Examen Analyse Syntaxique et Compilation*

Exercice 1 (6 points) *On considère la grammaire G suivante avec S symbole initial :*

$$S \to e \qquad e \to () \mid (e) \mid ee$$

(1.1) *Déterminez si la grammaire G est LL(1) et si elle est LR(0).*
Par ailleurs, on considère la variante G' suivante :

$$S \to e\$ \qquad e \to () \mid (e) \mid ee$$

(1.2) *Pour la grammaire G' aussi déterminez si elle est LL(1) et si elle est LR(0).*
NB *Les réponses sans justification ne seront pas prises en compte.*

Exercice 2 (6 points) *On se place dans le cadre du langage à objets étudié dans le cours et implémenté dans le projet. On considère un fragment du langage auquel on a ajouté les expressions* fail *et* catch(e, e') *:*

$$e ::= id \mid \mathsf{new}\ C(e, \ldots, e) \mid (e\ \mathsf{as}\ C) \mid \mathsf{fail} \mid \mathsf{catch}(e, e')$$

Un jugement d'évaluation pour les expressions a la forme $(e, \eta, \mu) \Downarrow (u, \mu')$ où η est un environnement, μ et μ' sont des mémoires et u est ou bien une valeur ou bien fail *(donc* fail *n'est pas une valeur).*
(2.1) *Proposez des règles d'évaluation pour les expressions qui respectent les conditions suivantes :*

- *L'évaluation des expressions sans* fail *et* catch *se passe normalement. On rappelle de suite les règles pour ce cas :*

$$\frac{}{(x, \eta, \mu) \Downarrow (\eta(x), \mu)} \qquad \frac{(e, \eta, \mu) \Downarrow (C(\vec{\ell}), \mu') \quad C \leq D}{(e\ \mathsf{as}\ D, \eta, \mu) \Downarrow (C(\vec{\ell}), \mu')}$$

$$\frac{(e_1, \eta, \mu) \Downarrow (v_1, \mu_1), \ldots, (e_n, \eta, \mu_{n-1}) \Downarrow (v_n, \mu_n), \quad \ell_1, \ldots, \ell_n = new(\mu, n)}{(\mathsf{new}\ C(e_1, \ldots, e_n), \eta, \mu) \Downarrow (C(\ell_1, \ldots, \ell_n), \mu_n[v_1, \ldots, v_n/\ell_1, \ldots, \ell_n])}$$

- *L'expression $(D()\ \mathsf{as}\ C)$ s'évalue en* fail *si D n'est pas une sous-classe de C.*
- *Si l'expression e s'évalue en une valeur alors l'expression* catch(e, e') *s'évalue comme e.*
- *L'expression* catch(fail, e') *s'évalue comme e' (L'idée est qu'un échec provoqué par* fail *se propage et entraîne l'arrêt du programme sauf s'il est traité par un* catch*).*

*Université Paris Diderot (Paris 7). Licence d'Informatique (L3). Première session 2008-2009. Durée 2h. L'utilisation de documents ou de dispositifs électroniques est interdite.

On dispose de deux classes C et D sans attributs et telles que $C \leq D$ mais $D \not\leq C$.
(2.2) *Utilisez vos règles pour évaluer les expressions suivantes dans un environnement et une mémoire vides :*

$$\begin{array}{lcl} e_1 & = & \mathsf{catch}(\quad (\mathsf{new}\ D())\ \mathsf{as}\ C, \quad \mathsf{new}\ C() \quad) \\ e_2 & = & \mathsf{catch}(\quad \mathsf{fail}, \quad (\mathsf{new}\ C())\ \mathsf{as}\ D \quad) \\ e_3 & = & \mathsf{catch}(\quad \mathsf{catch}(\ \mathsf{fail},\ \mathsf{new}\ C()\ ), \quad \mathsf{new}\ D() \quad) \end{array}$$

Exercice 3 (4 points) *On se place à nouveau dans le cadre du langage à objets étudié dans le cours et implémenté dans le projet. Cette fois on considère les expressions suivantes :*

$$e ::= id \mid \mathsf{new}\ C(e, \ldots, e) \mid e.f$$

où id est la catégorie syntaxique des identificateurs et f celle des attributs. L'objectif est de définir une fonction de compilation $\mathcal{C}(e, w)$, où w est une liste d'identificateurs, pour une machine virtuelle qui s'inspire de celle étudiée pour le langage impératif. Dans la suite on rappelle et, au passage, on adapte certaines instructions de la machine virtuelle. Un bloc d'activation a la forme $(\ldots, pc, u_1, \ldots, u_m)$ où pc est le compteur ordinal et u_i dénote soit une valeur $C(\ell_1, \ldots, \ell_n)$ d'un objet soit une location ℓ.

- build *C n : on remplace n locations $\ell_1 \cdots \ell_n$ au sommet de la pile par la valeur $C(\ell_1, \ldots, \ell_n)$ et on incrémente le compteur ordinal. Ici C est le nom d'une classe.*
- load *n : on copie l'n-ième élément de la pile au sommet de la pile et on incrémente le compteur ordinal.*
- prj *j : si l'élément au sommet de la pile est une valeur $\mathsf{D}(\ell_1, \ldots, \ell_n)$ avec $1 \leq j \leq n$ alors on remplace cette valeur par ℓ_j et on incrémente le compteur ordinal.*
- new *on génère une nouvelle location ℓ, on écrit la valeur au sommet de la pile dans la location, on remplace la valeur par ℓ au sommet de la pile et on incrémente le compteur ordinal.*
- 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.*

(3.1) *Donnez les règles pour la compilation des expressions.*
(3.2) *Générez le code associé à l'expression $e = (\mathsf{new}\ C(\mathsf{new}\ D(), x)).f$, par rapport à la liste d'identificateurs $w = x \cdot y \cdot x$ où l'on sait que f correspond au premier attribut de la classe C.*

Exercice 4 (4 points) *On se place dans le cadre du langage fonctionnel étudié dans le cours dont on rappelle les règles de typage :*

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

(4.1) *Montrez par récurrence sur la hauteur de la preuve que si $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$ est dérivable alors $(\tau_1 \to \cdots (\tau_n \to \tau) \cdots)$ est une tautologie[1] de la logique propositionnelle où l'on interprète $\to$ comme une implication et les types de base comme des variables propositionnelles.*
(4.2) *Déduire de (4.1) qu'il n'existe pas un terme e tel que le jugement suivant est dérivable pour tout τ_1, τ_2, τ_3 :*

$$\emptyset \vdash e : (\tau_1 \to (\tau_2 \to \tau_3)) \to (\tau_2 \to \tau_3)\ .$$

[1] A savoir une formule qui est vraie pour toute affectation de valeurs de vérité. Par exemple, $(\tau \to \tau)$ est une tautologie, mais $(\tau \to \tau) \to \tau$ ne l'est pas.

Corrigé

Exercice 1

1. G n'est pas LL(1) car $\mathit{First}(\ ()\ ) = \mathit{First}(\ (e)\ ) = \{(\}$.
2. G n'est pas LR(0) car le langage généré n'a pas la propriété du préfixe. Par exemple $(), ()() \in \mathcal{L}(G)$.
3. G' n'est pas LL(1) pour la même raison que G.
4. G' n'est pas LR(0). On construit une partie de l'AFN qui reconnaît les préfixes admissibles et on remarque que dans l'AFD associé il y a un état qui contient un item complet ainsi que d'autres items. On pose :

$$0 = q_0, 1 = S \to \cdot e\$, 2 = e \to \cdot ee, 3 = e \to e \cdot e, 4 = e \to ee\cdot$$

On a les transitions :

$$0 \xrightarrow{\epsilon} 1 \xrightarrow{\epsilon} 2 \xrightarrow{e} 3 \xrightarrow{e} 4 \text{ et aussi } 3 \xrightarrow{\epsilon} 2$$

Donc en lisant ee l'AFD peut se retrouver dans un état qui contient les items 3 et 4 et l'item 4 est complet.

Exercice 2

Les règles d'évaluation :

$$\frac{}{(\mathsf{fail}, \eta, \mu) \Downarrow (\mathsf{fail}, \mu)}$$

$$\frac{(e, \eta, \mu) \Downarrow (u, \mu') \quad u = \mathsf{fail} \text{ ou } u = C(\vec{\ell}) \text{ et } C \not\leq D}{(e \ \mathsf{as}\ D, \eta, \mu) \Downarrow (\mathsf{fail}, \mu')}$$

$$\frac{(e, \eta, \mu) \Downarrow (v, \mu')}{(\mathsf{catch}(e, e'), \eta, \mu) \Downarrow (v, \mu')}$$

$$\frac{(e, \eta, \mu) \Downarrow (\mathsf{fail}, \mu') \quad (e', \eta, \mu') \Downarrow (u, \mu'')}{(\mathsf{catch}(e, e'), \eta, \mu) \Downarrow (u, \mu'')}$$

$$\frac{(e_1, \eta, \mu) \Downarrow (v_1, \mu_1), \ldots, (e_{i-1}, \eta, \mu_{i-2}) \Downarrow (v_i, \mu_i), \quad (e_i, \eta, \mu_{i-1}) \Downarrow (\mathsf{fail}, \mu_i)}{(\mathsf{new}\ C(e_1, \ldots, e_n), \eta, \mu) \Downarrow (\mathsf{fail}, \mu_i)}$$

1. $(\mathsf{new}\ D())\ \mathsf{as}\ C$ s'évalue en fail car $D \not\leq C$ donc e_1 s'évalue en $C()$.
2. e_2 s'évalue comme $(\mathsf{new}\ C())\ \mathsf{as}\ D$ qui s'évalue en $C()$ car $C \leq D$.
3. $\mathsf{catch}(\mathsf{fail}, \mathsf{new}\ C())$ s'évalue en $C()$ et donc e_3 s'évalue en $C()$ aussi.

Exercice 3

En suivant la notation du cours :

$$\begin{array}{ll} \mathcal{C}(x,w) & = (\mathsf{load}\ i(x,w)) \\ \mathcal{C}(e.f,w) & = \mathcal{C}(e,w)\cdot(\mathsf{prj}\ j)\cdot(\mathsf{read}) \quad (*) \\ \mathcal{C}(\mathsf{new}\ C(e_1,\ldots,e_n),w) & = \mathcal{C}(e_1,w)\cdot(\mathsf{new})\cdots\mathcal{C}(e_n,w)\cdot(\mathsf{new})\cdot(\mathsf{build}\ C\ n) \end{array}$$

où dans (*) on suppose que f est le j-ème attribut de la classe de l'objet e. Le code généré est :

$$(\mathsf{build}\ D\ 0)(\mathsf{new})(\mathsf{load}\ 3)(\mathsf{new})(\mathsf{build}\ C\ 2)(\mathsf{prj}\ 1)(\mathsf{read})$$

Exercice 4

Si τ est un proposition et ρ est une affectation de valeurs de vérité alors $[\![\tau]\!]\rho$ dénote la valeur de vérité de τ par rapport à l'affectation ρ.

- Si la dérivation est un axiome alors la proposition a la forme

  $$\tau_1 \to \cdots \tau_n \to \tau_i$$

  qui est une tautologie.
- Si la dérivation termine avec une abstraction alors la conclusion suit directement de l'hypothèse inductive.
- Si la dérivation termine avec une application alors par hypothèse inductive on sait que (1) $\tau_1 \to \cdots \tau_n \to \tau \to \tau'$ est une tautologie et (2) $\tau_1 \to \cdots \tau_n \to \tau$ est une tautologie aussi. Mais alors $\tau_1 \to \cdots \tau_n \to \tau'$ est une tautologie car pour toute affectation de valeurs de vérité ρ, si $[\![\tau_i]\!]\rho = 1$ pour $i = 1,\ldots,n$ alors par (2) $[\![\tau]\!]\rho = 1$ et donc par (1) $[\![\tau']\!]\rho = 1$.

Par ailleurs il ne peut pas y avoir un tel terme car la proposition en question n'est pas une tautologie. Il suffit de prendre $[\![\tau_1]\!]\rho = 0$, $[\![\tau_2]\!]\rho = 1$ et $[\![\tau_3]\!]\rho = 0$.