Compléments au td 3 sur les grammaires LL

Laure Gonnord

30 octobre 2008

Exos

EXERCICE 1

Soit la grammaire suivante, défini sur le vocabulaire terminal $\{ :, =, \texttt{i}, \texttt{e}, ; \}$:

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

Cette grammaire est-elle LL(1) ? Peut-on la rendre LL(1) ? Comment résoudre ce problème ?

CORRECTION :
Cette grammaire n'est pas LL(1) (pourquoi ?). Le langage décrit est $(i :)^* i := e$. On peut essayer de ré-ecrire cette grammaire :

```
Z -> X ;
X -> Y i : = e
Y -> i : Y | epsilon /* le langage de Y est (i:)* */
```

Cette grammaire n'est pas LL(1) car $Suivant(Y) = \{i\}$, donc $Dir(Y \longrightarrow i : Y) = Dir(Y \longrightarrow \varepsilon)$.
Une solution pour résoudre ce pb est de modifier l'ensemble des symboles terminaux en choisisant $V_T = \{:, aff, ;, i, e\}$ où "aff" représente `:=`. On obtient alors :

```
Z -> S ;
S -> i X
X -> aff e
X -> : S
```

Autres ?

Allez voir les liens :

1. `http://www.lsv.ens-cachan.fr/~gastin/Langages/TD-07/td12.pdf`
2. `http://www-verimag.imag.fr/~peron/index.php?menu=2,2,0,8`
3. `https://www.cs.tcd.ie/~gilberj/3ba37/ll1.pdf`
4. Caractérisation algébrique des grammaires LL/LR `http://laure.gonnord.org/site-ens/mim/compilation/cours/LL-LR.pdf`

Analyseur récursif

Cette section est entièrement recopiée d'un poly de cours de l'UJF, écrit par J-C. Fernandez, merci pour les sources.

Soit la grammaire G suivante, reconnaissant le langage $L(G) = \{1^n 0^n 0^m \mid 0 \leq n \text{ et } 0 < m\}$ (c'est la même que dans le td...) :

$$G = \begin{cases} Z \to S\$ \\ S \to AB \\ A \to 1A0 \mid \varepsilon \\ B \to 0Y \\ Y \to B \mid \varepsilon \end{cases}$$

On suppose que la séquence à reconnaître est accessible par les primitives `init_sequence()` et `avancer()`. On peut alors écrire un analyseur LL(1) pour G "à la main" (mais de manière *systématique*) en associant une procédure récursive `Rec_X` à chaque non terminal X de G. On se guide pour cela sur l'ensemble des règles qui définissent X dans G, et sur les ensembles de directeurs qui leur sont associés, comme dans l'exemple ci-dessous :

$\boxed{Z \to S\$}$ `Directeur`$(Z \to S\$) = \{1, 0\}$

```
Rec_Z :
   init_sequence() ;
   si symbole_courant() = "1" ou symbole_courant() = "0"
   alors
     Rec_S  ; Rec_terminal("$")
   sinon
     erreur()
```

$\boxed{S \to AB}$ `Directeur`$(S \to AB) = \{1, 0\}$

```
Rec_S :
   si symbole_courant() = "1" ou symbole_courant() = "0" alors Rec_A ; Rec_B ; sinon erreur()
```

$\boxed{A \to 1A0 \mid \varepsilon}$ `Directeur`$(A \to 1A0) = \{1\}$; `Directeur`$(A \to \varepsilon) = \{0\}$

```
Rec_A :
   si symbole_courant() = "1" alors
       Rec_terminal("1") ; Rec_A ; Rec_terminal("0") ;
    sinon si symbole_courant() = "0" alors
        rien ;
    sinon erreur()
```

$\boxed{B \to 0Y}$ `Directeur`$(B \to 0Y) = \{0\}$

```
Rec_B :
    si symbole_courant() = "0" alors Rec_terminal("0") ; Rec_Y ;
    sinon erreur()
```

$\boxed{Y \to B \mid \varepsilon}$ `Directeur`$(Y \to B) = \{0\}$

```
Rec_Y :
    si symbole_courant() = "0" alors
         Rec_B ;
     sinon si symbole_courant() = "$" alors
         rien ;
     sinon erreur()

Rec_terminal (t : terminal) :
      si symbole_courant() = t alors avancer()
      sinon erreur()
```