Université Joseph Fourier **Polytech' Grenoble**
Département RICM 4

Langages et Traducteurs

TD 2 et 3 : Analyse Descendante

Exercice 1. Calculez les ensembles de directeurs associés à chacune des règles de la grammaire suivante :

```
Z -> S#
S -> Xa | epsilon
X -> bX | YW
Y -> aX | epsilon
W -> c | d
```

Cette grammaire est-elle LL(1) ?

Exercice 2. Montrez que la grammaire suivante n'est pas LL(1) :

```
Z -> S#
S -> X  | Yc
X -> Xa | epsilon
Y -> Yb | d
```

Quel est le langage décrit par cette grammaire ? Proposez une grammaire LL(1) qui décrit le même langage.

Exercice 3. On considère une grammaire décrivant des expressions arithmétiques :

```
Z -> E#
E -> E + T | T
T -> T * F | F
F -> i | e | (E)
```

Cette grammaire est-elle LL(1) ? Peut-on la modifier pour la rendre LL(1) ?

Exercice 4. Est-il possible d'écrire une grammaire LL(1) qui décrit l'ensemble des palindromes construits sur le vocabulaire $V = \{a, b\}$? Justifiez votre réponse ...

Exercice 5. Soit la grammaire suivante, défini sur le vocabulaire terminal $\{ :, =, \texttt{i}, \texttt{e}, ; \}$. Cette grammaire est-elle LL(1) ? Peut-on la rendre LL(1) ?

```
Z -> S ;
S -> V := e     /* affectation */
S -> L S        /* instruction avec etiquette */
L -> i :        /* etiquette */
V -> i         /* identificateur */
```

Exercice 6 (extrait du "devoir surveillé" de novembre 2004).

On considère un langage de commandes, noté L_1, construit sur le vocabulaire { `c`, `;`, `*`, `(`, `)`, `$` }. Intuitivement, "`c`" représente une commande de base, "`;`" un opérateur binaire infixé de composition séquentielle, "`*`" un opérateur unaire post-fixé d'itération. Les parenthèses permettent de grouper plusieurs commandes et "`$`" est le caractère de fin de fichier.

La syntaxe complète du langage est donnée par la grammaire G_1 suivante :

$$\begin{array}{lcl} Z & \longrightarrow & C\,\$ \\ C & \longrightarrow & \texttt{c} \mid C\,;C \mid C* \mid (C) \end{array}$$

1. Montrez (sur une phrase très simple du langage) que G_1 est une grammaire *ambigüe*.

2. On complète maintenant la définition du langage L_1 en considérant :
– que l'opérateur "`*`" est plus prioritaire que l'opérateur "`;`".
– que l'opérateur "`;`" est associatif à gauche ("`c ; c ; c`" doit être interprété comme "`(c ; c) ; c`").
On note L_2 le langage obtenu en prenant en compte ces nouvelles contraintes.

1. Proposez une grammaire G_2, non ambigüe, qui décrive le langage L_2.
2. Donnez l'arbre de dérivation produit par G_2 pour la séquence "`c ; c ; c* $`".

3. La grammaire G_2 que vous avez proposé à la question 2 est-elle LL(1) ? Si non transformez-là en une grammaire G_2' qui soit LL(1) et qui décrive L_2. Justifiez vos réponses.

4. Illustrez le fonctionnement de l'analyseur LL(1) obtenu à la question précédente sur la séquence "`c ; c ; c* $`".