UFR d'Informatique
L3 Informatique, Math-Info et Japonais-Info

Grammaires et analyse syntaxique
Année 2023-2024

TD n°5

LL(1), le retour

Exercice 1 *On considère la grammaire suivante :*

$Z \to S\$$
$S \to D \mid XA \mid X \mid \epsilon$
$X \to bX \mid YVWV$
$Y \to aX \mid \epsilon$
$W \to c \mid d$
$V \to v \mid \epsilon$
$D \to DE$
$E \to e \mid Ee$
$F \to f$

On considère deux méthodes pour réduire la grammaire :

1. *On détermine les non-terminaux non-productifs. On les enlève de la grammaire. On détermine les non-terminaux non-accessibles de la grammaire ainsi obtenu et on les enlève.*
2. *On détermine les non-terminaux non-accessibles. On les enlève de la grammaire. On détermine les non-terminaux non-productifs de la grammaire ainsi obtenu et on les enlève.*

— *Appliquer les deux méthodes.*
— *Avec laquelle des deux méthodes on obtient une grammaire réduite ?*
— *Réduisez la grammaire.*
— *Calculer l'ensemble de non-terminaux annulables* EPS.
— *Calculer l'ensemble* FIRST_1 *de chaque non-terminal.*
— *Calculer l'ensemble* FOLLOW_1 *de chaque non-terminal.*
— *Est-ce que la grammaire est* LL(1) *?*

Exercice 2 *Soit la grammaire suivante, définie sur le vocabulaire terminal* $\{[,], i, +, -, \$\}$ *:*

$Z \to S\$$
$S \to OE$
$O \to [$
$E \to iK$
$K \to -E \mid +E \mid]$

1. *Faites la table d'analyse (c'est-à-dire un tableau avec les non-terminaux en ordonnée et les terminaux en abscisse), qui indique à chaque fois quelle règle on est censé appliquer.*
2. *Récupérez les fichiers* `parser.ml`, `reader.ml`, `tree.ml`, *etc. fournis. La compilation se fait avec*

```
dune build main.exe
```

on peut faire des tests en exécutant

```
_build/default/main.exe < test.txt
```

où le fichier `test.txt` *contient un mot à tester. Attention, il ne faut pas mettre d'espace ni de saut de ligne dans le fichier donné en entrée.*

3. *Complétez le fichier* `parser.ml` *afin de faire une analyse* LL(1) *de la grammaire* LL(1) *correspondante au langage si dessus. Le symbole $ correspond à EOF.*

Exercice 3 *On souhaite construire un analyseur grammatical des expressions arithmétiques (avec − et +) bien parenthésées engendré par la grammaire*

$$S \to n \mid (S) \mid S + S \mid S - S$$

Dans la suite de l'énoncé, on appellera L_1 ce langage. Le symbole 'n' correspondra, dans la partie programmée, à des entiers sans signes "+" ni "-". Récupérez les fichiers fournis. Cette fois-ci on vous fournit un lexer, qui servira à lire des entiers et autres mots clefs.

1. *Donnez une grammaire* LL(1) *avec axiome Z pour le langage L_1.*
   Complétez le `parser.ml` *pour qu'il fasse l'analyse.*
2. *Ajoutez la possibilité de faire des opérations avec des noms de variables, que l'on représente par un nouveau terminal 'v'. (On appellera L_2 ce langage). A partir de là, il faudra aussi modifier les autres fichiers :* `token.ml`, `tree.ml`,...
3. *A présent, on veut définir le langage L_3 "let v = a_1 and v = a_2 ... and v = a_k in b" avec les a_i dans L_1, et b dans L_2. (On appelle ce langage L_3)*