

Syntaxe abstraite de Pseudo-Pascal

François Pottier

Cette fiche récapitule la syntaxe abstraite du langage Pseudo-Pascal. Elle correspond à la définition contenue dans le fichier **PP.mli** du compilateur.

La méta-variable b parcourt l'ensemble des booléens, à savoir $\{true, false\}$. La méta-variable n parcourt l'ensemble des entiers munis d'un signe représentables sur 32 bits, à savoir l'intervalle $[-2^{31} \dots 2^{31} - 1]$. La méta-variable x parcourt un ensemble dénombrable d'identificateurs employés pour nommer des variables. La méta-variable f parcourt un ensemble dénombrable d'identificateurs employés pour nommer des procédures ou fonctions.

Les types sont integer, boolean, et array of τ , où τ est lui-même un type, et décrit les éléments du tableau.

τ	::=	types
		integer entiers
		boolean booléens
		array of τ tableaux

Les constantes sont booléennes ou entières. Il n'y a pas de constantes de type tableau : les tableaux sont alloués dynamiquement.

k	::=	constantes
		b constante booléenne
		n constante entière

Les opérateurs unaires et binaires, qui apparaissent dans la syntaxe des expressions, sont les suivants. Tous s'appliquent à des expressions de type integer. Tous produisent un résultat de type integer, sauf les opérateurs binaires de comparaison, qui produisent un résultat de type boolean.

uop	::=	opérateurs unaires
		- négation
bop	::=	opérateurs binaires
		+ addition
		- soustraction
		× multiplication
		/ division
		< ≤ > ≥ = ≠ comparaison

Les opérations primitives, c'est-à-dire les procédures et fonctions prédéfinies, sont les suivantes.

π	::=	opérations primitives
		write affichage d'un entier
		writeln affichage d'un entier et retour à la ligne
		readln lecture d'un entier

La cible d'un appel de procédure ou fonction est soit une opération primitive, soit une procédure ou fonction définie par l'utilisateur, laquelle est alors désignée par son nom.

φ	::=	cible d'un appel
		π opération primitive
		f procédure ou fonction définie par l'utilisateur

La syntaxe des expressions, conditions, et instructions est la suivante. Notons que la distinction entre ces trois catégories est quelque peu arbitraire : on pourrait ne distinguer que deux catégories (expressions et instructions), comme en C, ou bien une seule (expressions), comme en Objective Caml. Les distinctions que la syntaxe du langage n'impose pas seront alors (si nécessaire) effectuées lors de la vérification des types.

$e ::=$		expressions
	k	constante
	x	variable
	$uop\ e$	application d'un opérateur unaire
	$e\ bop\ e$	application d'un opérateur binaire
	$\varphi(e\ \dots\ e)$	appel de fonction
	$e[e]$	lecture dans un tableau
	new array of $\tau[e]$	allocation d'un tableau
$c ::=$		conditions
	e	expression (à valeur booléenne)
	not c	négation
	c and c	conjonction
	c or c	disjonction
$i ::=$		instructions
	$\varphi(e\ \dots\ e)$	appel de procédure
	$x := e$	affectation
	$e[e] := e$	écriture dans un tableau
	$i\ \dots\ i$	séquence
	if c then i else i	conditionnelle
	while c do i	boucle

Une définition de procédure ou fonction est composée d'un en-tête, d'une série de déclarations de variables locales, et d'un corps. L'en-tête donne le nom de la procédure ou fonction, déclare ses paramètres formels et, s'il s'agit d'une fonction, indique le type de son résultat. J'écris informellement $\tau^?$ pour indiquer un type τ optionnel.

$d ::=$		définitions de procédures/fonctions
	$f(x:\tau\ \dots\ x:\tau) : \tau^?$	en-tête
	var $x:\tau\ \dots\ x:\tau$	variables locales
	i	corps

Un programme est composé d'une série de déclarations de variables globales, d'une série de déclarations de procédures ou fonctions, et d'un corps.

$p ::=$		programme
	var $x:\tau\ \dots\ x:\tau$	variables globales
	$d\ \dots\ d$	définitions de procédures/fonctions
	i	corps