Université Joseph Fourier **Polytech Grenoble**
Département RICM 2 **Année 2016-17**

Langages et Traducteurs

Examen du mardi 3 janvier 2017

éléments de correction

Exercice 1 ($\sim$ 13 points)

Partie 1 : sémantique statique

Q1.

1. Proposez une règle de sémantique statique pour l'expression `&x`

$$\frac{< \texttt{x}, \rho > \xrightarrow{e} (n, t)}{< \texttt{\&x}, \rho > \xrightarrow{e} (n+1, t)}$$

2. Donnez un exemple de programme incorrect vis-à-vis de cette règle.
   Pour que cette règle soit incorrecte il faut que la variable `x` soit non définie ...
   ```
   var y : entier ;
   y := &x ;
   ```
3. Proposez une règle de sémantique statique pour l'expression `*G`

$$\frac{< \texttt{G}, \rho > \xrightarrow{e} (n, t),\ n > 0}{< *\text{G}, \rho > \xrightarrow{e} (n-1, t)}$$

4. Donnez un exemple de programme incorrect vis-à-vis de cette règle.
   Pour que cette règle soit incorrecte il faut que l'opérateur `*` s'applique à une expression de type `e`ntier (et non `r`ef entier) :
   ```
   var y : entier ;
   var x : entier ;
   y := *x ;
   ```

Q2.

1. On suppose dans cette partie qu'une affectation est correcte (du point de vue des types) lorsque les opérandes gauches et droits sont corrects, et de *même type*. Proposez une règle de sémantique statique pour la commande `G := E`.

$$\frac{< \texttt{G}, \rho > \xrightarrow{d} (n, t),\ < \texttt{E}, \rho > \xrightarrow{d} (n, t)}{< \texttt{G} := \text{E}, \rho > \xrightarrow{c} Void}$$

2. Donnez un exemple de programme incorrect vis-à-vis de cette règle.
   Il suffit que les opérandes gauches et droits ne soient pas de même type, par exemple :

   ```
   var y : entier ;
   var x : entier ;
   y := &x ;
   ```

Partie 2 : sémantique dynamique

Q3.

1. Proposez une règle de sémantique dynamique pour l'expression `&x`

$$< \&\texttt{x}, \eta, \sigma > \xrightarrow{e} \eta(x)$$

2. Proposez une règle de sémantique dynamique pour l'expression `*G`

$$\frac{< G, \eta, \sigma > \xrightarrow{e} a}{< *G, \eta, \sigma > \xrightarrow{e} \sigma(a)}$$

3. Proposez une règle de sémantique dynamique pour l'affectation `G:=E`.
   Il faut distinguer deux cas, selon la nature de `G` :
   — si `G` est une variable :

$$\frac{< E, \eta, \sigma > \xrightarrow{e} v}{< x := E, \eta, \sigma > \xrightarrow{c} \sigma[\eta(x) \longmapsto v]}$$

   — si `G` est une expression de déréférencement :

$$\frac{< G, \eta, \sigma > \xrightarrow{e} a, < E, \eta, \sigma > \xrightarrow{e} v}{< *G := E, \eta, \sigma > \xrightarrow{c} \sigma[a \longmapsto v]}$$

Partie 3 : génération de code

Q4.

1. Ecrivez le code de la fonction *GenCodeAExp*(&x).

```
GenCodeAExp(&x)  =  Soit   i=AllouerRegistre()
                           k=GetSymbDepl(x)
                    dans   ((SUB Ri,FP,k),i)
```

2. Ecrivez le code de la fonction *GenCodeAExp*(∗G).

```
GenCodeAExp(*G)  =  Soit   j=AllouerRegistre()
                           (C, i) = GenCodeAExp(G)
                    dans   (C ‖ (LD Rj,[Ri]),j)
```

Partie 4 : Déréférençage implicite

Q5.

1. Ecrivez la nouvelle règle de sémantique statique pour l'expression `E1 + E2`

$$\frac{< \texttt{E1}, \rho > \xrightarrow{d} (n_1, Entier), < \texttt{E2}, \rho > \xrightarrow{d} (n_2, Entier)}{< \texttt{E1} + \texttt{E2}, \rho > \xrightarrow{d} (0, Entier)}$$

2. Ecrivez la nouvelle règle de sémantique statique pour l'expression `E1 = E2`

$$\frac{< \texttt{E1}, \rho > \xrightarrow{d} (n_1, t), < \texttt{E2}, \rho > \xrightarrow{d} (n_2, t)}{< \texttt{E1} = \texttt{E2}, \rho > \xrightarrow{d} (0, Booleen)}$$

3. Ecrivez la nouvelle règle de sémantique statique pour la commande `G := E`

$$\frac{< \texttt{G}, \rho > \xrightarrow{\text{d}} (n_g, t),\ < \texttt{E}, \rho > \xrightarrow{\text{d}} (n_e, t),\ n_g \geq n_e}{< \texttt{G} := \text{E}, \rho > \xrightarrow{\text{c}} Void}$$

4. Quelles sont les informations qu'il est nécessaire de mémoriser lors de la vérification de la sémantique statique pour la phase de génération de code ? Illustrez votre réponse (informelle) en donnant le code assembleur que produirait le compilateur pour les trois dernières instructions du programme précédent (et en précisant ce qui est fourni par la sémantique statique).

   Pour pouvoir générer un code correct compte-tenu de ces nouvelles règles il faut mémoriser lors de la vérification des types le nombre (minimal) d'opérateurs de déréférencement qu'il est nécessaire d'introduire pour que la sémantique statique soit respectée. Cette information peut être mémorisée sur l'arbre abstrait. Ainsi, l'instruction `x := x+y` est correcte que si y est déréférencé une fois. Le code produit sera donc :

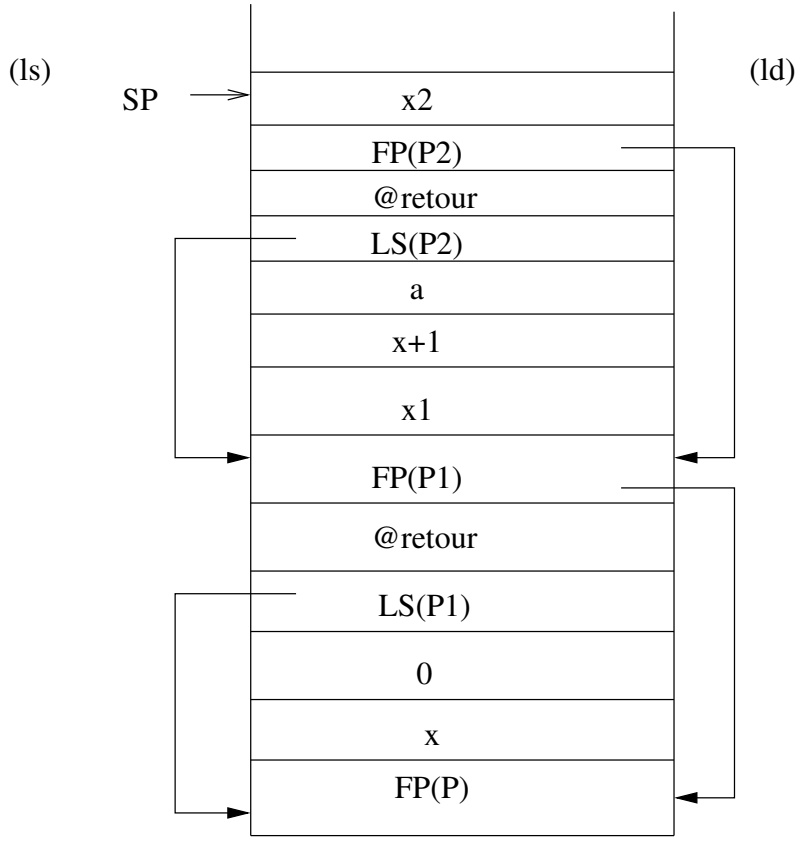
```
LD R1, [FP+dx]
LD R2, [FP+dy]
LD R3, [R2]     -- y doit être déréférencé une fois
ADD R4, R1, R3
ST R4, [FP+dx]
```

Exercice 2 (∼ 7 points)

```
procedure P(void);
  var x ;

  procedure P1 (a)
  var x1 ;

  procedure P2(b, c);
     var x2 ;
     begin { P2 }
          x2 := c ;
          x := x1+ x2 + b ;    /* instruction (3) */
     end; { P2 }

  begin { P1 }
     x1 :=  a ;
     P2 (x+1, a) ;   /* instruction (2) */
     x := 2 ;
  end; { P1 }

begin { P }
  x := 0 ;
  P1 (x);   /* instruction (1) */
end { P }
```

Q1. Dessinez le contenu de la pile lors de l'exécution de la procédure `P2`. On précisera les liens statiques et dynamiques des procédures appelées.

Q2. Donnez le code assembleur correspondant à l'instruction **(2)** de la procédure `P1`.




```
-- P1(x)
LD R1, [FP-4]      -- Ri contient x
empiler R1         -- on empile le parametre x
empiler FP         -- on empile env(P) = LS(P1)
call P1            -- appel a P1
ADD SP, SP, 8      -- on restaure SP
```

Q2. Donnez le code assembleur correspondant à l'instruction **(2)** de la procédure `P1`.

```
-- P2(x+1, a)
LD R1, [FP+8]     -- R1 contient LS(P1)=env(P)
LD R2, [R1-4]     -- R2 contient x
ADD R3, R2, 1     -- R3 contient x+1
empiler (R3)      -- on empile x+1
LD R4, [FP+12]    -- R4 contient a
empiler (R4)      -- on empile a
empiler (FP)      -- on empile env(P1) = LS(P2)
call P2           -- appel a P2
ADD SP, SP, 12    -- on restaure SP
```

Q3. Donnez le code assembleur correspondant à l'instruction **(3)** de la procédure `P2`.

```
-- x := x1 + b + x2
LD R1, [FP+8]     -- R1 contient LS(P2)=env(P1)
```

```
LD R2, [R1-4]     -- R2 contient x1
LD R3, [FP+16]    -- R3 contient b
ADD R4, R2, R1    -- R4 contient x1+b
LD R5, [FP-4]     -- R5 contient x2
ADD R6, R4, R5    -- R6 contient x1+b+x2
LD R7, [R1+8]     -- R7 contient LS(P1)=env(P)
ST R6, [R7-4]     -- x := R6
```