Langages et Compilation : sémantique statique

Extensions du langage `while`

On étend la syntaxe abstraite du langage en considérant maintenant :

— qu'une déclaration peut contenir des *déclarations de procédure*, soit sans paramètre (noté `proc p C`), soit avec un paramètre unique (noté `proc p (y t) C`).

— qu'une commande `C` peut être un *bloc* (une commande avec une déclaration locale, noté `B`), ou un *appel de procédure*, soit sans paramètre (noté `call p`), soit avec un paramètre unique (noté `call p E`) ;

La syntaxe abstraite complète est donc la suivante, avec p, x, y ∈ Noms :

P	: :=	B
B	: :=	D C
D	: :=	var x t | proc p C | proc p (y t) C | D ; D
C	: :=	x := E | C ; C | si E alors C sinon C | tantque E C
		| B | call p | call p E
E	: :=	n | b | x | E + E | E * E | E et E | not E

Pour prendre en compte les procédures on étend l'ensemble des types :

$$\begin{aligned} \mathit{Type\_de\_Base} &= \{\ \mathit{Entier}\ ,\ \mathit{Booleen}\ ,\ \mathit{Void}\ \}. \\ \mathit{Type} &= \mathit{Type\_de\_Base} \cup \{\{\ \mathit{Proc}\ \} \times\ \mathit{Type\_de\_Base}\ \} \end{aligned}$$

Le type d'une procédure sans paramètre sera donc noté (*Proc*, *Void*), celui d'une procédure avec paramètre de type t sera noté (*Proc*, t).

On introduit également une notion de configuration pour les blocs, et on modifie les configurations utilisées pour les déclarations :

$$\begin{aligned} \mathcal{C}_B &\subseteq (B \times\ \mathit{Env}\ ) \cup\ \mathit{Types} \\ \mathcal{C}_D &\subseteq (D \times\ \mathit{Env}\ ) \cup\ \mathit{Env} \end{aligned}$$

Enfin, on introduit un opérateur de *composition* de fonctions qui permet de prendre en compte l'imbrication possible des blocs :

$$f1[f2](x) = \begin{cases} f1(x) & \text{si x} \notin \text{Dom(f2)} \\ f2(x) & \text{sinon} \end{cases}$$

Sémantique statique pour les blocs (relation $\xrightarrow{\mathbf{b}} \subseteq \mathcal{C}_B \times \mathcal{C}_B$)

Un bloc `D ; C` est correct dans l'environnement ρ ssi les déclarations `D` sont correctes dans ρ et produisent l'environnement ρ_l et si les commandes `C` sont correctes dans l'environnement composé $\rho[\rho_l]$:

$$\frac{< D, \rho > \xrightarrow{\mathrm{d}} \rho_l \quad < C, \rho[\rho_l] > \xrightarrow{\mathrm{c}} Void}{< D\ ;\ C, \rho > \xrightarrow{\mathrm{b}} Void}$$

Nouvelles règles pour les déclarations (relation $\xrightarrow{\mathbf{d}} \subseteq \mathcal{C}_D \times \mathcal{C}_D$)

Une déclaration de variable est toujours correcte :

$$< var\ x\ t, \rho > \xrightarrow{\mathrm{d}} [x \longmapsto t]$$

Une déclaration de procédure sans paramètre `p` est correcte dans un env. ρ ssi son corps `C` est correct dans cet environnement ρ étendu avec la définition de `p` (pour prendre en compte un possible appel récursif sur `p`) :

$$\frac{< C, \rho[p \longmapsto (Proc, Void)] > \xrightarrow{\mathrm{c}} Void}{<\ proc\ p\ C\ , \rho > \xrightarrow{\mathrm{d}} [p \longmapsto (Proc, Void)]}$$

Pour une procédure `p` avec un paramètre `y` il faut en plus étendre ρ avec la définition de `y` :

$$\frac{< C, \rho[p \longmapsto (Proc, t), y \longmapsto t] > \xrightarrow{\mathrm{c}} Void}{<\ proc\ p\ (y\ t)\ C\ , \rho > \xrightarrow{\mathrm{d}} [p \longmapsto (Proc, t)]}$$

Lorsque deux déclarations `D1` et `D2` ont lieu en séquence, on "propage" l'environnement généré par `D1` pour vérifier `D2` (ainsi les variables ou procédures déclarées dans `D1` sont définies lorsque l'on vérifie `D2`) :

$$\frac{< D1, \rho > \xrightarrow{\mathrm{d}} \rho 1 \quad < D2, \rho[\rho 1] > \xrightarrow{\mathrm{d}} \rho 2 \quad Dom(\rho 1) \cap Dom(\rho 2) = \emptyset}{< D1\ ;\ D2, \rho > \xrightarrow{\mathrm{d}} \rho 1 \cup \rho 2}$$

Nouvelles règles pour les commandes (relation $\xrightarrow{\mathbf{c}} \subseteq \mathcal{C}_C \times \mathcal{C}_C$)

Une commande de type "bloc" (`B`) est correcte ssi le bloc correspondant est correct :

$$\frac{< B, \rho > \xrightarrow{\mathrm{b}} Void}{< B, \rho > \xrightarrow{\mathrm{c}} Void}$$

Un appel de procédure sans paramètre est correct dans l'env. ρ ssi cette procédure est définie dans ρ :

$$\frac{x \in Dom(\rho) \quad \rho(x) = (Proc, Void)}{< \ call\ x\ , \rho > \stackrel{\text{c}}{\longrightarrow} \ Void}$$

Un appel de procédure avec paramètre est correct dans l'env. ρ ssi cette procédure est définie dans ρ, si le paramètre effectif est correct, et si son type correspond à celui du paramètre formel :

$$\frac{x \in Dom(\rho) \quad \rho(x) = (Proc, t) \quad < e, \rho > \stackrel{\text{e}}{\longrightarrow} \ t}{< \ call\ x\ e\ , \rho > \stackrel{\text{c}}{\longrightarrow} \ Void}$$

Enfin, **la correction d'un programme est maintenant définie par la règle suivante :**

$$\frac{< B, \emptyset > \stackrel{\text{b}}{\longrightarrow} \ Void}{B \ \longrightarrow \ Void}$$