

Tous documents autorisés. Sujet : 3 pages

Exercice 1 : Couture, analyse LL(1) et grammaire attribuée

On s'intéresse à la description de tracés décrivant les mouvements de la tête d'une machine à coudre. La machine effectue une suite de tracés mais pour simplifier on ne s'intéresse qu'à la description d'un seul tracé. La tête de la machine peut être en position haute (terminal **UP**, indique qu'on lève la tête), et alors elle ne pique pas, ou en position basse (terminal **DOWN**, indique qu'on baisse la tête) et alors elle se déplace en piquant. Il est possible de faire avancer la tête (en piquant ou non) d'une distance donnée en mm (terminal **ENTIER**). Il est possible de changer l'orientation de la tête en la faisant tourner d'un angle donné en degrés (même terminal **ENTIER**) dans le sens des aiguilles d'une montre (sens horaire - terminal **HOR**) ou le sens contraire (sens anti-horaire - terminal **ANTIHOR**). Si le sens n'est pas indiqué, par défaut le sens anti-horaire est choisi. Initialement la tête est en position haute avec une orientation de 0 degrés. Le tracé commence éventuellement par des déplacements sans piquer pour amener la tête à l'emplacement de la couture, suivi par des déplacements en piquant. Au début de chaque déplacement il est possible de modifier l'orientation de la tête (non terminal *modOr*) en lui indiquant de quel angle elle doit tourner.

On suppose donnée la grammaire G_c suivante, d'axiome *trace*, de terminaux {TURN, ENTIER, HOR, ANTIHOR, UP, DOWN} et de productions :

$$\begin{aligned} \text{trace} &\rightarrow \text{listeDepl DOWN listeDepl UP} \mid \text{DOWN listeDepl UP} \\ \text{listeDepl} &\rightarrow \text{depl suite} \\ \text{suite} &\rightarrow \epsilon \mid \text{listeDepl} \\ \text{depl} &\rightarrow \text{modOr ENTIER} \\ \text{modOr} &\rightarrow \epsilon \mid \text{TURN dir ENTIER} \\ \text{dir} &\rightarrow \text{HOR} \mid \text{ANTIHOR} \mid \epsilon \end{aligned}$$

Q 1.1 : Calculer pour G_c les ensembles *Premier* et *Suivant*. Donner le détail des calculs et un tableau récapitulatif du résultat. □

Q 1.2 : Donner la table d'analyse LL(1) de G_c et justifier que G_c est LL(1). Vous pouvez si vous le souhaitez numéroter sur votre copie les productions de la grammaire et remplir la table avec ces numéros. □

On souhaite coder un analyseur syntaxique récursif descendant pour G_c . On suppose donné un type Java **TypeSymboles** définissant comme des constantes statiques les symboles terminaux de la grammaire TURN, ENTIER, HOR, ANTIHOR, DOWN, UP plus le marqueur de fin de fichier EOF. On s'autorisera à abrévier **TypeSymboles** en TS, on écrira par exemple TS.UP. On utilisera la méthode **public void consommer(TypeSymboles t) throws ScannerException** et l'attribut courant de type **TypeSymboles** vus en cours.

Q 1.3 : Donner les méthodes Java (signature et corps) de l'analyseur récursif descendant qui reconnaissent respectivement le non-terminal *suite* et le non-terminal *trace*, avec levée d'une exception **ParserException** en cas d'erreur syntaxique et **ScannerException** en cas d'erreur lexicale. □

On souhaite maintenant attribuer cette grammaire pour associer à l'axiome la liste des déplacements effectués, en indiquant pour chaque déplacement sa longueur, la position haute ou basse de la tête, et son orientation effective (ainsi que d'autres informations comme le fil utilisé ou sa couleur, qu'on ne traite pas ici). L'orientation effective est l'angle courant de la tête pendant le déplacement, après éventuelle modification de l'orientation précédant le déplacement. Considérons par exemple le mot suivant dans lequel la valeur associée au terminal ENTIER est indiquée dessous :

DOWN	ENTIER	TURN	HOR	ENTIER	ENTIER	TURN	ANTIHOR	ENTIER	ENTIER	UP
	5 (mm)			45 (degrés)	10 (mm)			30 (degrés)	20 (mm)	

Le premier déplacement de 5mm se fait avec l'orientation initiale de 0 degrés. Le second déplacement de 10mm se fait avec une orientation de $0-45 = -45$ degrés (sens horaire = angle négatif). Le troisième déplacement de 20mm se fait avec une orientation de $-45+30 = -15$ degrés (sens anti-horaire = angle positif).

Les questions 1.4 à 1.6 ont pour but de construire progressivement la grammaire attribuée. On pourra s'aider d'un arbre syntaxique pour le mot ci-dessus. On pourra aussi commencer par réfléchir à l'attribution de la grammaire G'_c , version non LL(1) de G_c mais sans doute plus intuitive, obtenue en remplaçant les trois productions

Liste	Pos	Dir
Liste()	UP	HOR
Liste(Deplacement d)	DOWN	ANTIHOR
ajoutTete(Deplacement d)	*	Deplacement
ajoutQueue(Deplacement d)		
concat(Liste l) : Liste		Deplacement(Pos p, int angle, int lg)

FIG. 1 – Types de données de l'exercice 1

(1) par les deux productions (2) ;

$$(1) \quad \begin{aligned} listeDepl &\rightarrow depl \ suite \\ suite &\rightarrow \epsilon \mid listeDepl \end{aligned}$$

$$(2) \quad listeDepl \rightarrow depl \mid depl \ listeDepl$$

On suppose donnés les types de données à la Java de la figure 1 (on pourra donc écrire `Pos.UP` et `Dir.HOR`). Si une production n'a pas besoin d'être attribuée, inutile de la recopier. **Pour les questions 1.4 à 1.6 on donnera pour chaque attribut utilisé** le symbole de G_c auquel il est associé, son type, et s'il est hérité ou synthétisé (et s'il est synthétisé, s'il est fixé par l'analyseur lexical le cas échéant).

Q 1.4 : Associer à chaque déplacement sa longueur en mm et la position haute ou basse de la tête. □

Q 1.5 : Associer à chaque déplacement l'orientation effective de la tête. □

Q 1.6 : En tenant pour acquises les attributions précédentes (même si vous n'avez pas répondu aux questions 1.4 et 1.5), associer à chaque déplacement un objet de type **Deplacement** et au tracé un objet de type **Liste** contenant la liste des déplacements effectués dans le tracé. □

Exercice 2 : Analyse ascendante

Soit la grammaire $G_1 = (S, V_{T1}, V_{N1}, P_1)$ avec $V_{T1} = \{a, b, c, d\}$, $V_{N1} = \{S, A\}$ et $P_1 = \{ S \rightarrow Aa \mid bAc \mid dc \mid bda, A \rightarrow d \}$

Q 2.1 : Construire l'automate LR-AFD de G_1 (11 états). □

Q 2.2 : Expliquer pourquoi G_1 n'est ni LR(0), ni SLR(1) en explicitant les conflits LR(0) et SLR(1). □

Q 2.3 : On souhaite reconnaître le mot $m_1 = bda$ avec un analyseur SLR(1). Donner la suite des piles résultant de l'analyse de m_1 en indiquant à quel moment l'analyseur doit faire un choix, quel choix conduit à l'acceptation de m_1 et quel choix conduit à l'échec (donner dans les 2 cas la suite des piles correspondante). □

Q 2.4 : G_1 est-elle LR(1) ? Justifier en construisant le moins possible d'états dans l'automate LR(1). □

Exercice 3 : TP Ava

On souhaite rajouter à AVA une boucle «for» de la forme :

```
for ( var := expri ; exprb ; var := expri ) loopfor listeInstruction end loopfor ;
```

où **var** est la variable de boucle, de type entier et préalablement déclarée, **exprb** est une expression de type booléen et **expri** est une expression de type entier (**expri** et **exprb** respectant par ailleurs les conventions AVA propres aux expressions, de même pour l'affectation **var := expri**). La première affectation sert à initialiser

la variable de boucle. La seconde affectation est effectuée à la fin du corps, à chaque tour de boucle. Le `end loopfor` est formé comme le `end loop` existant déjà en AVA. On écrira par exemple :

```
int i;
for ( i := 0 ; i >= 0 and i < 30 ; i := i + 1 ) loopfor
    writeln(%i,i);
end    loopfor;
```

ou des bizarreries syntaxiquement et sémantiquement correctes comme :

```
int j, i; for ( j := 5; i >= j; j := 3 ) loopfoor end loopfor;
mais pas comme : int i; for (j := true; 3; i := j ) loopfor end loopfor;
```

Q 3.1 : Détailler les ajouts à apporter à votre compilateur au niveau de l'analyseur lexical (en suivant la syntaxe de JFLEX) et de l'analyseur syntaxique (en suivant la syntaxe de CUP). □

Q 3.2 : En utilisant les instructions de bytecode du TP6 et du cours `iload <index>`, `ldc <index>`, `iadd`, `istore <index>`, `iread`, `iprint` et `sprint`, et en supposant que la variable `x` est à l'index 0 et la constante `"x+1 : "` est à l'index 3, donner (sans numéroter les instructions) le bytecode Java que votre compilateur est supposé engendrer pour le code `read x; write(%s,"x+1 : "); write(%i, x+1);`. □