Université Joseph Fourier **Polytech Grenoble**
Département RICM 2 **Année 2005-06**

Langages et Traducteurs

Correction de l'Examen du mercredi 11 janvier 2006

Exercice 1

Q1. Si E_2 n'est pas initialisée, alors $E_1 + E_2$ ne l'est pas non plus. De manière générale, une opération sur les expressions est correctement initialisée ssi toutes ses opérandes le sont (donc le type est le minimum de toutes les opérandes, si il existe une opérande à 0, le type de l'opération est 0) :

$$\frac{(e_1, \rho) \xrightarrow{e} t_1 \quad (e_2, \rho) \xrightarrow{e} t_2}{(e_1 + e_2, \rho) \xrightarrow{e} Min(t_1, t_2)}$$

(les autres règles sont similaires)

Une constante entière/booléenne est correctement initialisée, une variable est correctement initialisée ssi la fonction ρ le dit :

$$(n, \rho) \xrightarrow{e} 1 \qquad (\texttt{true}, \rho) \xrightarrow{e} 1 \qquad (\texttt{x}, \rho) \xrightarrow{e} \rho(x)$$

Q2.

1. Pas de problème pour le skip : $(\rho, \texttt{skip}) \xrightarrow{c} \rho$.
2. L'affectation $x := e$ est correcte ssi e est bien initialisée, et dans la suite on ajoute l'information "x est bien initialisée" :
$$\frac{E, \rho \xrightarrow{e} 1}{(\texttt{x :=E}, \rho) \xrightarrow{c} \rho[x \mapsto 1]}$$
3. Pour la séquence, il faut passer à S_2 l'information des nouvelles variables initialisées dans S_1, ce qui donne :
$$\frac{(S_1, \rho) \xrightarrow{c} \rho_1 \quad (S_2, \rho_1) \xrightarrow{c} \rho_2}{(S_1; S_2, \rho) \xrightarrow{c} \rho_2}$$

Q3. Si une variable est correctement initialisée à l'entrée du `if`, alors elle sera correctement initialisée à la sortie (qu'elle ait été affectée ou non). Sinon, pour qu'une variable soit correctement initialisée à la sortie, il **faut** que les deux branches du `if` l'aient correctement initialisée (on fait de la sémantique statique, on ne peut savoir qu'elle branche du `if` sera effectivement prise).

Q4. Lorsqu'on déroule les règles, on voit que le test `(b and x=5)` est correctement initialisé (car `x` a un type `1` au moment du test). Le `if .. then ... else` est correctement typé, mais comme une des deux branches n'initialise pas la variable `z`, le typage de l'affectation `t:=z+1` conduit à un programme non correctement initialisé. L'environnement final a pour valeur : $\rho_f = [x \to 1, b \to 1, z \to 0, t \to 0]$.

Q5. La boucle `while` peut ou non être exécutée. Ainsi une variable est correctement initialisée si elle l'est dans l'environnement courant, et si elle l'est après une exécution du corps de la boucle.

$$\frac{(e, \rho) \xrightarrow{e} 1 \quad (S, \rho) \xrightarrow{c} \rho_1}{(\texttt{while } e \texttt{ do } S, \rho) \xrightarrow{c} \rho'} \text{ avec } \rho'(x) = \begin{cases} 1 & \text{si } \rho(x) = 1 \text{ et } \rho_1(x) = 1 \\ 0 & \text{sinon} \end{cases}$$

Exercice 2

Q1. On distingue selon la "taille" de l'instruction F à l'intérieur du `case ... esac` :

$$\frac{(E,\rho) \xrightarrow{\text{e}} Booleen \quad (C,\rho) \xrightarrow{\text{e}} Void}{(\texttt{case}\ [E] \longrightarrow C\ \texttt{esac},\rho) \xrightarrow{\text{c}} Void}$$

$$\frac{(E,\rho) \xrightarrow{\text{e}} Booleen \quad (C,\rho) \xrightarrow{\text{e}} Void \quad (\texttt{case}\ F\ \texttt{esac},\rho) \xrightarrow{\text{c}} Void}{(\texttt{case}\ [E] \longrightarrow C \# F\ \texttt{esac},\rho) \xrightarrow{\text{c}} Void}$$

Q2. Correction rapide :

1. Le premier test s'évalue à `ff` donc la règle (4) nous dit de continuer l'exécution du corps du `case`, le deuxième test s'évalue à `tt`, donc (règle (1)) on effectue la commande qui nous donne la mémoire à l'arrivée : $\sigma' = [@_0 \to 2, @_1 \to 2]$.
2. Le premier test s'évalue à `tt`, donc on effectue la commande `x:=x+1`, et finalement on trouve $\sigma' = [@_0 \to 3, @_1 \to 3]$. On ne se préoccupe pas de la suite (règle (3)).
3. Le premier test s'évalue à `tt`, donc on effectue la commande `x:=x+1` et on ne se préoccupe pas du test derrière. On trouve $\sigma' = [@_0 \to 3, @_1 \to 3]$

Exercice 3

Q1. Pile :

Q2. Code de P_3 :

```
prologue(0)
-- if b then
LD R1, [FP+12]              -- parametre b à l'adr. FP+12
CMP R1, 1
BLE sinon
-- P3(b-1)
ADD R2, R1, -1              -- R2 contient b-1
empiler(R2)                 -- on empile le paramètre b-1
LD R3, [FP+8]               -- LS(P3) à l'adr. FP+12
empiler(R3)                 -- on empile LS(P3)
CALL P3
ADD SP, SP, 8               -- on "dépile" LS(P3) et b-1
-- x1:=x2+1
LD R4, [FP+8]     -- env(x2)=env(P2) => une indirection sur LS(P3)
LD R5 [R4-4]                -- R5 contient x2
ADD R5, R5, 1               -- R5 contient x2+1
LD R6, [R4+8]               -- env(x1)=env(P1) => deux indirections sur LS(P3)
ST R5, [R6-4]               -- variable x1 à l'adr. R6-4
BA fsi
sinon:
-- Q(x1)
LD R7, [FP+8]               -- env(x1)=env(P1) => deux indirections sur LS(P3)
LD R8, [R7+8]
LD R9, [R8-4]               -- variable x1 à l'adr. R8-4
```

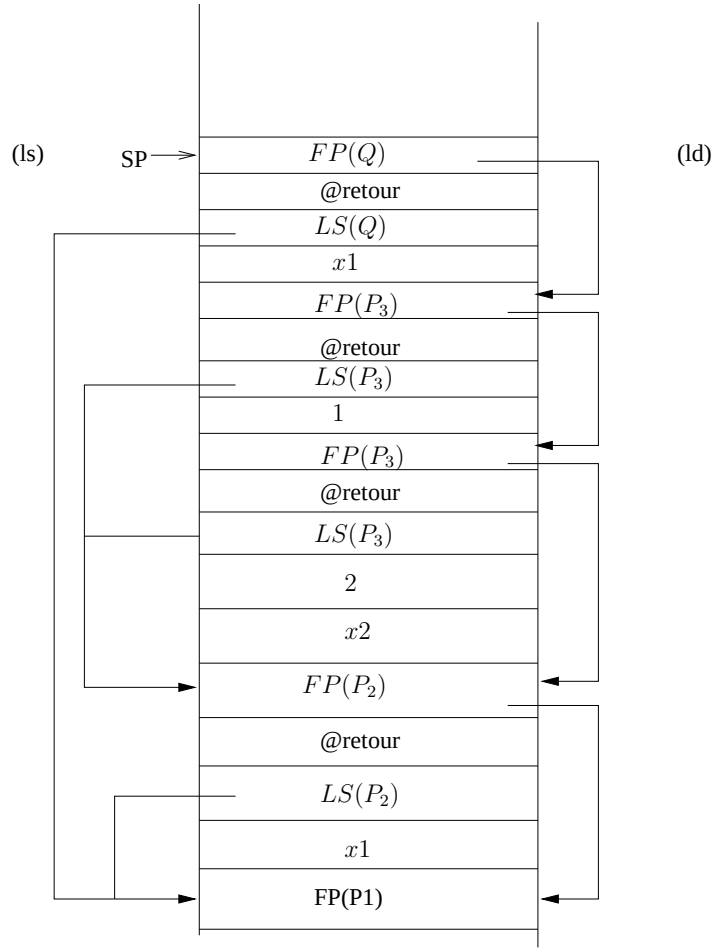

FIG. 1 – Pile à l'exécution de Q

```
empiler(R9)                -- on empile le paramètre x1
empiler(R8)                -- on empile LS(Q)
CALL Q
ADD, SP, SP, 8             -- on "dépile" LS(Q) et x1
fsi:
epilogue()
RET
```