# Examen Analyse Syntaxique et Compilation

Université Paris 7. Licence d'Informatique (L3). Première session 2006-2007. Durée 2h.

L'utilisation de documents ou de dispositifs électroniques est interdite.

4 mai 2007

**Exercice 1 (5 points)** *On considère la grammaire suivante avec $S$ symbole initial :*

$$\begin{array}{ll} S \to E\$ & E \to id \\ E \to id(E) & E \to E + id \end{array}$$

1. *Donnez la représentation graphique d'un automate fini non-déterministe qui reconnaît les préfixes admissibles de la grammaire.*

   Sol. On abrège les états comme suit : $0 = q_0, 1 = S \to .E\$, 2 = S \to E.\$, 3 = S \to E\$., 4 = E \to .id, 5 = E \to id., 6 = E \to .E + id, 7 = E \to .id(E), 8 = E \to id.(E), 9 = E \to id(.E), 10 = E \to id(E.), 11 = E \to id(E)., 12 = E \to E. + id, 13 = E \to E + .id, 14 = E \to E + id.$ .

   Les transitions sont : $0 \xrightarrow{\epsilon} 1; 1 \xrightarrow{E} 2; 2 \xrightarrow{\$} 3; 1 \xrightarrow{\epsilon} 4; 1 \xrightarrow{\epsilon} 6; 1 \xrightarrow{\epsilon} 7; 4 \xrightarrow{id} 5; 6 \xrightarrow{E} 12; 6 \xrightarrow{\epsilon} 4, 6, 7; 12 \xrightarrow{+} 13; 13 \xrightarrow{id} 14; 7 \xrightarrow{id} 8; 8 \xrightarrow{(} 9; 9 \xrightarrow{E} 10; 9 \xrightarrow{\epsilon} 4, 6, 7; 10 \xrightarrow{)} 11.$

2. *La grammaire est-elle LR(0) ? Justifiez votre réponse.*

   Sol. Non. Par exemple de 0 avec *id* on peut arriver à 5 et 8 et 5 est un item complet.

**Exercice 2 (5 points)** *On se place dans le cadre du langage impératif étudié dans le cours. On modifie la catégorie syntaxique des commandes $S$ en remplaçant la commande while par la commande goto :*

$$S ::= id := e \mid lab : S \mid goto\ lab \mid (if\ e\ then\ S) \mid S; S$$

*Ici lab est une nouvelle catégorie syntaxique d'étiquettes qu'on dénote par $a, b, c, \ldots$*

*Pour décrire l'évaluation de ces nouvelles commandes on élargit le domaine de définition des environnements aux étiquettes. Si $a$ est une étiquette, $\eta$ est un environnement et $\eta(a)$ est défini alors $\eta(a)$ est une commande. Les règles d'évaluation pour les deux nouvelles commandes introduites sont alors les suivantes :*

$$\frac{(S, \eta[S/a], \mu) \Downarrow \mu'}{(a : S, \eta, \mu) \Downarrow \mu'} \qquad \frac{(\eta(a), \eta, \mu) \Downarrow \mu'}{(goto\ a, \eta, \mu) \Downarrow \mu'}$$

1. *Complétez la description de l'évaluation en donnant les règles pour l'affectation $x := e$, le branchement $(if\ e\ then\ S)$ et la séquentialisation $S_1; S_2$.*

   Sol. Les règles données dans le cours.

2. *Proposez un schéma de traduction de la commande* $(\mathit{while}\ e\ \mathit{do}\ S)$ *dans le langage avec goto.*

   SOL. $\langle \mathit{while}\ e\ \mathit{do}\ S\rangle = a : \mathit{if}\ e\ \mathit{then}\ (\langle S\rangle; \mathit{goto}\ a)$.

   où $a$ est une nouvelle étiquette.

3. *Appliquez votre schéma de traduction à la commande :*

$$\begin{array}{l}(\mathit{while}\ x\ \mathit{do}\\ \quad (\mathit{while}\ y\ \mathit{do}\ S_1))\ ;\\ (\mathit{while}\ z\ \mathit{do}\ S_2)\end{array}$$

   SOL.

   $a : (\mathit{if}\ x\ \mathit{then}\ b : (\mathit{if}\ y\ \mathit{then}\ \langle S_1\rangle; \mathit{goto}\ b); \mathit{goto}\ a); c : (\mathit{if}\ z\ \mathit{then}\ (\langle S_2\rangle; \mathit{goto}\ c))$.

**Exercice 3 (3 points)** *On rappelle un fragment de la compilation des commandes du langage impératif :*

$$\begin{array}{ll}\mathcal{C}(x := e, w, \kappa) & = \mathcal{C}(e, w) \cdot (\mathsf{load}\ i(x, w)) \cdot (\mathsf{write}) \cdot (\mathsf{goto}\ \kappa)\\ \mathcal{C}(S_1; S_2, w, \kappa) & = \nu\kappa'\ \mathcal{C}(S_1, w, \kappa')\ \kappa' : \mathcal{C}(S_2, w, \kappa)\\ \mathcal{C}(\mathit{while}\ e\ \mathit{do}\ S, w, \kappa) & = \nu\kappa'\ (\kappa' : \mathcal{C}(e, w) \cdot (\mathsf{branch}\ \kappa) \cdot \mathcal{C}(S, w, \kappa'))\\ \mathcal{C}(\mathit{if}\ e\ \mathit{then}\ S_1\ \mathit{else}\ S_2, w, \kappa) & = \mathcal{C}(e, w)\ \nu\kappa'\ (\mathsf{branch}\ \kappa') \cdot \mathcal{C}(S_1, w, \kappa)\ \kappa' : \mathcal{C}(S_2, w, \kappa)\end{array}$$

*où certaines adresses sont traitées de façon symbolique. Dans cet exercice, on souhaite définir une nouvelle fonction de compilation où les adresses sont toujours calculées explicitement.*

1. *Définissez une fonction* $\mathit{size}(S)$ *qui calcule le nombre d'instructions présentes dans la compilation de* $S$ *(on suppose que la fonction* $\mathit{size}(e)$ *est déjà définie sur les expressions).*
   SOL. $\mathit{size}(x := e) = \mathit{size}(e)+3$. $\mathit{size}(S_1; S_2) = \mathit{size}(S_1)+\mathit{size}(S_2)$. $\mathit{size}(\mathit{while}\ e\ \mathit{do}\ S) = \mathit{size}(e) + \mathit{size}(S) + 1$. $\mathit{size}(\mathit{if}\ e\ \mathit{then}\ S_1\ \mathit{else}\ S_2) = \mathit{size}(e) + \mathit{size}(S_1) + \mathit{size}(S_2) + 1$.
2. *Utilisez la fonction size pour définir une fonction de compilation (sans adresses symboliques !)* $\mathcal{C}(S, w, i, \kappa)$ *qui compile la commande* $S$ *par rapport à une liste de variables* $w$*, en sachant que l'adresse de la première instruction du code compilé est* $i$ *et que l'adresse de la première instruction à exécuter après* $S$ *est* $\kappa$*.*
   *Par exemple, si* $\mathit{size}(e) = 1$*, alors la fonction* $\mathcal{C}(x := e; y := e, w, 12, 33)$ *pourrait être une liste d'instructions de la forme suivante :*

$$\mathcal{C}(e, w) \cdot (\mathsf{load}\ i(x, w)) \cdot (\mathsf{write}) \cdot (\mathsf{goto}\ 16) \cdot \mathcal{C}(e, w) \cdot (\mathsf{load}\ i(y, w)) \cdot (\mathsf{write}) \cdot (\mathsf{goto}\ 33)$$

   *qui est mémorisée entre les adresses 12 et 19.*

   SOL.

   $\mathcal{C}(x := e, w, i, \kappa) = \mathcal{C}(e, w) \cdot (\mathsf{load}\ i(x, w)) \cdot (\mathsf{write}) \cdot (\mathsf{goto}\ \kappa)$

   $\mathcal{C}(S_1; S_2, w, i, \kappa) = \mathcal{C}(S_1, w, i, i + \mathit{size}(S_1)) \cdot \mathcal{C}(S_2, w, i + \mathit{size}(S_1), \kappa)$

   $\mathcal{C}(\mathit{while}\ e\ \mathit{do}\ S, w, i, \kappa) = \mathcal{C}(e, w) \cdot (\mathsf{branch}\ \kappa) \cdot \mathcal{C}(S, w, i + \mathit{size}(e) + 1, i)$.

   $\mathcal{C}(\mathit{if}\ e\ \mathit{then}\ S_1\ \mathit{else}\ S_2, w, \kappa) = \mathit{let}\ j = i + \mathit{size}(e) + \mathit{size}(S_1) + 1\ \mathit{in}\ \mathcal{C}(e, w)(\mathsf{branch}\ j) \cdot \mathcal{C}(S_1, w, i + \mathit{size}(e) + 1, \kappa) \cdot \mathcal{C}(S_2, w, j, \kappa)$.

**Exercice 4 (2,5 points)** *Dans l'étude de la méthode de* marquage et ramassage *(*mark and sweep*) nous avons considéré l'algorithme non-récursif ci-dessous qui effectue une visite* en profondeur *d'un graphe à l'aide d'une* pile *qui initialement contient la racine du graphe.*

$$\begin{array}{l}\mathit{while}\ sp \neq \mathit{nil}\ \mathit{do}\\ \quad v := \mathit{pop}(sp);\\ \quad v.\mathit{mark} := 1;\\ \quad \forall w(w\ \mathit{pointer\ in\ cell}\ v\ \mathit{and}\ w.\mathit{mark} = 0)\ \mathit{do}\ \mathit{push}(w, sp)\end{array}$$

*Comment peut-on modifier les structures de données de cet algorithme pour qu'il visite le graphe* en largeur *?*

*Rappel : considérez un arbre binaire avec racine 1 dont les fils sont 2 et 3, et tel que les fils de 2 sont 4 et 5 et les fils de 3 sont 6 et 7. Dans une visite* en profondeur *(de gauche a droite) on visite les noeuds dans l'ordre 1,2,4,5,3,6,7 alors que dans une visite* en largeur *on visite les noeuds dans l'ordre 1,2,3,4,5,6,7.*

Sol. Il suffit de remplacer la pile par une file FIFO. L'opération *push* insère à la fin de la file et l'opération *pop* récupère le premier élément de la file.

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

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

1. *Trouvez un type $\tau$ tel que $\emptyset \vdash \lambda x.\lambda y.x(yx) : \tau$ est dérivable et explicitez la dérivation.*
   Sol. $(\tau \to \tau') \to (((\tau \to \tau') \to \tau) \to \tau')$. La construction de la dérivation associée est immédiate.
2. *Soient $\tau_1, \tau_2, \tau_3$ trois types* différents*. Trouvez, un $\lambda$-terme $e$ tel que $\emptyset \vdash e : (\tau_1 \to \tau_2) \to ((\tau_2 \to \tau_3) \to (\tau_1 \to \tau_3))$ est dérivable et explicitez la dérivation.*
   Sol. $\lambda f.\lambda g.\lambda x.g(fx)$. La construction de la dérivation associée est immédiate.