Syntax von PSP — Programmiersprache mit Prozeduren

Ganze Zahlen	*Int*	:	Z		
Bezeichner	*Ide*	:	I		
Deklarationen	*Decl*	:	Δ	$::=$	$\Delta_C \Delta_V \Delta_P$
			Δ_C	$::=$	$\varepsilon \mid \textbf{const } I_1 = Z_1; \ldots; I_n = Z_n$ $(n \geq 1)$
			Δ_V	$::=$	$\varepsilon \mid \textbf{var } I_1, \ldots, I_n;$ $(n \geq 1)$
			Δ_P	$::=$	$\varepsilon \mid \textbf{proc } I_1; B_1; \ldots \textbf{proc } I_n; B_n;$ $(n \geq 1)$
Arithmetische Ausdrücke	*AExp*	:	E	$::=$	$Z \mid I \mid (E_1 \; aop \; E_2)$ $(aop \in \{+, -, *, \ldots\})$
Boolesche Ausdrücke	*BExp*	:	BE	$::=$	$E_1 \; relop \; E_2$ $\mid \textbf{not } BE \mid (BE_1 \textbf{ and } BE_2)$ $\mid (BE_1 \textbf{ or } BE_2)$ $(relop \in \{=, \neq, <, \ldots\})$
Anweisungen	*Cmd*	:	Γ	$::=$	$I := E \mid I() \mid \Gamma_1; \Gamma_2$ $\mid \textbf{if } BE \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2$ $\mid \textbf{while } BE \textbf{ do } \Gamma$
Blöcke	*Block*	:	B	$::=$	$\Delta\Gamma$
Programme	*Prog*	:	P	$::=$	$\textbf{in/out } I_1, \ldots, I_n; B$

Kontextsensitive Bedingungen:

- Bezeichner einer Deklaration Δ müssen paarweise verschieden sein.
- Ein im Anweisungsteil Γ eines Blocks $\Delta\Gamma$ verwendeter Bezeichner muss in Δ oder in der Deklarationsliste eines umschließenden Blocks deklariert sein.
- Mehrfachdeklaration eines Bezeichners ist auf verschiedenen Niveaus erlaubt: Die „innerste“ Deklaration ist für ein Auftreten gültig.

Der *Gültigkeitsbereich (engl.: scope)* eines Bezeichners bzw. einer Bezeichnerdeklaration ist der Teil des Programms, in dem sich ein angewandtes Vorkommen des Bezeichners auf diese Deklaration beziehen kann.

Static scope	bedeutet:	Beim Aufruf einer Prozedur ist ihre Deklarationsumgebung gültig.
Dynamic scope	bedeutet:	Beim Aufruf einer Prozedur ist ihre Aufrufumgebung gültig.

Beispiel:

```
in/out X;
    const C = 10;
    var Y;
    proc A;
        var  Y, Z;
        proc B;
            var X,Z;
            [... A() ...]
        [... B() ... D() ...]
    proc D;
        [... A() ...]
    [... A() ...].
```

1. **static scope:** Beim Prozeduraufruf `A()` im Anweisungsteil von `B` bezeichnet `X` jeweils die Ein-/Ausgabevariable `X` und `Z` die lokale Variable `Z` von `A`.
2. **dynamic scope:** Beim Prozeduraufruf `A()` im Anweisungsteil von `B` bezeichnen `X` und `Z` die lokalen Variablen von `B`.
3. `D` kann in `A` aufgerufen werden, obwohl die Deklaration textuell später erfolgt.

Semantik von PSP (Skizze)

Semantische Bereiche:

Speicherplätze	Loc	$:=$	$\{\alpha_1, \alpha_2, \alpha_3, \ldots\}$	(locations)
Zustandsraum	S	$:=$	$\{\sigma \mid \sigma : Loc \dashrightarrow \mathbb{Z}\}$	(states)
Speichertransf.	C	$:=$	$\{\theta \mid \theta : S \dashrightarrow S\}$	(continuations)
Umgebung	Env	$:=$	$\{\rho \mid \rho : Ide \dashrightarrow \mathbb{Z} \cup \mathsf{Loc} \cup \mathsf{C}\}$	(environment)

Deklarationssemantik

$$\mathcal{D} :: Decl \times Env \ \times \ S \dashrightarrow Env \ \times \ S$$

$$\mathcal{D}[\![\Delta_C \Delta_V \Delta_P]\!]\rho\sigma \quad := \quad \mathcal{D}[\![\Delta_P]\!](\mathcal{D}[\![\Delta_V]\!](\mathcal{D}[\![\Delta_C]\!]\rho)\ \sigma)$$

$$\mathcal{D}[\![\varepsilon]\!]\rho\ \sigma \quad := \quad \rho\ \sigma$$

$$\mathcal{D}[\![\textbf{const}\ I_1 = Z_1; \ldots; I_n = Z_n]\!]\rho\ \sigma$$

$$:= \quad \rho\,[I_1/Z_1, \ldots, I_n/Z_n]\,\sigma$$

$$\mathcal{D}[\![\textbf{var}\ I_1, I_2, \ldots, I_n]\!]\rho\ \sigma \quad := \quad \rho[I_1 \mapsto \alpha_{j+1}, \ldots, I_n \mapsto \alpha_{j+n}]$$

$$\sigma[\alpha_{j+1} \mapsto 0, \ldots, \alpha_{j+n} \mapsto 0]$$

wobei j höchster Index eines belegten Speicherplatzes in σ sei, falls $\sigma = \emptyset$, sei $j = 0$

$$\mathcal{D}[\![\textbf{proc}\ I_1; B_1; \ldots \textbf{proc}\ I_n; B_n;]\!]\rho\ \sigma$$

$$:= \quad \rho[I_1 \mapsto \theta_1, \ldots, I_n \mapsto \theta_n]\ \sigma$$

Dabei sei für $1 \leq i \leq n$:

$$\theta_i(\sigma) := \mathcal{BL}[\![B_i]\!]\rho[I_1 \mapsto \theta_1, \ldots, I_n \mapsto \theta_n]\ \sigma.$$

Dies definiert eine „static scope“-Semantik, da ρ die Deklarationsumgebung der Prozeduren ist.

Semantik von Anweisungen

$$\mathcal{C} :: Cmd \times Env \times S \dashrightarrow S$$

$$\mathcal{C}[\![I := E]\!]\rho\,\sigma \quad := \quad \sigma\,[\alpha \mapsto \mathcal{E}[\![E]\!]\rho\,\sigma] \quad \text{falls } \rho(I) = \alpha \in Loc$$

$$\mathcal{C}[\![I()]\!]\rho\,\sigma \quad := \quad \theta(\sigma) \text{ falls } \rho(I) = \theta \in C$$

$$\mathcal{C}[\![\Gamma_1;\Gamma_2]\!]\rho\,\sigma \quad := \quad \mathcal{C}[\![\Gamma_2]\!]\rho(\mathcal{C}[\![\Gamma_1]\!]\rho\,\sigma)$$

$$\mathcal{C}[\![\textbf{if } BE \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2]\!]\rho\,\sigma$$
$$:= \begin{cases} \mathcal{C}[\![\Gamma_1]\!]\rho\,\sigma & \text{falls } \mathcal{B}[\![BE]\!]\rho\,\sigma = \text{ true} \\ \mathcal{C}[\![\Gamma_2]\!]\rho\,\sigma & \text{falls } \mathcal{B}[\![BE]\!]\rho\,\sigma = \text{ false} \end{cases}$$

$$\mathcal{C}[\![\textbf{while } BE \textbf{ do } \Gamma]\!]\rho\,\sigma \ := \begin{cases} \mathcal{C}[\![\textbf{while } BE \textbf{ do } \Gamma]\!]\rho(\mathcal{C}[\![\Gamma]\!]\rho\,\sigma) & \\ \qquad\qquad \text{falls } \mathcal{B}[\![BE]\!]\rho\,\sigma = \text{ true} \\ \sigma \qquad\quad \text{falls } \mathcal{B}[\![BE]\!]\rho\,\sigma = \text{ false} \end{cases}$$

Blocksemantik $\mathcal{BL} :: Block \times Env \times S \dashrightarrow S$

$$\mathcal{BL}[\![\Delta\Gamma]\!]\rho\,\sigma := \mathcal{C}[\![\Gamma]\!](\mathcal{D}[\![\Delta]\!]\rho\,\sigma)$$

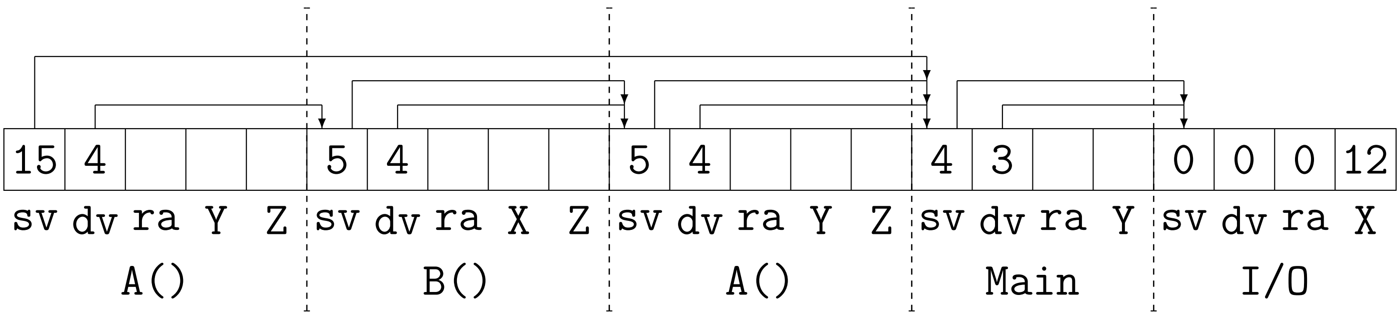
Programmsemantik $\mathcal{M} :: Prog \times \mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$

$$\mathcal{M}[\![\textbf{in/out } I_1,\ldots,I_n; B]\!](z_1,\ldots,z_n) := (\sigma(\alpha_1),\ldots,\sigma(\alpha_n)) \text{ mit}$$
$$\sigma := \mathcal{BL}[\![B]\!]\underbrace{\rho_\emptyset[I_1 \mapsto \alpha_1,\ldots,I_n \mapsto \alpha_n]}_{\text{Anfangsumgebung}}\underbrace{\sigma_\emptyset[\alpha_1 \mapsto z_1,\ldots,\alpha_n \mapsto z_n]}_{\text{Anfangszustand}}$$

Struktur des Laufzeitkellers

```
in/out X;
    const C = 10;
    var Y;
    proc A;
        var  Y, Z;
        proc B;
            var X,Z;
            [... A() ...]
        [... B() ... D() ...]
    proc D;
        [... A() ...]
    [... A() ...].
```

Bei der Ausführung des Programms hat der Prozedurkeller nach dem zweiten Aufruf von `A` die folgende Struktur:



Beispielübersetzung

Gegeben sei das folgende Programm:

$$
\begin{array}{l}
\textbf{in/out}\ \mathtt{X;}\\
\Delta\left\{\begin{array}{l}
\textbf{var}\ \mathtt{E;}\\
\textbf{proc}\ \mathtt{F;}\\
\Gamma_F\left\{\begin{array}{l}
\textbf{if}\ \mathtt{1<X}\ \textbf{then begin}\\
\qquad \mathtt{E:=E*X;}\\
\qquad \mathtt{X:=X-1;}\\
\qquad \mathtt{F()}\\
\textbf{end;}
\end{array}\right.
\end{array}\right.\\
\Gamma\left\{\begin{array}{l}
\mathtt{E:=1;}\\
\mathtt{F();}\\
\mathtt{X:=E.}
\end{array}\right.
\end{array}
$$

Es gilt $st_{I/O}(X) = (var, 0, 1)$ und

$$
\boxed{
\begin{array}{rcl}
trans(\textbf{in/out}\ \mathtt{X};\ \Delta\Gamma) & = & 1:\ \mathtt{CALL}\ (a_\Gamma, 0, 1);\\
 & & 2:\ \mathtt{JMP}\ 0;\\
 & & bt(\Delta\Gamma, st_{I/O}, a_\Gamma, 1).\\
 & & \\
bt(\Delta\Gamma, st_{I/O}, a_\Gamma, 1) & = & dt(\Delta, up(\Delta, st_{I/O}, a_1, 1), a_1, 1)\\
 & & ct(\Gamma, up(\Delta, st_{I/O}, a_1, 1), a_\Gamma, 1)\\
 & & a_2 : \mathtt{RET}
\end{array}
}
$$

Die Symboltabelle für die Übersetzung der Prozedur $\mathtt{F}$ und des Hauptanweisungsteils Γ ergibt sich zu:

$$up(\Delta, st_{I/O}, a_1, 1) = \underbrace{st_{I/O}[\mathtt{E} \mapsto (var, 1, 1), \mathtt{F} \mapsto (proc, a_{11}, 1, 0)]}_{\overline{st}}.$$

Beispielübersetzung (Forts.)

Damit folgt mit

$$\overline{st} = [X \mapsto (var, 0, 1), \mathtt{E} \mapsto (var, 1, 1), \mathtt{F} \mapsto (proc, a_{11}, 1, 0)] :$$

Übersetzung des Hauptprogramms:

$$\boxed{\begin{aligned} dt(\Delta, \overline{st}, a_1, 1) = bt(\Gamma_F, \overline{st}, a_{11}, 2); \ = \ & ct(\Gamma_F, \overline{st}, a_{11}, 2) \\ & a_3 : \mathtt{RET} \\ ct(\Gamma, \overline{st}, a_\Gamma, 1) = \ a_\Gamma : \ & \mathtt{LIT}\ 1; \\ & \mathtt{STORE}\ (0,1); \\ & \mathtt{CALL}\ (a_{11}, 0, 0); \\ & \mathtt{LOAD}\ (0,1); \\ & \mathtt{STORE}\ (1,1); \end{aligned}}$$

Übersetzung der Prozedur F:

$$\begin{aligned} & ct(\Gamma_F, \overline{st}, a_{11}, 2) \\ = \quad & et(\mathtt{1} < \mathtt{X}, \overline{st}, a_{11}, 2); \\ a_4 : \ & \mathtt{JPFALSE}\ a_5; \\ & ct(\textbf{begin} \ldots \textbf{end}, \\ & \qquad \overline{st}, a_4 + 1, 2) \\ a_5 : \ & \end{aligned}$$

Es gilt:

$$\begin{aligned} & et(\mathtt{1} < \mathtt{X}, \overline{st}, a_{11}, 2) \\ = \ a_{11} : \ & \mathtt{LIT}\ 1; \\ & \mathtt{LOAD}\ (2,1); \\ & \mathtt{LESS}; \end{aligned}$$

und

$$\begin{aligned} & ct(\textbf{begin} \ldots \textbf{end}, \overline{st}, a_4 + 1, 2) \\ = \ & ct(\mathtt{E} := \mathtt{E} * \mathtt{X}, \overline{st}, a_4 + 1, 2) \\ & ct(\mathtt{X} := \mathtt{X} - \mathtt{1}, \overline{st}, a_6, 2) \\ & ct(\mathtt{F}(), \overline{st}, a_7, 2) \\ = \ & a_4 + 1 : \ \mathtt{LOAD}\ (1,1); \\ & \qquad \mathtt{LOAD}\ (2,1); \\ & \qquad \mathtt{MULT}; \\ & \qquad \mathtt{STORE}\ (1,1); \\ & \qquad \mathtt{LOAD}\ (2,1); \\ & \qquad \mathtt{LIT}\ 1; \\ & \qquad \mathtt{SUB}; \\ & \qquad \mathtt{STORE}\ (2,1); \\ & \qquad \mathtt{CALL}\ (a_{11}, 1, 0) \end{aligned}$$

Ergebnis der Übersetzung:

$$
\boxed{
\begin{aligned}
&\mathit{trans}(\textbf{in/out}\ \texttt{X};\ \Delta\Gamma)\\
&\qquad = \qquad\qquad 1 : \ \texttt{CALL}\ (17,0,1);\\
&\qquad\qquad\qquad\qquad 2 : \ \texttt{JMP}\ 0;\\
&\qquad a_{11} = 3 : \ \texttt{LIT}\ 1;\\
&\qquad\qquad\qquad\qquad 4 : \ \texttt{LOAD}\ (2,1);\\
&\qquad\qquad\qquad\qquad 5 : \ \texttt{LESS};\\
&\qquad a_4 = 6 : \ \texttt{JPFALSE}\ 16;\\
&\qquad\qquad\qquad\qquad 7 : \ \texttt{LOAD}\ (1,1);\\
&\qquad\qquad\qquad\qquad 8 : \ \texttt{LOAD}\ (2,1);\\
&\qquad\qquad\qquad\qquad 9 : \ \texttt{MULT};\\
&\qquad\qquad\qquad\qquad 10 : \ \texttt{STORE}\ (1,1);\\
&\qquad\qquad\qquad\qquad 11 : \ \texttt{LOAD}\ (2,1);\\
&\qquad\qquad\qquad\qquad 12 : \ \texttt{LIT}\ 1;\\
&\qquad\qquad\qquad\qquad 13 : \ \texttt{SUB};\\
&\qquad\qquad\qquad\qquad 14 : \ \texttt{STORE}\ (2,1);\\
&\qquad\qquad\qquad\qquad 15 : \ \texttt{CALL}\ (3,1,0);\\
&a_3 = a_5 = 16 : \ \texttt{RET};\\
&\qquad a_\Gamma = 17 : \ \texttt{LIT}\ 1;\\
&\qquad\qquad\qquad\qquad 18 : \ \texttt{STORE}\ (0,1);\\
&\qquad\qquad\qquad\qquad 19 : \ \texttt{CALL}\ (3,0,0);\\
&\qquad\qquad\qquad\qquad 20 : \ \texttt{LOAD}\ (0,1);\\
&\qquad\qquad\qquad\qquad 21 : \ \texttt{STORE}\ (1,1);\\
&\qquad a_2 = 22 : \ \texttt{RET}.
\end{aligned}
}
$$

Berechnungsprotokoll bei Eingabe von 3:

$m \in BZ$	$d \in DK$	$p \in PK$
1	ε	$0:0:0:3$
17	ε	$4:3:2:0:0:0:0:3$
18	1	$4:3:2:0:0:0:0:3$
19	ε	$4:3:2:1:0:0:0:3$
3	ε	$3:2:20:4:3:2:1:0:0:0:3$
4	1	$3:2:20:4:3:2:1:0:0:0:3$
5	$1:3$	$3:2:20:4:3:2:1:0:0:0:3$
6	1	$3:2:20:4:3:2:1:0:0:0:3$
7	ε	$3:2:20:4:3:2:1:0:0:0:3$
8	1	$3:2:20:4:3:2:1:0:0:0:3$
9	$1:3$	$3:2:20:4:3:2:1:0:0:0:3$
10	3	$3:2:20:4:3:2:1:0:0:0:3$
11	ε	$3:2:20:4:3:2:3:0:0:0:3$
12	3	$3:2:20:4:3:2:3:0:0:0:3$
13	$3:1$	$3:2:20:4:3:2:3:0:0:0:3$
14	2	$3:2:20:4:3:2:3:0:0:0:3$
15	ε	$3:2:20:4:3:2:3:0:0:0:2$
3	ε	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
4	1	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
5	$1:2$	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
6	1	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
7	ε	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
8	3	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
9	$3:2$	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
10	6	$6:2:16:3:2:20:4:3:2:3:0:0:0:2$
11	ε	$6:2:16:3:2:20:4:3:2:6:0:0:0:2$
12	2	$6:2:16:3:2:20:4:3:2:6:0:0:0:2$
13	$2:1$	$6:2:16:3:2:20:4:3:2:6:0:0:0:2$
14	1	$6:2:16:3:2:20:4:3:2:6:0:0:0:2$
15	ε	$6:2:16:3:2:20:4:3:2:6:0:0:0:1$
3	ε	$9:2:16:6:2:16:3:2:20:4:3:2:6:0:0:0:1$
4	1	$9:2:16:6:2:16:3:2:20:4:3:2:6:0:0:0:1$
5	$1:1$	$9:2:16:6:2:16:3:2:20:4:3:2:6:0:0:0:1$
6	0	$9:2:16:6:2:16:3:2:20:4:3:2:6:0:0:0:1$
16	ε	$9:2:16:6:2:16:3:2:20:4:3:2:6:0:0:0:1$
16	ε	$6:2:16:3:2:20:4:3:2:6:0:0:0:1$
16	ε	$3:2:20:4:3:2:6:0:0:0:1$
20	ε	$4:3:2:6:0:0:0:1$
21	6	$4:3:2:6:0:0:0:1$
22	ε	$4:3:2:6:0:0:0:6$
0	ε	$0:0:0:6$

Korrektheit der Übersetzung

Für jedes $P \in \mathit{PSP\text{-}Prog}$ mit n Ein-/Ausgabevariablen und $(z_1, \ldots, z_n), (z'_1, \ldots, z'_n) \in \mathbb{Z}^n$ gilt:

$$\boxed{\begin{array}{c} \mathcal{M}[\![P]\!](z_1, \ldots, z_n) = (z'_1, \ldots, z'_n) \\ \\ \Longleftrightarrow \\ \\ \mathcal{I}[\![\mathit{trans}(P)]\!](1, \varepsilon, 0 : 0 : 0 : z_1, \ldots, z_n) = (0, \varepsilon, 0 : 0 : 0 : z'_1, \ldots, z'_n) \end{array}}$$

Beweis:

M. Mohnen: A Compiler Correctness Proof for the Static Link Technique by means of Evolving Algebras,
Fundamenta Informaticae 29 (1997) pp. 257–303.