PSPP: Programmiersprache mit parametrisierten Prozeduren

Syntax (nur Änderungen gegenüber PSP)

$$
\begin{array}{llll}
\text{Deklarationen } \mathit{Decl} : \Delta & ::= & \Delta_C \Delta_V \Delta_P & \\
\Delta_P & ::= & \varepsilon \mid & \\
 & & \textbf{proc } I \underbrace{(I_1, \ldots, I_p; \textbf{var} J_1, \ldots J_q)}_{\text{Wert- und Variablenparameter}}; B; & \\
 & & \ldots & \\
\text{Anweisungen } \mathit{Cmd} : \Gamma & ::= & I := E \mid I(E_1, \ldots, E_p; V_1, \ldots, V_q) & \\
 & & \mid \Gamma_1; \Gamma_2 & \\
 & & \mid \textbf{if } BE \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2 & \\
 & & \mid \textbf{while } BE \textbf{ do } \Gamma &
\end{array}
$$

Semantik

Prozedurdeklarationen: Formale Parameter werden wie (in der Umgebung des Prozedurrumpfs) lokal deklarierte Variablen behandelt.

Prozeduraufrufe:

- Wertparameter als lokale Variable (neuer Speicherplatz)
- Variablenparameter durch vorhandenen Speicherplatz aktualisierung (Zeiger anlegen)

MPP - eine Stackmaschine für PSPP

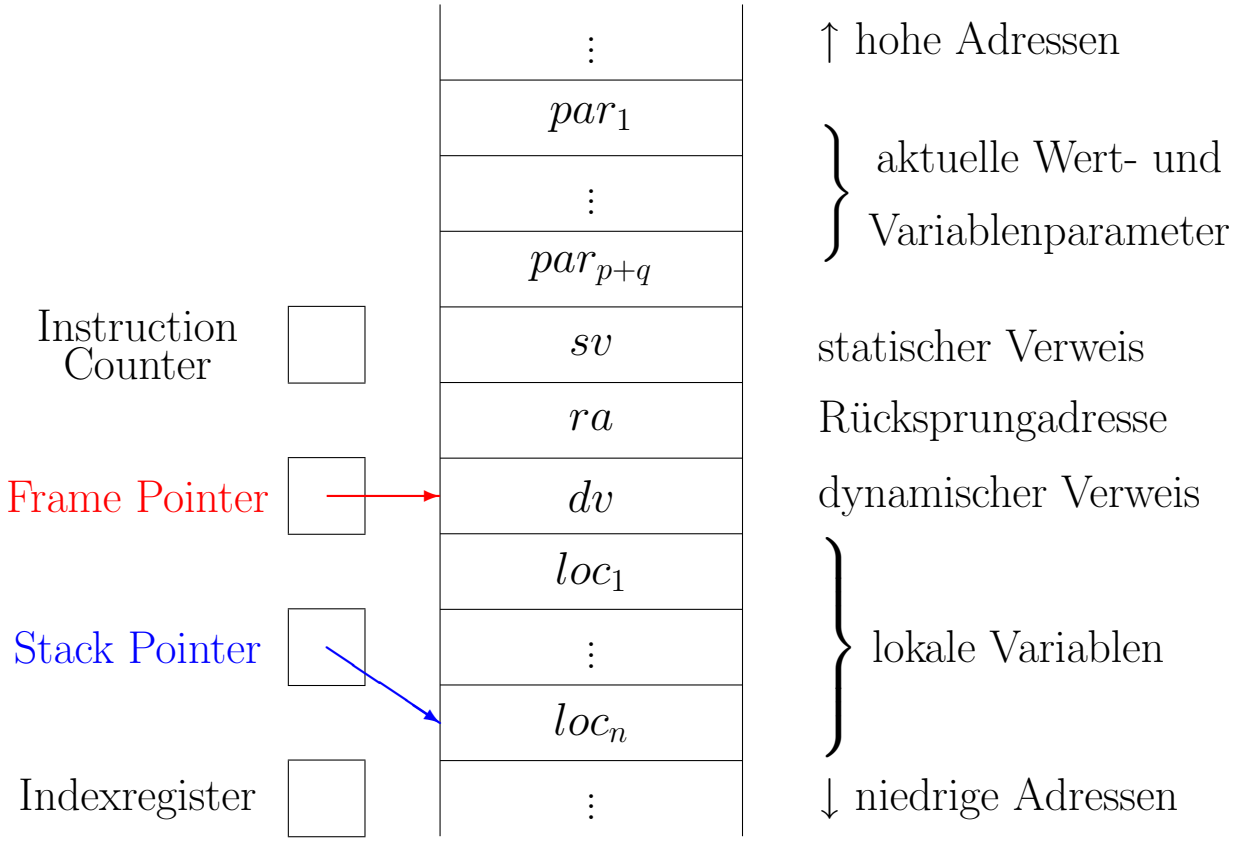
Realistischeres Maschinenmodell mit nur noch einem Keller für Daten und Aktivierungsblöcke.

Zustandsraum:

$$ZR_{MPP} := STACK \times IC \times SP \times FP \times R$$

mit

- $STACK := [SAdr \to \mathbb{Z}]; SAdr := \mathbb{Z}$.
- $IC := Adr$ (instruction counter)
- $SP := FP := SAdr$ (stack pointer/frame pointer)
- $R := SAdr$ Indexregister (Berechnung statischer Verweise)



Befehlssatz der MPP:

arithmetisch/logische und Sprungbefehle:

wie bisher in MA bzw. MP

Erweiterung: SP setzen

Keller- und Lade-/Speicherbefehle:

- bisher:
  CALL(ca, $diff$, lv), RET, LOAD ($diff$, off), STORE ($diff$, off)
- jetzt:

Befehl	Bedeutung
CALL ca	$SP \leftarrow SP-1; \langle SP\rangle \leftarrow IC+1; IC \leftarrow ca$
RET k	$IC \leftarrow \langle SP\rangle; SP \leftarrow SP + \underbrace{k+1}_{\text{(pars/sv)+ra}}$
PUSH Z	$SP \leftarrow SP-1; \langle SP\rangle \leftarrow Z$
PUSH FP	$SP \leftarrow SP-1; \langle SP\rangle \leftarrow FP$
PUSH $\langle FP\rangle$	$SP \leftarrow SP-1; \langle SP\rangle \leftarrow \langle FP\rangle$
PUSH $R+2$	$SP \leftarrow SP-1; \langle SP\rangle \leftarrow R+2$
POP FP	$FP \leftarrow \langle SP\rangle; SP \leftarrow SP+1$
POP $\langle n\rangle$	$Stack[n] \leftarrow \langle SP\rangle; SP \leftarrow SP+1$
SUB SP, n	$SP \leftarrow SP-n$
LOAD FP, SP	$FP \leftarrow SP$
LOAD $R, \langle n\rangle$	$R \leftarrow Stack[n]$
LOAD $R, \langle R+2\rangle$	$R \leftarrow \langle R+2\rangle$

Übersetzungsfunktionen

$$trans : \textit{PSPP-Prog} \dashrightarrow \textit{MPP-Code}$$

$trans(\textbf{in/out}\ I_1, \ldots, I_n;\ B)$

$:=$ $1:$ `PUSH` FP %sv

$2:$ `CALL` a_Γ %ra

$3:$ `JMP` $0;$ %$STOP$

$bt(B, st_{I/O}, a_\Gamma, 1, \underbrace{0}_{\#Params}).$

$bt(\Delta\Gamma, st, a, l, r)$

$:= dt(\Delta, up(\Delta, st, a_1, l), a_1, l)$

$a:$ `PUSH` $FP;$
$\_:$ `LOAD` $FP, SP;$
$\_:$ `SUB` $SP, size(\Delta\Gamma);$

Entry-Code: Anlegen von *dl* & Platz für lok. Vars.

$ct(\Gamma, up(\Delta, st, a_1, l), a+3, l)$

$\_:$ `LOAD` $SP, FP;$
$\_:$ `POP` $FP;$
$\_:$ `RET` $r+1;$

Exit-Code: Freigabe des akt. Akt.-Block mit Rücksprung

$dt(\textbf{proc}\ I(I_1, \ldots, I_p; \textbf{var}\ J_1, \ldots J_q); B; \ldots, st, a, l)$

$:= \textbf{if}\ \textit{diff\_id}\ (I_1, \ldots, I_p, J_1, \ldots J_q, B)\ \textbf{and}\ \textit{diff\_id}\ (\ldots)$

$\textbf{then}\ \ bt(B, \tilde{st}, a_1, l+1, p+q)bt(\ldots)\ldots$

where

$$\tilde{st} := st[\ I_1/(var, l+1, p+q+2), \ldots, I_p/(var, l+1, q+3),$$
$$J_1/(vpar, l+1, q+2), \ldots, J_q/(vpar, l+1, 3)]$$

Hier gilt jeweils:

Parameterlevel $\simeq$ Blocklevel $\simeq$ Level lokaler Variablen

Übersetzung von Anweisungen

Hilfsfunktion zum Dereferenzieren der statischen Verweiskette:

$$deref(R, FP, k, a) := \begin{array}{l} a: \texttt{LOAD}\ R, \langle FP+2\rangle; \\ \left.\begin{array}{l} \_: \texttt{LOAD}\ R, \langle R+2\rangle; \\ \vdots \\ \_: \texttt{LOAD}\ R, \langle R+2\rangle; \end{array}\right\} k\text{-mal} \end{array}$$

$$\begin{array}{l} ct(I := E, st, a, l) := et(E, st, a, l) \\ \quad \textbf{if } type(E) = int \textbf{ then} \\ \quad\quad \textbf{if } st(I) = (var, dl, o) \textbf{ then} \\ \quad\quad\quad \textbf{if } l = dl \textbf{ then } \_: \texttt{POP}\ \langle FP+o\rangle \\ \quad\quad\quad\quad\quad\quad \textbf{else } deref(R, FP, l-dl-1, a') \\ \quad\quad\quad\quad\quad\quad\quad \_: \texttt{POP}\ \langle R+o\rangle \\ \quad\quad \textbf{else if } st(I) = (vpar, dl, o) \textbf{ then} \\ \quad\quad\quad \textbf{if } l = dl \textbf{ then } \_: \texttt{LOAD}\ R, \langle FP+o\rangle; \\ \quad\quad\quad\quad\quad\quad\quad \_: \texttt{POP}\ \langle R\rangle \\ \quad\quad\quad\quad\quad\quad \textbf{else } deref(R, FP, l-dl-1, a'') \\ \quad\quad\quad\quad\quad\quad\quad \_: \texttt{LOAD}\ R, \langle R+o\rangle; \\ \quad\quad\quad\quad\quad\quad\quad \_: \texttt{POP}\ \langle R\rangle \end{array}$$

Aufbau eines Aktivierungsblocks:

Code für Prozeduraufruf:

1. Berechnung der aktuellen Parameter
2. Berechnung des statischen Verweises mit dem Indexregister
3. Sprung zur Prozedur mit Ablage der Rücksprungadresse

Code für Prozedur:

4. Alten FP als dynamischen Verweis speichern
5. Speicherplatz für lokale Variablen bereitstellen

$ct(I(E_1, \ldots, E_p; V_1, \ldots, V_q), st, a, l)$

$:=$ **if** $st(I) = (proc, ca, dl, lv)$

and $type(E_i) = int\ (1 \leq i \leq p)$

and $st(V_j) = (var, l_j, o_j)\ (1 \leq j \leq q)$

then

$et(E_1, st, a, l)$
$\vdots$
$et(E_p, st, a, l)$
} Wertparameter

if $l = l_1$ **then** _ : `PUSH` $(FP + o_1)$

else $deref(R, FP, l - l_1 - 1, a')$

_ : `PUSH` $(R + o_1)$

$\vdots$

(* analog für $j = 2 \ldots q$ *)

if $l = dl$ **then** _ : `PUSH` FP

else $deref(R, FP, l - dl - 1, a'')$

_ : `PUSH` R;

`CALL` ca;

Ähnlich für Variablenparameter, also falls $st(V_j) = (vpar, l_j, o_j)$.

Übersetzung von Ausdrücken

In der Übersetzungsfunktion et muss die Übersetzung von Bezeichnern angepasst werden.

$$
\begin{array}{lll}
et(I, st, a, l) := & \textbf{if } st(I) = (const, Z) \textbf{ then } \texttt{PUSH } Z; & \\
& \textbf{else if } st(I) = (var, dl, o) \textbf{ then} & \\
& \quad \textbf{if } l = dl \textbf{ then} & a : \texttt{PUSH } \langle FP + o \rangle \\
& \quad \textbf{else} & deref(R, FP, l - dl - 1, a') \\
& & \_ : \texttt{PUSH } \langle R + o \rangle \\
& \textbf{else if } st(I) = (vpar, dl, o) \textbf{ then} & \\
& \quad \textbf{if } l = dl \textbf{ then} & \_ : \texttt{LOAD } R, \langle FP + o \rangle; \\
& & \_ : \texttt{PUSH } \langle R \rangle \\
& \quad \textbf{else} & deref(R, FP, l - dl - 1, a'') \\
& & \_ : \texttt{LOAD } R, \langle R + o \rangle; \\
& & \_ : \texttt{PUSH } \langle R \rangle
\end{array}
$$

MPP-Programm:

```
 1:  PUSH FP;          % Aufruf des
 2:  CALL 26;          % Hauptprogramms
 3:  JMP  0;           % STOP
 4:  PUSH FP;          % Entry-Code von F
 5:  LOAD FP, SP;
 6:  SUB SP, 0;
 7:  LIT  1;           % Rumpf der Prozedur F
 8:  PUSH ⟨FP+4⟩;      % Lade X
 9:  LESS;
10:  JPFALSE 23;
11:  LOAD  R, ⟨FP+3⟩;
12:  PUSH  ⟨R⟩;        % Zugriff auf V
13:  PUSH  ⟨FP+4⟩;     % Lade X
14:  MULT;
15:  LOAD  R, ⟨FP+3⟩;
16:  POP ⟨R⟩;          % Zuweisung an V
17:  PUSH ⟨FP+4⟩;
18:  LIT  1;
19:  SUB;              % Wertparameter X−1
20:  PUSH ⟨FP+3⟩;      % Variablenparameter V
21:  PUSH FP;          % statischen Verweis speichern
22:  CALL  4;          % rekursiver Aufruf von F
23:  LOAD SP, FP;      % Exit-Code von F
24:  POP FP;
25:  RET 3;
26:  PUSH FP;          % Entry-Code des Hauptprogramms
27:  LOAD FP, SP;
28:  SUB SP, 1;
29:  LIT 1;            % Anweisungsteil des Hauptprogramms
30:  POP ⟨FP-1⟩;       % Zuweisung an E
31:  LOAD R, ⟨FP+2⟩;
32:  PUSH ⟨R−1⟩;       % Wertparameter X
33:  PUSH ⟨FP-1⟩;      % Variablenparameter E
34:  PUSH FP;          % statischen Verweis anlegen
35:  CALL 4;           % Aufruf von F
36:  PUSH ⟨FP-1⟩;
37:  LOAD R,⟨FP+2⟩;
38:  POP ⟨R-1⟩;        % Zuweisung an X
39:  LOAD SP, FP;      % Exit-Code des Hauptprogramms
40:  POP FP;
41:  RET 1;
```

PSPP-Programm:

```
in/out X;
  var E;
  proc F(X;var V);
    if 1<X then
      ( V:=V*X;
        F(X-1;V)
      );
  E:=1;
  F(X,E);
  X:=E.
```