Licence ST Informatique – S5 – 2006/2007

Compilation – 1ère session

samedi 20 janvier 2007

documents autorisés, calculettes interdites — durée 3 heures

Exercice 1 : Grammaires algébriques

On considère la grammaire $G_1 = (X_1, V_1, S, P_1)$ où $X_1 = \{\ a\ ,\ b\ \}$, $V_1 = \{\ S\ ,\ S_1\ ,\ S_2\ \}$, et $P_1 = \{$

$$\begin{array}{lcl} S & \longrightarrow & S_1 \text{ b } S_1 \mid S_2 \text{ b } S_2 \\ S_1 & \longrightarrow & a\ S_1\ a \mid \varepsilon \\ S_2 & \longrightarrow & a\ S_2\ a \mid a \end{array}$$

$\}$.

Question 1 : La grammaire G_1 est-elle $LL(1)$, $SLR(1)$? Justifier.

Question 2 : La grammaire G_1 est-elle ambiguë ? Justifier.

Question 3 : Trouver une grammaire régulière engendrant le même langage que la grammaire G_1.

Exercice 2 : Analyse ascendante

Dans un logiciel de saisie de notes, les formules pour le calcul des moyennes coefficientées adoptent la syntaxe suivante : Une note coefficientée est un calcul de note suivi du symbole `':'`, suivi d'une valeur entière (le coefficient). Quand le symbole `':'` et la valeur entière sont omis alors le coefficient associé vaut implicitement 1. La valeur d'une note coefficientée est le produit de la valeur du calcul de note par la valeur du coefficient. Un calcul de note est soit un entier (une note unique), soit une liste de notes coefficientées, séparées par des point-virgules, encadrées par des crochets. La valeur d'une liste de notes coefficientées se calcule comme la moyenne coefficientée des notes coefficientées c'est à dire comme la somme des valeurs des notes coefficientées le tout divisé par la somme des coefficients.

Exemple 1

- *L'expression* `[ [ 3:2 ; 4:3 ; 0 ]:3 ; [ 7 ; 4:2 ]:3 ]:2` *est une note coefficientée syntaxiquement correcte. Sa valeur est la moyenne des valeurs de deux notes coefficientées :* n_1 *valant* $\left(\frac{3\times2+4\times3+0\times1}{2+3+1}\right)\times 3 = 9$ *et* n_2 *valant* $\left(\frac{7\times1+4\times2}{1+2}\right)\times 3 = 15$. *On obtient donc finalement comme valeur* $\frac{n_1+n_2}{3+3}\times 2 = 8$.
- *L'expression* `[ [ 3:2 ; 4 ]:2 ; 4:2 ]` *est une note coefficientée syntaxiquement correcte qui vaut 3 (les divisions sont des divisions entières).*
- *L'expression* `2:3` *est une note coefficientée syntaxiquement correcte qui vaut 6*

Pour formaliser la syntaxe des notes coefficientées, on se donne la grammaire suivante $G = (X, V, N', P)$ avec $X = \{[\ ,\ ;\ ,\ \texttt{int}\ ,\ ]\}$, $V = \{N', N, C, L\}$ et $P = \{$

$$\begin{array}{lcl} N' & \longrightarrow & N \\ N & \longrightarrow & C \mid C : \texttt{int} \\ C & \longrightarrow & \texttt{int} \mid [\ L\ ] \\ L & \longrightarrow & L\ ;\ N \mid N \end{array}$$

$\}$

Question 1 : Construire les ensembles `premiers` et `suivants` pour les variables de la grammaire G.

Question 2 : Construire l'automate $LR(0)$ de la grammaire G.

Question 3 : Donner la table `action` $SLR(1)$ de la grammaire G.

Question 4 : La grammaire G est-elle $LL(1)$, $LR(0)$, $SLR(1)$, ambiguë ? Justifier.

Question 5 : Décorer la grammaire G d'actions sémantiques permettant le calcul des notes coefficientées lors de leur analyse syntaxique. Préciser les attributs utilisés en indiquant pour chaque attribut son type et s'il s'agit d'un attribut hérité ou synthétisé.

Exercice 3 : Analyse récursive descendante

Le langage de programmation `JÉDUBOL` permet l'écriture de programmes simples utilisant la notion de successeur, de prédécesseur et de remise à zéro de variables ne contenant que des valeurs entières positives ou nulles, ainsi que des boucles tant que ou des si alors sinon basés sur des tests simples de la forme x $/=$ y.

Exemple 2 *Le programme* `JÉDUBOL` *suivant calcule dans la variable x le pgcd du contenu (strictement positif) de la variable x et du contenu (strictement positif) de la variable y*

```
debut
  raz(z) ; raz(a) ;
  tant que x /= y faire
    debut
      pred(x) ;  pred(y) ; succ(a) ;
      tant que a /= z faire
        si x /= z alors
          si y /= z alors debut pred(x) ; pred(y) ; succ(a)  fin
          sinon tant que a /= z faire debut succ(y) ; pred(a)   fin
        sinon tant que a /= z faire debut succ(x) ; pred(a)  fin
    fin
fin
```

On souhaite réaliser l'analyse syntaxique de tels programmes à l'aide d'un analyseur récursif descendant. Au cours de cette analyse, il n'est pas nécessaire de distinguer les différentes actions élémentaires possibles (`pred(`variable`)`, `succ(`variable`)` ou `raz(`variable`)`) qui seront toutes représentées par le seul terminal `elem`. Pour les mêmes raisons, toute construction de la forme `tant que` variable `/=` variable `faire` sera représentée par le terminal `boucle` et toute construction de la forme `si` variable `/=` variable `alors` sera représentée par le terminal `test`.

Exemple 3 *Pour le programme de l'exemple 2, on obtient*[1] *:*

```
debut
  elem ; elem ;
  boucle
    debut
      elem ; elem ; elem ;
      boucle
        test
          test debut elem ; elem ; elem  fin
          sinon boucle debut elem ; elem  fin
        sinon boucle debut elem ; elem  fin
    fin
fin
```

[1] Toutes ces simplifications peuvent clairement être réalisées au cours de l'analyse lexicale.

Pour tenir compte de ces simplifications, on dispose de la grammaire G' suivante, d'alphabet terminal $Y = \{$`debut` , `fin` , ; , `boucle` , `test` , `sinon` , `elem`$\}$ d'axiome P et d'ensemble de règles $R = \{$

$P \longrightarrow$ `debut` S `fin`
$S \longrightarrow A\ R$
$R \longrightarrow$; $A\ R \mid \varepsilon$
$A \longrightarrow P \mid T \mid I \mid$ `elem`
$T \longrightarrow$ `boucle` A
$I \longrightarrow$ `test` A `sinon` A

$\}$

Question 1 : Construire les ensembles `premiers` et `suivants` pour les variables de la grammaire G'.

Question 2 : Construire la table d'analyse $LL(1)$ de la grammaire G'.

Le professeur de `JÉDUBOL` décide de mettre en place une application d'aide à la notation des nombreux projets d'étudiants qu'il a à corriger. Pour cela, il définit deux mesures (ou métriques) de la qualité des logiciels `JÉDUBOL` à partir de la représentation du programme sous forme d'arbre programmatique. Un arbre programmatique permet de représenter graphiquement la structure d'un programme ; l'arbre présenté figure 1 est l'arbre programmatique[2] du programme de l'exemple 2.

- La première mesure est le nombre de parcours possibles du programme ; elle est définie par :
  - Le nombre de parcours possibles d'une action élémentaire est 1.
  - Le nombre de parcours possibles d'une séquence est le produit des nombres de parcours possibles des éléments de cette séquence.
  - Le nombre de parcours possibles d'une alternative est la somme du nombre de parcours possibles de la partie alors et de celui de la partie sinon.
  - Le nombre de parcours possibles d'une itération est égal au nombre de parcours possibles du corps de la boucle augmenté de 1.
- La seconde mesure est la complexité structurelle du programme. Elle est définie comme la somme des poids des feuilles de l'arbre programmatique ; le poids d'une feuille est défini comme étant la somme des coûts des noeuds du chemin menant de la racine à cette feuille avec le coût d'un noeud défini par :
  - Le coût d'une feuille est nul.
  - Le coût d'une séquence est 1.
  - Le coût d'une alternative est 2.
  - Le coût d'une itération est 4.

Exemple 4 *L 'arbre programmatique correspondant au programme de l'exemple 2 est donné figure 1. Pour ce programme, le nombre de parcours possibles est 7 et sa complexité structurelle est 137 (=1+1+6+6+6+15+15+15+19+19+17+17 si on liste les poids des feuilles dans l'ordre d'apparition dans le programme).*

Question 3 : Donner le nombre de parcours possibles et la complexité structurelle du programme suivant :

```
debut
  succ (x) ;
  si x /= y alors pred(y)
  sinon tant que y /= z faire succ(z) ;
  si a /= z alors raz(x)
  sinon raz(y)
fin
```

[2] L'arbre programmatique permet de définir les mesures de qualité, mais n'est pas à construire pour donner la valeur des mesures de qualité d'un programme

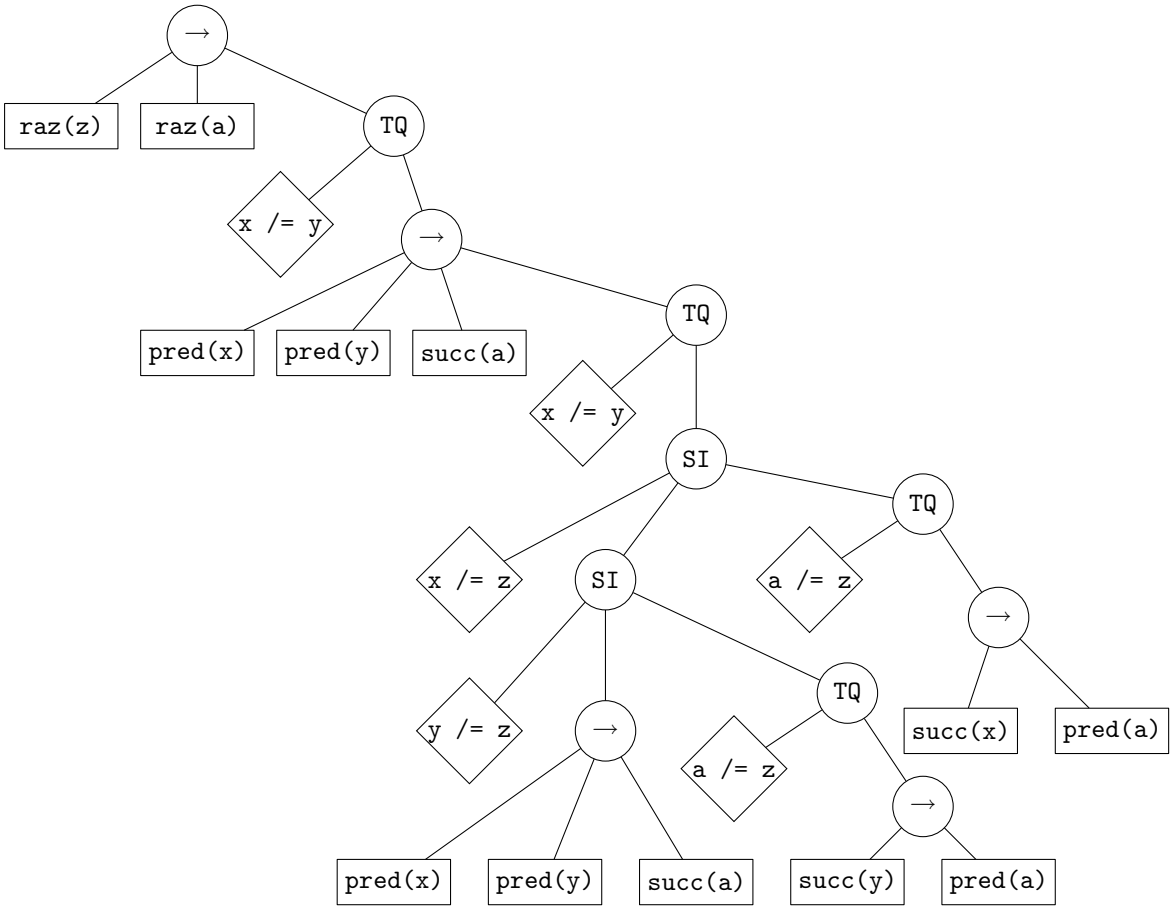

FIG. 1 – L'arbre programmatique du programme de l'exemple 2

Question 4 : Associer à chaque variable de la grammaire G' un ou plusieurs attributs permettant de calculer le nombre de parcours possibles et la complexité structurelle d'un programme `JÉDUBOL` au cours de l'analyse récursive descendante. Pour chacun des attributs, préciser s'il s'agit d'un attribut hérité ou synthétisé.

Question 5 : Insérer les actions sémantiques nécessaires à ce calcul dans la grammaire G'.

Question 6 : On suppose disponible un analyseur lexical dont vous préciserez la signature des méthodes publiques nécessaires. Écrire en `java` la méthode associée à la variable R faisant partie de l'analyseur récursif descendant induit par la grammaire attribuée de la question 5.