Langages et Traducteurs

TD9 : Sémantique Statique (2)

Exercice 1

On étend la syntaxe du langage pour autoriser l'initialisation des variables lors de leur déclaration :

D ::= var x : t := e | ...

Exemple de programme :

```
var x1 : Entier := 3 ;
var x2 : Booleen := x1 > 4 ;
if (x2) then x1 := ...
```

Complétez les règles de sémantique statique des déclarations pour prendre en compte cette nouvelle construction.

Exercice 2.

Montrez que, d'après les règles de sémantique statique vues en cours :

1. le programme suivant est **correct** :
```
var x1 : Entier ;
var x2 : Booleen ;
proc p1 (x1 : Booleen)
     x2 := x1 ;
call p1 (x2) ;
```
2. le programme suivant est **incorrect** :
```
proc p1
     call p2 ;
proc p2
     call p1 ;
call p1 ;
```

Modifiez ces règles pour autoriser la définition de telles procédures mutuellement récursives.

Indication : il faut pour cela analyser deux fois chaque séquence de déclaration, une première fois pour construire l'environnement local qui lui est associé, et une seconde fois pour vérifier que cette séquence de déclaration est correcte lorsque l'on prend en compte cet environnement local.

Exercice 3 [Examen RICM2 LT - Janvier 2006]

On dit dans un programme qu'une variable est *non initialisée* si elle est utilisée dans une expression sans avoir elle-même été affectée au préalable, ou en ayant été affectée avec une expression qui comportait elle-même des variables non initialisées. Considérons par exemple le programme suivant :

```
x := 0 ; y := 2 + x ; z := y + t ; u := 1 ; u := w ; v := v+1 ;
```

Dans ce programe :
– `x` et `y` sont bien initialisés ;
– `z` est non initialisée (car `t` est non initalisée), `u` est non initialisée (car `w` est non initialisée), et `v` est non initialisée (car `v` est non initialisée).

Certains compilateurs effectuent une vérification afin de rejeter des programmes qui contiennent des variables non initialisées (c'est le cas par exemple du compilateur `javac`) On souhaite ici définir une telle analyse pour le langage **While** en étendant la sémantique statique de ce langage. On modifie pour cela la notion de type en considérant **dans tout l'exercice** qu'un **type est désormais un entier appartenant à** $\{0, 1\}$.

Q1. On s'intéresse tout d'abord aux *expressions* : une expression E a le type 1 si toutes ses variables sont correctement initialisées et le type 0 sinon.

1. Si, dans l'environnement courant, E_1 a pour type 1 et E_2 pour type 0, que vaut le type de $E_1 + E_2$ dans cet environnement ? Donnez les règles de typage pour l'expression $E_1 + E_2$.
2. Donner les règles de typage des autres expressions (on n'oubliera pas les constantes).

Q2. On s'intéresse maintenant aux *commandes*. On constate qu'une affectation `x:=E` peut modifier l'environnement pour la variable `x`, et laisse les autres variables inchangées (le type de `x` peut être modifié en fonction du type de `E`, le type des autres variables ne change pas). La sémantique statique d'une commande est donc ici définie par une relation $\xrightarrow{c} \subseteq \mathcal{C}_C \times \mathcal{C}_C$ avec $\mathcal{C}_C \subseteq (C \times \texttt{Env}) \cup \texttt{Env}$.

A titre d'exemple, la sémantique statique de l'instruction `skip`, qui ne modifie pas l'environnement, est donné par la règle suivante :

$$< \texttt{skip}, \rho > \xrightarrow{c} \rho$$

1. Donner la règle de sémantique statique de l'affectation (`x:=E`)
2. Donner la règle de sémantique statique de la séquence (`C1 ; C2`)

Q3. On donne la règle de sémantique statique de l'instruction conditionnelle :

$$\frac{< E, \rho > \xrightarrow{e} 1 \quad < C_1, \rho > \xrightarrow{c} \rho_1 \quad < C_2, \rho > \xrightarrow{c} \rho_2}{< \texttt{if } E \texttt{ then } C_1 \texttt{ else } C_2, \rho > \xrightarrow{c} \rho'}$$

avec

$$\rho'(x) = \begin{cases} 1 & \text{si } \rho_1(x) = 1 \text{ et } \rho_2(x) = 1 \\ 0 & \text{sinon} \end{cases}$$

Expliquez cette règle.

Q4. D'après ces règles, le programme suivant est-il "correct" du point de vue de l'initialisation des variables ?

```
x:=4; b:=true;
if (b and x=5) then skip else z:=x+3;
t:=z+1;
```

Q5. Donner la règle de sémantique statique de l'itération (`while`) en la justifiant.