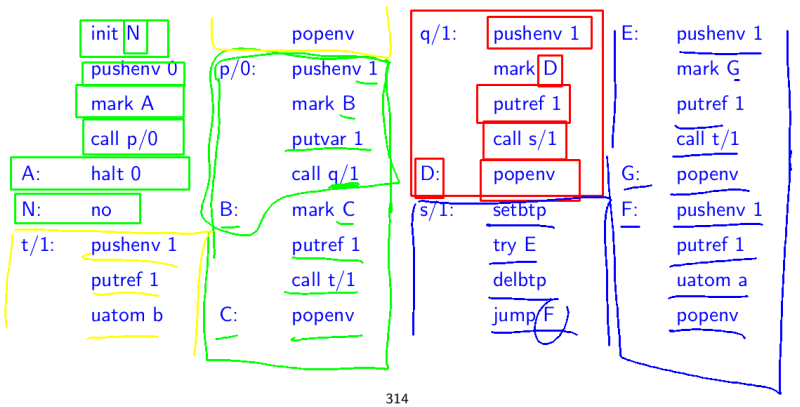
BP = FP = SP = 5;
S[0] = A;
S[1] = S[2] = -1;
S[3] = 0;
BP = FP;

At address "A" for a failing goal we have placed the instruction no for printing no to the standard output and halt.

The Final Example



The translation yields:



Consider a clause r: $p(X_1, \ldots, X_k) \leftarrow g_1, \ldots, g_n$
with m locals where $g_n \equiv q(t_1, \ldots, t_h)$. The interplay between code_C and code_G:

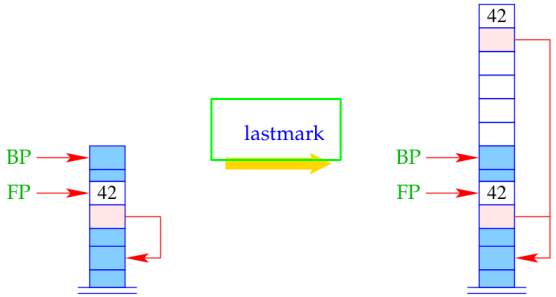
$\text{code}_C\ r\ =$

```
pushenv m
code_G g_1 ρ
...
code_G g_{n-1} ρ
lastmark
code_A t_1 ρ
...
code_A t_h ρ
lastcall q/h m
```

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

If the current clause is not last or the $g_1, \ldots, g_{n-1}$ have created backtrack points, then $\text{FP} \leq \text{BP}$.

Then lastmark creates a new frame but stores a reference to the predecessor:




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

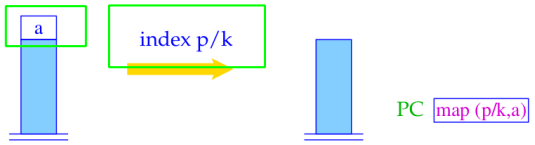
If $\text{FP} > \text{BP}$ then lastmark does nothing.

Example The app-predicate:

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

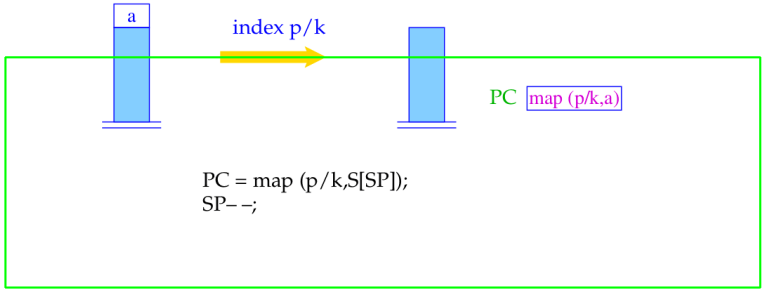
- If the root constructor is [ ], only the first clause is applicable.
- If the root constructor is [|] only the second clause is applicable.
- Every other root constructor should fail !!
- Only if the first argument equals an unbound variable, both alternatives must be tried ;-)

The instruction index p/k performs an indexed jump to the appropriate try chain:




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

The instruction index p/k performs an indexed jump to the appropriate try chain:



The function map() returns, for a given predicate and node content, the start address of the appropriate try chain.

It typically is defined through some hash table ...

38 Extension: The Cut Operator

Realistic Prolog additionally provides an operator "!" (cut) which explicitly allows to prune the search space of backtracking.

Example

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

Once the queries before the cut have succeeded, the choice is committed:

Backtracking will return only to backtrack points preceding the call to the left-hand side ...