Examen Analyse Syntaxique et Compilation

Université Paris 7. Licence d'Informatique (L3). Deuxième session 2006-2007. Durée 2h. L'utilisation de documents ou de dispositifs électroniques est interdite.

18 juin 2007

Exercice 1 (4 points) *On considère la grammaire suivante avec symbole initial S :*

$$\begin{array}{ll} S \to G\$ & G \to P \\ G \to PG & P \to id : R \\ R \to \epsilon & R \to idR \end{array}$$

1. *Calculez les fonctions null, First et Follow sur les symboles non-terminaux de la grammaire.*
   Sol. Le seul symbole nullable est R
   $First(S) = First(G) = First(P) = First(R) = \{id\}$
   $Follow(S) = \emptyset$, $Follow(G) = \{\$\}$, $Follow(P) = Follow(R) = \{id, \$\}$
2. *La grammaire est-elle LL(1) ? Motivez votre réponse.*
   Sol. Non, car, par exemple, $G \to P$, $G \to PG$ et $id \in First(P) \cap First(PG)$.

Exercice 2 (4 points) *On considère la grammaire suivante avec symbole initial S :*

$$S \to A\$ \qquad A \to BA \mid \epsilon \qquad B \to aB \mid b$$

Expliquez pourquoi cette grammaire n'est pas LR(0).
Suggestion *$A \to \cdot$ est un item complet.*

Sol. On construit (une petite partie de) l'AFN qui reconnaît les préfixes admissibles. Par exemple, on remarque qu'a partir de l'état initial par des transitions ϵ l' automate arrive à l'item complet $A \to \cdot$ ainsi qu'à d'autres items. Donc la grammaire n'est pas LR(0).

Exercice 3 (4 points) *On se focalise sur le langage impératif étudié dans le cours. On souhaite analyser les programmes suivants qui se composent d'une déclaration de procédure f, d'une déclaration de variable x et d'un corps principal $f()$.*

Programme 1

```
procedure f() =
  let x = ref 1 : ref int;
  let y = ref !x : ref int;
  x :=!y;
let x = ref true : ref bool;
f()
```

Programme 2

```
procedure f() =
  let x = ref true : ref bool;
  let y = ref !x : ref int;
  x :=!y;
let x = ref 1 : ref int;
f()
```

Programme 3

```
procedure f() =
  let y = ref !x : ref int;
  x :=!y;
let x = ref 1 : ref int;
f()
```

Pour chaque programme, précisez si :

1. *Le programme est bien typé.*
2. *L'exécution du programme est susceptible de produire une erreur.*

Suggestion *Vous pouvez répondre à ces questions sans calculer formellement le typage et l'évaluation des programmes.*

Sol. D'après les règles de portée lexicale du langage en considération, la déclaration de la variable x qui suit la déclaration de la procédure f n'est pas visible dans le corps de la procédure ni au moment du typage ni au moment de l'exécution.
Le programme 1 est bien typé et donc l'exécution ne produit pas d'erreur.
Le programme 2 n'est pas bien typé car on mémorise une valeur true dans une location qui doit contenir des entiers. Cependant, l'évaluation du programme pourrait se passer normalement si le fait de mémoriser un booléen à la place d'un entier ne soulève pas d'exception.
Le programme 3 n'est pas bien typé car dans la déclaration de f on ne trouve pas la variable x dans l'environnement de type. Par ailleurs l'évaluation du corps de la fonction échoue car aucune location est associée à la variable x au moment de l'évaluation.

Exercice 4 (4 points) *On considère la commande S suivante dans le langage impératif étudié dans le cours :*

```
while !x > 0 do
  x :=!x−!y;
  y :=!y+!y
```

Proposez une compilation de la commande S dans le code octet étudié dans le cours. La compilation de S est relative à une liste de variables $x \cdot y$ et à une continuation κ.

Rappel *Le code compilé peut contenir les instructions suivantes :*

- build c n *: on remplace n valeurs $v_1 \cdots v_n$ au sommet de la pile par $c(v_1, \ldots, v_n)$ et on incrémente le compteur ordinal. Ici c est un constructeur. Par exemple,* (build 34 0) *écrit* 34 *au sommet de la pile.*
- load n *: on copie l'n-ième valeur de la pile au sommet de la pile et on incrémente le compteur ordinal.*

- goto j : *on affecte j au compteur ordinal.*
- branch j : *si la valeur au sommet de la pile est true on incrémente le compteur ordinal sinon on affecte le compteur ordinal à j. Dans les deux cas on supprime la valeur au sommet de la pile.*
- op n : *on remplace les n valeurs $v_1 \cdots v_n$ au sommet de la pile par $\underline{op}(v_1, \ldots, v_n)$ et on incrémente le compteur ordinal. Ici on suppose disposer des opérations binaires, geq, add, sub pour comparer, additionner et soustraire.*
- read *si la valeur au sommet de la pile est une location ℓ, on remplace ℓ par son contenu et on incrémente le compteur ordinal.*
- write *si au sommet de la pile on a une location ℓ située au dessus d'une valeur v alors on écrit v dans la location ℓ et on élimine v et ℓ.*

Sol. On utilise un ';' pour séparer les instructions.
$(\nu\kappa')\kappa'$: load 1; read; build 0 0; geq 2;
branch κ; load 1; read; load 2; read;
sub 2; load 1; write; load 2; read;
load 2; read; add 2; load 2; write; goto κ'

Exercice 5 (4 points) *Les règles de typage étudiées pour le λ-calcul sont les suivantes :*

$$\frac{E(x) = \tau}{E \vdash x : \tau} \qquad \frac{E[\tau/x] \vdash e : \tau'}{E \vdash \lambda x.e : \tau \to \tau'} \qquad \frac{E \vdash e : \tau \to \tau' \quad E \vdash e' : \tau}{E \vdash ee' : \tau'}$$

et les règles d'évaluation en appel par valeur sont les suivantes :

$$\frac{}{v \Downarrow v} \qquad \frac{e \Downarrow \lambda x.e_1, \quad e' \Downarrow v' \quad [v'/x]e_1 \Downarrow v}{ee' \Downarrow v}$$

Rappel *Une valeur v est un λ-terme de la forme $\lambda x.e$.*

1. *Trouvez, s'il existe, un type τ tel que $\emptyset \vdash \lambda x.\lambda y.\lambda z.xz(yz) : \tau$ est dérivable.*
   Sol. $(\tau_1 \to (\tau_2 \to \tau_3)) \to ((\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3))$
2. *Trouvez, s'il existe, un λ-terme e tel que $\emptyset \vdash e : \tau \to ((\tau \to \tau') \to \tau')$ est dérivable où τ, τ' sont deux types différents.*
   Sol. $\lambda x.\lambda f.fx$
3. *Soit $v = \lambda f.\lambda x.f(fx)$. Évaluez en appel par valeur le λ-terme $(vv)v$.*
   Suggestion *Évaluez d'abord vv.*
   Sol. On dérive $vv \Downarrow v_1$ avec $v_1 = \lambda x.v(vx)$.
   Ensuite on dérive $(vv)v \Downarrow \lambda x.v_1(v_1 x)$.