TD n°5

LL(1), le retour

Exercice 1 *On considère la grammaire suivante (l'axiome est Z) :*

$$
\begin{array}{ll}
Z \to S\$ & X \to bX \mid YVWV \\
S \to D \mid XA \mid X \mid \varepsilon & Y \to aX \mid \varepsilon \\
D \to DE & V \to v \mid \varepsilon \\
E \to e \mid Ee & W \to c \mid d \\
F \to f &
\end{array}
$$

1. *Réduire la grammaire (toutes les questions suivantes portent sur la grammaire réduite).*
2. *Calculer l'ensemble de non-terminaux annulables* EPS.
3. *Calculer l'ensemble* FIRST_1 *de chaque non-terminal.*
4. *Calculer l'ensemble* $\mathrm{FIRST}_{\leq 1}$ *de chaque membre droit d'une règle de la grammaire.*
5. *Calculer l'ensemble* FOLLOW_1 *de chaque non-terminal.*
6. *Est-ce que la grammaire est* LL(1) *?*

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

$$
\begin{array}{l}
Z \to S\$ \\
S \to OE \\
O \to [\\
E \to iK \\
K \to -E \mid +E \mid]
\end{array}
$$

1. *Faire 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. *Récupérer 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. On peut aussi donner le mot à tester directement dans la ligne de commande, par exemple :*

   ```
   _build/default/main.exe <<< "[i+i]"
   ```

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

Exercice 3 *On souhaite construire des analyseurs grammaticaux, en commençant par le langage L_1 des expressions arithmétiques (avec $-$ et $+$) bien parenthésées engendré par la grammaire*

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

Le symbole "n" correspondra, dans la partie programmée, à des entiers sans signe "+" ni "−". Récupérer les fichiers fournis. On fournit ici un lexer, qui servira à lire des entiers et autres mots-clés.

1. *Donner une grammaire* LL(1) *avec axiome Z pour le langage L_1.*
   Compléter le `parser.ml` *pour qu'il fasse l'analyse.*
2. *On définit L_2 à partir de L_1 en ajoutant la possibilité de faire des opérations avec des noms de variables, que l'on représente par un nouveau terminal "v".*
   Modifier votre parser pour reconnaître ce nouveau langage. Il faudra aussi modifier les autres fichiers : `token.ml`, `tree.ml`, `lexer.mll`.
3. *On définit le langage L_3 des expressions "let v = a_1 and v = a_2 ... and v = a_k in b" où les a_i sont dans le langage L_1, et b est dans le langage L_2.*
   Modifier à nouveau votre parser pour reconnaître ce langage.