Script **generated by TTT**

Title: Petter: Virtual Machines (03.06.2019)

Date: Mon Jun 03 10:30:31 CEST 2019

Duration: 67:57 min

Pages: 8

- The unification code performs a pre-order traversal over t.
- In case, execution hits at an unbound variable, we switch from checking to building.

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

```
ustruct f/n A          // test
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   // occur-check
    code_A f(t_1,...,t_n) ρ         // building !!
    bind                            // creation of bindings
B : ...
```


The Building Block

Before constructing the new (sub-) term t' for the binding, we must exclude that it contains the variable X' on top of the stack !!!

This is the case iff the binding of no variable inside t' contains (a reference to) X'.

$\Longrightarrow$ $ivars(t')$ returns the set of already initialized variables of t'.

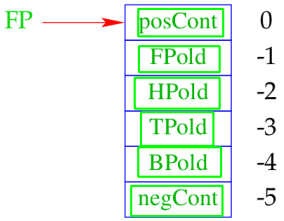
$\Longrightarrow$ The macro check $\{Y_1,\ldots,Y_d\}\ \rho$ generates the necessary tests on the variables $Y_1,\ldots,Y_d$:

$$\text{check}\ \{Y_1,\ldots,Y_d\}\ \rho \quad = \quad \begin{array}{l}\text{check}\ (\rho\ Y_1)\\ \text{check}\ (\rho\ Y_2)\\ \ldots\\ \text{check}\ (\rho\ Y_d)\end{array}$$

A backtrack point is stack frame to which program execution possibly returns.

- We need the code address for trying the next alternative (negative continuation address);
- We save the old values of the registers HP, TP and BP.
- Note: The new BP will receive the value of the current FP.

For this purpose, we use the corresponding four organizational cells:



For more comprehensible notation, we thus introduce the macros:

posCont	≡	$S[FP]$
FPold	≡	$S[FP-1]$
HPold	≡	$S[FP-2]$
TPold	≡	$S[FP-3]$
BPold	≡	$S[FP-4]$
negCont	≡	$S[FP-5]$

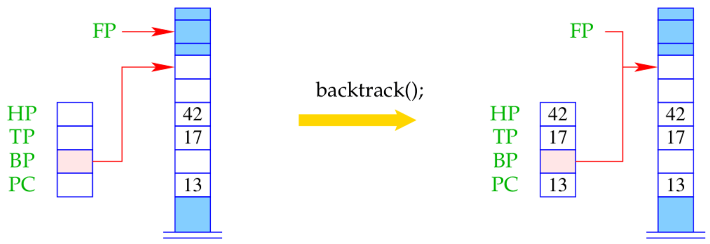
for the corresponding addresses.

Remark

Occurrence on the left ═══ saving the register

Occurrence on the right ═══ restoring the register

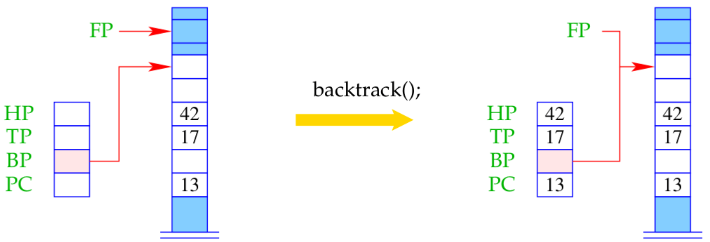
Calling the run-time function `void backtrack()` yields:




```
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.

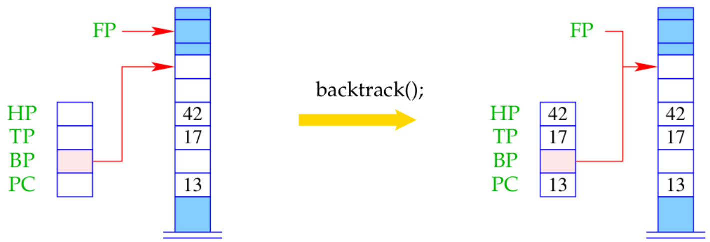
Calling the run-time function `void backtrack()` yields:



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.

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.

33.2 Trailing and Resetting Variables

Idea

- The variables which have been created since the last backtrack point can be removed together with their bindings by popping the heap !!
- Bindings of variables in the old heap section, though, must be reset explicitly.
- These are therefore recorded in the trail.