Langages et compilation : sémantique dynamique

EXERCICES

Exercice 1.

En appliquant les règles vues en cours, calculer la sémantique dynamique des programmes suivants :

Programme 1	Programme 2

`var x1 : Entier ; var x2 : Entier`
`x1 := 3 ; x2 := 1 ;`
`tantque x1 < 0`
`  x1 := x2 + 1 ;`
`x2 := 2 ;`
	`var x1 : Entier ; var x2 : Entier`
	`x1 := 3 ; x2 := 1`
	`tantque x1 > 0`
	`  x1 := x2 + 1 ;`
	`x2 := 2 ;`

Exercice 2

On considère trois extensions pour le langage `while`. Donnez dans chaque cas les règles de sémantique dynamique permettant de prendre en compte ces nouvelles constructions.

1. on étend la syntaxe des *expressions* en ajoutant un opérateur `etpuis` et un opérateur `oualors` :
   e ::= e `etpuis` e | e `oualors` e | ...
2. on étend la syntaxe des *commandes* en ajoutant une nouvelle forme d'itération :
   c ::= `repeter` c `jusqua` e | ...
3. on étend la syntaxe des *déclarations* pour permettre l'initialisation des variables :
   d ::= `var` x `:` t `:=` e | ...
   exemple de programme :
   ```
   var x1 : Entier := 3 ; var x2 : Booleen := x1 > 4 ; if (x2) then x1 := ...
   ```

Exercice 3

On considère le programme suivant. Indiquez pour chaque *occurence d'utilisation* d'un identificateur quelle est l'*occurrence de définition* qui lui est associée : dans l'hypothèse d'une **liaison dynamique**, puis dans l'hypothèse d'une **liaison statique**.

```
var x : Entier := 1 ;
proc p is x := x*2 ;
proc q is call p ;
begin
     var x : Entier := 4 ;
     proc p is x := x+1 ;
     call q ;
     y := x ;
 end ;
```

Exercice 4

On étend le langage **While** en lui ajoutant une nouvelle commande, appelée *commande gardée*, dont la syntaxe est la suivante :

$$\begin{array}{lcl} C & ::= & \texttt{case}\ F\ \texttt{esac} \\ F & ::= & [E] \to C \mid [E] \to C\ \#\ F \\ E & ::= & n \mid x \mid E_1 + E_2 \mid \texttt{true} \mid E_1 = E_2 \mid \texttt{not}\ E \mid E_1\ \texttt{and}\ E_2 \end{array}$$

Q1. Une commande gardée de la forme "`case` F `esac`" est correctement typée si et seulement si, pour chacun de ses sous-termes de la forme "$[E] \to C$" alors :

1. l'expression E est correctement typée et fournit un résultat booléen ;
2. la commande C est correctement typée et fournit un résultat `Void`.

Donnez les règles de sémantique statique indiquant qu'une commande gardée est correctement typée.

Q2. On donne les règles de sémantique dynamique d'une commande gardée, en rappelant que η désigne un environnement (fonction partielle `Nom` $\to$ `Adr`), et σ une mémoire (fonction partielle `Adr` $\to$ `Valeur`).

$$\frac{< E, \eta, \sigma > \xrightarrow{e} tt \qquad < C, \eta, \sigma > \xrightarrow{c} \sigma'}{< [E] \to C, \eta, \sigma > \xrightarrow{c} \sigma'}$$

$$\frac{< E, \eta, \sigma > \xrightarrow{e} ff}{< [E] \to C, \eta, \sigma > \xrightarrow{c} \sigma}$$

$$\frac{< E, \eta, \sigma > \xrightarrow{e} tt \qquad < C, \eta, \sigma > \xrightarrow{c} \sigma'}{< [E] \to C\ \#\ F, \eta, \sigma > \xrightarrow{c} \sigma'}$$

$$\frac{< E, \eta, \sigma > \xrightarrow{e} ff \qquad < F, \eta, \sigma > \xrightarrow{c} \sigma'}{< [E] \to C\ \#\ F, \eta, \sigma > \xrightarrow{c} \sigma'}$$

$$\frac{< F, \eta, \sigma > \xrightarrow{c} \sigma'}{< \texttt{case}\ F\ \texttt{esac}, \eta, \sigma > \xrightarrow{c} \sigma'}$$

On dispose d'un environnement initial $\eta_0 = [\texttt{x} \to @_0, \texttt{y} \to @_1]$, et une mémoire initiale $\sigma_0 = [@_0 \to 2, @_1 \to 3]$. Donnez le contenu de la mémoire obtenue après exécution de chacun des fragments de programme suivants :

1. `case [x=1] → x:=x+1 # y=3 → y:=2 esac`
2. `case [x=2] → x:=x+1 # y=3 → y:=2 esac`
3. `case [y=3] → x:=x+1 # x=3 → y:=0 esac`