Université Joseph Fourier **Polytech Grenoble**
Département RICM 4 **Année 2015-16**

Langages et Traducteurs

Examen du lundi 4 janvier 2016
(éléments de correction)

Durée : 1H30 - **Tous documents autorisés**. - Les 2 exercices sont **indépendants**.

Exercice 1 - Génération de Code (∼ 8 points)

```
procedure P() is
     var x ; var y ;

     procedure P1 (a, b) is
           var x1 ; var y1 ;
           x1 := b ;
           y1 := x + a ;              /* instruction (1) */
     end ;

     procedure P2(b, c) is
           var x2 ;

           procedure P3 (c, d) is
             var x3 ;
             y := b + x2 + x3 ;       /* instruction (2) */
             call P1(c, x)            /* instruction (3) */
           end;

           call P3 (x2, 2) ;
     end;

     x := 0 ; y := 1 ; call P2 (12, 42) ;
end
```

Q1. Dessinez le contenu de la pile lors de l'exécution de la procédure `P1`. On précisera les liens statiques et dynamiques des procédures appelées.

(voir feuille jointe)

Q2. Donnez le **code assembleur commenté** correspondant à l'instruction **(1)** de la procédure `P1`.

```
LD R1, [FP+8]  ! R1 := LS(P1)=env(P)
LD R2, [R1-4]  ! R2 := x
LD R3, [FP+16]  ! R3 := a
ADD R4, R2, R3  ! R4 := x+a
ST R4, [FP-8] ! y1 := R4
```

Q3. Donnez le **code assembleur commenté** correspondant à l'instruction **(2)** de la procédure `P3`.

```
LD R1, [FP+8]  ! R1 := LS(P3)=env(P2)
LD R2, [R1+16]  ! R2 := b
LD R3, [R1-4]  ! R3 := x2
ADD R4, R2, R3  ! R4 := b+x2
LD R5, [FP-4] ! R5 := x3
ADD R6, R4, R5  ! R6 := b+x2+x3
LD R7, [R1+8]  ! R7 := LS(P2)=env(P)
ST R6, [R7-8] ! y1 := R4
```

Q4. Donnez le **code assembleur commenté** correspondant à l'instruction **(3)** de la procédure `P3`.

```
LD R1, [FP+16] ! R1 := c
empiler(R1)
LD R2, [FP+8]  ! R2 := LS(P3)=env(P2)
LD R3, [R2+8]  ! R3 := LS(P2)=env(P)
LD R4, [R3-4] ! R4 := x
empiler(R4)
empiler (R3)  ! on empile LS(P1)=env(P)
call P1
ADD SP, SP, 12 ! on restaure la pile
```

Exercice 2 - Sémantique ($\sim$ 12 points)

On s'intéresse à une **extension** du langage étudié en cours dans laquelle les variables déclarées peuvent avoir deux **statuts** : soit ***secrètes*** (`sec`) soit ***publiques*** (`pub`). Dans ce langage, un programme sera **correct** si et seulement si :

(C1) il respecte les règles de sémantique statique et dynamique vues en cours ;

(C2) il ne contient pas d'**affectation interdite**, c'est-à-dire une affectation **à une variable publique d'une valeur calculée à partir de variables secrètes**.

On s'intéresse uniquement à la condition **(C2)** dans cet exercice, la condition **(C1)** étant déjà formalisée par les règles de sémantique statique et dynamique vues en cours.

On donne d'abord la syntaxe abstraite de ce nouveau langage (seules les déclarations changent par rapport au langage vu en cours) :

P	: :=	D C
D	: :=	var x t : s | D ; D
C	: :=	x := E | C ; C | si E alors C sinon C fin | tantque E C fin
E	: :=	n | b | x | E op E | E = E | not E

Dans cette syntaxe :
- D, C et E désignent respectivement une *déclaration*, une *commande* et une *expression*.
- x désigne un identificateur, dont le nom appartient à un ensemble Noms.
- *op* désigne un opérateur arithmétique ($+$, $-$, $\times$, etc.)
- t désigne un type, élément de l'ensemble Type = {`int`, `bool` }.
- s désigne un **statut**, élément de l'ensemble Statut = {`sec`, `pub`}.
  On peut supposer dans la suite que `sec` > `pub` (cela peut simplifier l'écriture des règles).
- n désigne une constante entière, et b une constante booléenne (`true` ou `false`).

On donne ci-dessous deux exemple de programmes respectant cette syntaxe :

```
(* P1 *)
var x int : sec ;
var y int : sec ;
var z int : pub ;
x := 0 ; y := x + 1 ;
z := 3 ;
si not (z = 0) alors
   z := z+1 ;
sinon
   y := z * y ;
fin
```

```
(* P2 *)
var x int : sec ;
var y int : sec ;
var z int : pub ;
x := 0 ; y := x + 1 ;
z := 3 ;
si not (z = 0) alors
   z := 2 * y ; (* Incorrect *)
sinon
   y := z * x ;
fin
```

Le programme `P2` est incorrect car il contient une affectation de `2 * y` (qui est une valeur secrète) à la variable `z` qui est publique. Le programme `P1` est correct.

Sémantique statique

On étend les notations du cours en ajoutant à la fonction Environnement ρ une nouvelle fonction α :
– ρ : Noms $\longrightarrow$ Type, $\rho(x)$ désigne le type (`in` ou `bool`) de la variable `x`.
– α : Noms $\longrightarrow$ Statut, $\alpha(x)$ désigne le statut (`sec` ou `pub`) de la variable `x`.
Dans la suite de cette partie partie on cherche à écrire de **nouvelles** règles de sémantique permettant de vérifier **uniquement** la condition **(C2)**. On ne s'intéresse donc pas au **type** des variables, mais uniquement à leur **statut**.

Déclarations

On définit la relation $\xrightarrow{\text{d}}$ de la manière suivante : $D \xrightarrow{\text{d}} \alpha$ signifie que la fonction α donne le statut de chaque variable déclarée dans D.

Q1. Définissez cette relation $\xrightarrow{\text{d}}$ en complétant les deux règles ci-dessous :

$$var\ x\ t : s \xrightarrow{\text{d}} [x \longmapsto s]$$

$$\frac{D1 \xrightarrow{\text{d}} \alpha 1 \quad D2 \xrightarrow{\text{d}} \alpha 2}{D1\ ;\ D2 \xrightarrow{\text{d}} \alpha 1 \cup \alpha 2}$$

Expression

A chaque expression est associée son statut (`pub` ou `sec`). Informellement :
– une constante est toujours publique ;
– une expression non constante est secrète si sa valeur dépend d'au moins une variable secrète.
La relation $\xrightarrow{\text{e}}$ associe donc son **statut** à toute expression E :
$< E, \alpha > \xrightarrow{\text{e}} s$ signifie que, étant donnée la fonction α indiquant le statut de chacune de ses variables, l'expression E a pour statut s.

Q2. Définissez cette relation $\xrightarrow{\text{e}}$ en complétant les 5 règles ci-dessous :

$$< n, \alpha > \xrightarrow{e} pub \qquad < b, \alpha > \xrightarrow{e} pub \qquad < x, \alpha > \xrightarrow{e} \alpha(x)$$

$$\frac{< E1, \alpha > \xrightarrow{e} s1 \quad < E2, \alpha > \xrightarrow{e} s2}{< E1 \ op \ E2, \alpha > \xrightarrow{e} max(s1, s2)} \qquad \frac{< E1, \alpha > \xrightarrow{e} s1 \quad < E2, \alpha > \xrightarrow{e} s2}{< E1 = E2, \alpha > \xrightarrow{e} max(s1, s2)}$$

Commandes

Il reste maintenant à indiquer si une commande est correcte ou non vis-à-vis des affectations qu'elle contient. On considère donc la relation $\xrightarrow{c}$ qui associe à chaque commande C la "valeur" `OK` si et seulement si cette commande est correcte : $< C, \alpha > \xrightarrow{c} OK$ signifie que, étant donnée la fonction α indiquant le statut de chacune de ses variable, la commande C vérifie la condition **C2** (elle ne contient pas d'affectation interdite).

Q3. Définissez cette relation $\xrightarrow{c}$ en complètant tout d'abord les deux règles suivantes :

$$\frac{\alpha(x) = s1 \quad < E, \alpha > \xrightarrow{e} s2 \quad s1 \geq s2}{< x := E, \alpha > \xrightarrow{c} OK} \qquad \frac{< C1, \alpha > \xrightarrow{c} OK \quad < C2, \alpha > \xrightarrow{c} OK}{< C1; C2, \alpha > \xrightarrow{c} OK}$$

Q4. Dans le cas d'une commande conditionelle (`si alors sinon`) ou itérative (`tantque`), il faut vérifier que **toute exécution possible** [1] ne contient aucune affectation d'une valeur secrète à une variable publique. Ecrivez les règles correspondantes.

$$\frac{< C1, \alpha > \xrightarrow{c} OK \quad < C2, \alpha > \xrightarrow{c} OK}{< \ si \ E \ alors \ C1 \ sinon \ C2, \alpha > \xrightarrow{c} OK} \qquad \frac{< C, \alpha > \xrightarrow{c} OK}{< \ tantque \ E \ C, \alpha > \xrightarrow{c} OK}$$

Le programme complet

Enfin, un programme complet D ; C vérifie **(C2)** ssi la commande C est correcte vis-à-vis du statut des variables déclarées dans D.

Q5. Formalisez cette définition en complétant la règle suivante :

$$\frac{D \xrightarrow{d} \alpha \quad < C, \alpha > \xrightarrow{c} OK}{D \ C \longrightarrow OK}$$

Sémantique Dynamique

On étend maintenant la sémantique dynamique vue en cours pour vérifier **à l'exécution** la condition **(C2)**. Pour cela on reprend les notations vues en cours :

1. quelle que soit la valeur de l'expression booléene

– Entiers relatifs : $\mathbb{Z}$; Booléens : $\mathbb{B} = \{tt, ff\}$; Valeurs : Val $= \mathbb{Z} \cup \mathbb{B}$; Adresses : Adr $= \mathbb{N}$.
– environnement : Env = Noms $\rightarrow$ Adr, fonction partielle qui associe une adresse à certains noms ; η désignera un environnement.
– mémoire : Mem = Adr $\rightarrow$ Val, fonction partielle qui associe une valeur à certaines adresses ; σ désignera une mémoire.

La sémantique dynamique des déclarations (D) est inchangée par rapport à celle vue en cours.

Expression

La sémantique des expressions est formalisée par une nouvelle relation $\xrightarrow{e}$ qui associe maintenant à toute expression sa valeur **et son statut** :
$< \texttt{E}, \alpha, \eta, \sigma > \xrightarrow{e} < v, s >$ signifie que l'évaluation de l'expression E, dans l'environnement η, dans la mémoire σ, et en considérant la fonction α qui indique le statut des variables présentes dans E, produit la valeur v de statut s.

On donne à titre d'exemple la règle définissant cette relation pour une constante entière n :

$$< n, \alpha, \eta, \sigma > \xrightarrow{e} < n, pub >$$

Q6. Complétez la règle définissant cette relation pour une variable x :

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

Q7. Complétez les deux règles suivantes :

$$\frac{< e1, \alpha, \eta, \sigma > \xrightarrow{e} < v1, s1 > \quad < e2, \alpha, \eta, \sigma > \xrightarrow{e} < v2, s2 >}{< e1 \ op \ e2, \alpha, \eta, \sigma > \xrightarrow{e} < v1 \ op \ v2, max(s1, s2) >}$$

$$\frac{< e1, \alpha, \eta, \sigma > \xrightarrow{e} \ v1 \quad < e2, \alpha, \eta, \sigma > \xrightarrow{e} \ v2}{< e1 = e2, \alpha, \eta, \sigma > \xrightarrow{e} < v1 = v2, max(s1, s2) >}$$

Commandes

La sémantique des commandes est formalisée par une relation $\xrightarrow{c}$ qui associe à toute commande une nouvelle mémoire **ssi cette commande vérifie (C2)** (sinon la relation $\xrightarrow{c}$ n'est pas définie).

Ainsi, $< \texttt{C}, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma'$ signifie que l'exécution de C, dans l'environnement η et dans la mémoire σ et en considérant la fonction α qui indique le statut des variables présentes dans C, **ne contient pas d'affectations interdites** et produit la nouvelle mémoire σ'.

Q8. Complétez les deux règles suivantes :

$$\frac{< e, \alpha, \eta, \sigma > \xrightarrow{e} < v, s > \quad \alpha(x) \geq s}{< x := e, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma[\eta(x) \longmapsto v]} \qquad \frac{< c1, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma' \quad < c2, \alpha, \eta, \sigma' > \xrightarrow{c} \ \sigma''}{< c1; c2, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma''}$$

Q9. Donnez les règles de sémantique statique des commandes conditionnelle et itérative.

$$\frac{< e, \alpha, \eta, \sigma > \xrightarrow{e} < \ tt\ , s > \quad < c1, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma'}{< \ si\ e\ alors\ c1\ sinon\ c2, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma'}$$

$$\frac{< e, \alpha, \eta, \sigma > \xrightarrow{e} < \ ff\ , s > \quad < c2, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma'}{< \ si\ e\ alors\ c1\ sinon\ c2, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma'}$$

$$\frac{< e, \alpha, \eta, \sigma > \xrightarrow{e} \ tt\ , \quad < c, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma', \quad < \ tantque\ e\ c, \alpha, \eta, \sigma' > \xrightarrow{c} \ \sigma''}{< \ tantque\ e\ c, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma''}$$

$$\frac{< e, \alpha, \eta, \sigma > \xrightarrow{e} \ ff}{< \ tantque\ e\ c, \alpha, \eta, \sigma > \xrightarrow{c} \ \sigma}$$

Q10. On considère le programme suivant :

```
var x int : pub ;
var y int : sec ;
x := 4 ;
y := 4 * (x - x) ;
```

Ce programme est-il correct d'après votre sémantique statique ? Et d'après votre sémantique dynamique ? Justifiez vos réponses en dessinant les arbres de preuve obtenus (ou non !) avec vos règles de sémantique (et en complétant ces règles si besoin ...).

– Ce programme est correct d'après la semantique statique :

Pour la partie déclaration :

$$\frac{var\ x\ int : pub \ \xrightarrow{d}\ [\ x\ \rightarrow\ pub\ ] \qquad var\ y\ int : sec\ \rightarrow [\ y\ \rightarrow\ pub\ ]}{var\ x\ int : pub\ ;\ var\ y\ int : sec\ \xrightarrow{d}\ \alpha = [\ x\ \rightarrow\ sec\ ,\ y\ \rightarrow\ pub\ ]}$$

Pour la partie commande :

$$\frac{< \ x\ , \alpha > \xrightarrow{e} \ \alpha(x) = \ pub \qquad < 4, \alpha > \xrightarrow{e} \ pub \qquad pub\ \leq\ pub}{< \ x := 4\ , \alpha > \xrightarrow{c} \ OK}$$

et

$$\dfrac{\dfrac{\dfrac{}{< \ x\ , \alpha > \xrightarrow{e} \ \alpha(x) = \ pub} \qquad \dfrac{}{<4,\alpha> \xrightarrow{e}_{pub} < (x\ -\ x)\ ,\alpha> \xrightarrow{e}_{sec}}}{< y\ ,\alpha> \xrightarrow{e}_{\alpha(y)= sec} \ < 4\ *\ (x\ -\ x)\ ,\alpha> \xrightarrow{e}_{max(\{pub,\ sec\})= sec} \qquad pub \geq seq}}{< y\ :=\ 4\ *\ (x\ -\ x)\ ,\alpha> \xrightarrow{c} OK}$$

On en déduit donc que ce programme est correct.

– Ce programme est également correct d'après la semantique dynamique ...

Q11. Donnez un exemple de programme qui est rejeté par la sémantique statique mais qui serait accepté par la sémantique dynamique. Justifiez votre réponse en quelques lignes.

```
var x int : sec ;
var y int : sec ;
var z int : pub ;
x := 0 ; y := x + 1 ;
z := 3 ;
si faux alors
   z := 2 * y ; (* Incorrect, mais jamais execute ! *)
sinon
   y := z * x ;
fin
```

L'instruction `z := 2 * y` affecte une valeur secrète (`2 * y`) à la variable publique `z`. Ce programme est donc rejeté par la sémantique statique. Par contre, comme cette instruction n'est jamais exécutée ce programme est accepté par la sémantique dynamique (toutes les instructions **exécutées** vérifient la condition **C2**).