Syntax von PSP — Programmiersprache mit Prozeduren

Ganze Zahlen	*Int*	:	Z		
Bezeichner	*Ide*	:	I		
Deklarationen	*Decl*	:	Δ	$::=$	$\Delta_C \Delta_V \Delta_P$
			Δ_C	$::=$	$\varepsilon \mid \textbf{const}\ I_1 = Z_1; \ldots; I_n = Z_n \quad (n \geq 1)$
			Δ_V	$::=$	$\varepsilon \mid \textbf{var}\ I_1, \ldots, I_n; \quad (n \geq 1)$
			Δ_P	$::=$	$\varepsilon \mid \textbf{proc}\ I_1; B_1; \ldots \textbf{proc}\ I_n; B_n; \quad (n \geq 1)$
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

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

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

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

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

$$\mathcal{D}[\![\textbf{var } I_1, I_2, \ldots, I_n]\!]\rho\sigma := \rho[I_1 \mapsto \alpha_{j+1}, \ldots, I_n \mapsto \alpha_{j+n}]\; \sigma[\alpha_{j+1} \mapsto 0, \ldots, \alpha_{j+n} \mapsto 0]$$

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

$$\mathcal{D}[\![\textbf{proc } I_1; B_1; \ldots \textbf{proc } I_n; B_n;]\!]\rho\sigma := \rho[I_1 \mapsto \theta_1, \ldots, I_n \mapsto \theta_n]\sigma$$

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

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

$$\begin{array}{lll}
\mathcal{C}[\![I := E]\!]\rho\sigma & := & \sigma\,[\alpha \mapsto \mathcal{E}[\![E]\!]\rho\sigma] \\
 & & \text{falls } \rho(I) = \alpha \in Loc \\
\mathcal{C}[\![I()]\!]\rho\sigma & := & \theta(\sigma) \text{ falls } \rho(I) = \theta \in C \\
\mathcal{C}[\![\Gamma_1; \Gamma_2]\!]\rho\sigma & := & \mathcal{C}[\![\Gamma_2]\!]\rho(\mathcal{C}[\![\Gamma_1]\!]\rho\sigma) \\
\multicolumn{3}{l}{\mathcal{C}[\![\textbf{if } BE \textbf{ then } \Gamma_1 \textbf{ else } \Gamma_2]\!]\rho\sigma} \\
 & := & \begin{cases} \mathcal{C}[\![\Gamma_1]\!]\rho\sigma & \text{falls } \mathcal{B}[\![BE]\!]\rho\sigma = \text{ true} \\ \mathcal{C}[\![\Gamma_2]\!]\rho\sigma & \text{falls } \mathcal{B}[\![BE]\!]\rho\sigma = \text{ false} \end{cases} \\
\mathcal{C}[\![\textbf{while } BE \textbf{ do } \Gamma]\!]\rho\sigma & := & \begin{cases} \mathcal{C}[\![\textbf{while } BE \textbf{ do } \Gamma]\!]\rho(\mathcal{C}[\![\Gamma]\!]\rho\sigma) & \\ \qquad\qquad \text{falls } \mathcal{B}[\![BE]\!]\rho\sigma = \text{ true} \\ \sigma \qquad\quad \text{falls } \mathcal{B}[\![BE]\!]\rho\sigma = \text{ false} \end{cases}
\end{array}$$

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

$$\mathcal{M}[\![\textbf{in/out } I_1, \ldots, I_n; B]\!](z_1, \ldots, z_n) := (\sigma(\alpha_1), \ldots, \sigma(\alpha_n)) \text{ mit}$$

$$\sigma := \mathcal{BL}[\![B]\!]\,\underbrace{\rho_\emptyset[I_1 \mapsto \alpha_1, \ldots, I_n \mapsto \alpha_n]}_{\text{Anfangsumgebung}}\,\underbrace{\sigma_\emptyset[\alpha_1 \mapsto z_1, \ldots, \alpha_n \mapsto z_n]}_{\text{Anfangszustand}}$$