TD6 - Grammaires (éléments de réponses)

Exercice 1

Exercice 2

Exercice 3

Le langage engendré par la grammaire est celui des mots bien parenthésés.

Exercice 4

Exercice 5

Exercice 6

Soit $\mathcal{G}_1 = (\mathcal{V}_1, \Sigma, \mathcal{R}_1, S_1)$ $\mathcal{G}_2 = (\mathcal{V}_2, \Sigma, \mathcal{R}_2, S_2)$ des grammaires. Alors :

- $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \Sigma, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 \mid S_2\}, S)$ reconnaît l'union des langages.
- $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \Sigma, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 S_2\}, S)$ reconnaît la concaténation des langages.
- $\mathcal{G} = (\mathcal{V}_1, \Sigma, \mathcal{R}_1 \cup \{S \to S_0\, S_0 \to S_1 S_0 \mid \varepsilon\}, S)$ reconnaît l'étoile de Kleene.

Exercice 7

On a $\mathcal{G} = (\{\}, \{\texttt{[}, \texttt{]}, \texttt{;}, \texttt{1}\}, \mathcal{R}, S)$ avec : $\mathcal{R} = \{S \to \texttt{[}L\texttt{]} \mid \texttt{[]}; L \to \texttt{1} \mid \texttt{1; L}\}$.

Exercice 8

Question 1. Une preuve utilisant le lemme de l'étoile directement est fastidieuse. On utilise plutôt les propriétés de stabilité et une preuve par l'absurde. Supposons D régulier. On peut remarquer que $B = (^*)^*$ est régulier. Les langages réguliers étant stables par intersection, $D \cap B$ serait régulier. Mais $D \cap B = \{(^n)^n \mid n \geq 0\}$ dont on sait qu'il n'est pas régulier, ce qui est contradictoire. D n'est donc pas régulier.

Question 2. On peut proposer : $(\{S\}, \Sigma, \{S \to [S]\, S \mid \varepsilon\}, S)$

Question 3. On utilise une pile pour assurer le bon parenthésage :

```
let closing c = match c with
  | '[' -> ']'
  | '(' -> ')'
  | '{' -> '}'
  | _ -> failwith "invalid char"

let dyck s =
  let n = String.length s in
  let rec loop stack i =
    if i = n then stack = []
    else
      match (s.[i], stack) with
        | ( ('[' | '(' | '{') as c ), _
          -> loop (c :: stack) (i+1)
        | (']' | ')' | '}'), []
          -> false
        | ( (']' | ')' | '}') as c ), p :: sstack
          -> c = closing p && loop sstack (i+1)
        | _
          -> false
  in
  loop [] 0
```

Exercice 9

Question 1. On peut commencer par réécrire la formule logique $(i \neq j) \vee (j \neq k)$ sous la forme $(i < j) \vee (i > j) \vee (j < k) \vee (j > k)$ et ainsi envisager une règle de production comme la disjonction de quatre règles. Soit $(x, y) \in \{a, b, c\}^2$ et introduisons à présent les régles suivantes :

- $S_x \to xS_x \mid \varepsilon$ qui calculent les mots de la forme x^p ;
- $E_{x,y} \to xE_{x,y}y \mid \varepsilon$ qui calculent les mots de la forme $x^q y^q$.

On peut alors proposer la règle de production suivante pour définir une CFG associée à $\mathcal{L}_0$.

$$S \to S_a\, E_{b,c}\, c\, S_c \mid S_a\, b\, S_b\, E_{b,c} \mid S_a\, a\, E_{a,b}\, S_c \mid E_{a,b}\, b\, S_b\, c$$

Question 2. Montrons par récurrence sur la taille des mots que :

$$S \to \epsilon \mid aS \mid SaSbS \mid SbSaS$$

Si un mot est de taille 2 ou moins il est généré par la grammaire. Étant donné un mot $w_1 \dots w_n \in \mathcal{L}_\Sigma$ on note $f(k) = |w_1 \dots w_k|_a - |w_1 \dots w_k|_b$.

S'il existe i tel que $f(i) = 0$ et $1 < i \leqslant n$ alors on prend le plus petit tel i et on a $w = aw_2 \dots w_{i-1} b w_{i+1} w_n$ ou $w = bw_2 \dots w_{i-1} a w_{i+1} w_n$ avec $w' = w_2 \dots w_{i-1}$ et $w'' = w_{i+1} w_n$ qui vérifient la propriété donc par récurrence $S \Rightarrow^* w'$ et $S \Rightarrow^* w''$ soit $S \to SaSbS \mid SbSaS \Rightarrow^* w$.

Si un tel i n'existe pas alors on a $f(0) = 0$ et $f(n) > 0$. Soit f est strictement croissante et $w = a^k$ (qui est généré par $S \to aS$) soit on a $1 \leqslant i < j$ avec $f(i) = f(j)$ et $0 \leqslant f(j) \leqslant f(n)$. Notez que $i + 1 < j$ car $f(i + 1) \neq f(i)$. On coupe alors w en $x = w_1 \dots w_i$, $y = w_{i+2} \dots w_{j-1}$ et $z = w_{j+1} \dots w_n$. Ces trois bouts respectent la propriété car $|x|_a - |x|_b = f(i)$, $|y|_a - |y|_b = f(j) - f(i)$ et $|z|_a - |z|_b$.

En prenant i et j minimal on a $w_{i+1} \neq w_j$. Dans le cas $w_{i+1} = a$ et $w_j = b$ (resp. $w_{i+1} = b$ et $w_j = a$) comme on a $S \Rightarrow^* x$, $S \Rightarrow^* y$, $S \Rightarrow^* z$ alors $S \to SaSbS \Rightarrow^* xaybz$ (resp. $S \to SbSaS \Rightarrow^* xbyaz$).

Question 3.

$$S \to 0 \mid 1 \mid S + S \mid S * S \mid (S)$$

Exercice 10

Proposons les règles suivantes.

$$S \to T \mid T + S \qquad T \to U * T \mid U \qquad U \to (S) \mid 0 \mid 1$$

Ainsi, on peut dériver $2 + 3 * 4$ comme suit.

$$\begin{aligned}
S &\Rightarrow T + S \\
&\Rightarrow T + T \\
&\Rightarrow T + U \\
&\Rightarrow U + (S) \\
&\Rightarrow (S) + (T) \\
&\Rightarrow (T + S) + (U * T) \\
&\Rightarrow (U + T) + ((S) * U) \\
&\Rightarrow (1 + U) + ((T + S) * (S)) \\
&\Rightarrow (1 + 1) + ((U + T + S) * (T + S)) \\
&\Rightarrow (1 + 1) + ((1 + U + T) * (U + T + S)) \\
&\Rightarrow (1 + 1) + ((1 + 1 + U) * (1 + U + T + S)) \\
&\Rightarrow (1 + 1) + ((1 + 1 + 1) * (1 + 1 + U + T)) \\
&\Rightarrow (1 + 1) + ((1 + 1 + 1) * (1 + 1 + 1 + U)) \\
&\Rightarrow (1 + 1) + ((1 + 1 + 1) * (1 + 1 + 1 + 1)) \\
&\Rightarrow 2 + (3 * 4)
\end{aligned}$$

Exercice 11

Exercice 12

Exercice 13

Exercice 14

Exercice 15

Exercice 16

Question 1. Considérons les dérivations suivantes.

$$S \Rightarrow L\# \Rightarrow EL\# \Rightarrow EEL\# \Rightarrow EEEL\# \Rightarrow \cdots \Rightarrow \underbrace{E \dots E}_{n} L\#$$

En remplaçant chaque E par le terminal `sym`, L par le mot vide, on obtient le mot de taille $n+1$ suivant.

$$\underbrace{\texttt{sym}\,\texttt{sym}\,\dots\,\texttt{sym}}_{n}\,\texttt{\#}$$

Ce même mot peut être obtenu par dérivation à gauche de la manière suivante.

$$\begin{aligned} S &\Rightarrow L\# \Rightarrow EL\# \Rightarrow \texttt{sym}\,L\# \Rightarrow \texttt{sym}\,EL\# \Rightarrow \texttt{sym}\,\texttt{sym}\,L\# \Rightarrow \texttt{sym}\,\texttt{sym}\,EL\# \Rightarrow \dots \\ &\Rightarrow \texttt{sym}\,\texttt{sym}\dots\texttt{sym}\,L\# \Rightarrow \texttt{sym}\,\texttt{sym}\dots\texttt{sym}\,\varepsilon\,\texttt{\#} \Rightarrow \texttt{sym}\,\texttt{sym}\dots\texttt{sym}\,\texttt{\#} \end{aligned}$$

Question 2.

```
exception SyntaxError
type token = Sym | Lpar | Rpar | Eof

(* quelques fonctions de conversions *)
let f_convert sym = match sym with
  | '(' -> Lpar
  | ')' -> Rpar
  | '#' -> Eof
  | _ -> Sym

let finv_convert sym = match sym with
  | Lpar -> '('
  | Rpar -> ')'
  | Eof -> '#'
  | Sym -> 'x'

let str_to_char_list s =
  let lst = s |> String.to_seq |> List.
      of_seq in
  List.map (fun c -> f_convert c) lst

let char_list_to_str lst =
  let lst = List.map (fun c ->
      finv_convert c) lst in
  lst |> List.to_seq |> String.of_seq
```

```
(* fonctions de parsing *)
let rec parseS l = match l with
  | (Sym | Lpar | Eof) :: _ ->
    (match parseL l with
      | Eof :: q -> q
      | _ -> raise SyntaxError
    )
  | [] | Rpar :: _ -> raise SyntaxError
and parseL l = match l with
  | (Sym | Lpar) :: _ -> parseL (parseE
      l)
  | (Rpar | Eof) :: _ -> l
  | [] -> raise SyntaxError
and parseE l = match l with
  | Sym :: q -> q
  | Lpar :: q ->
    (match parseL q with
      | Rpar :: r -> r
      | _ -> raise SyntaxError
    )
    | [] | (Rpar | Eof) :: _ -> raise
       SyntaxError
```

Mise en œuvre sur un exemple.

```
let w = "(a+b)(a-b)#"
let w' = str_to_char_list w
```

```
let s1 = char_list_to_str (parseS w')
let s2 = char_list_to_str (parseL w')
let s3 = char_list_to_str (parseE w')
```

Question 3.

```
let accepts l =
  try parseS l = [] with SyntaxError -> false
```

Question 4. Le symbole L est trivialement nul, car on a la règle de production $L \to \varepsilon$. Les symbole S et E ne sont pas nuls car tout mot dérivé de S contient au moins le terminal `#`, et de même tout mot dérivé de E contient soit le terminal `sym`, soit le terminal `(`.

Question 5. La valeur de $\text{Nul}(X)$ n'évolue que dans le sens `false` vers `true`. Dès lors, le nombre de valeurs `false` ne fait que diminuer. S'il ne change pas, l'algorithme s'arrête. Sinon, il diminue strictement et constitue donc un variant de l'algorithme. En particulier, le nombre d'étapes est borné par le nombre de non terminaux de la grammaire.

Question 6. D'une part, on montre par récurrence sur le nombre d'étapes de l'algorithme que si on obtient $\text{Nul}(X) =$ `true`, alors effectivement $X \Rightarrow^* \varepsilon$. C'est clair initialement car $\text{Nul}(X) =$ `false` pour tout X. Et si $\text{Nul}(X)$ devient

`true`, alors soit $X \to \varepsilon$, soit $X \to X_1X_2 \dots X_p$ avec $X_i \Rightarrow^* \varepsilon$ par hypothèse de récurrence. D'autre part, on montre par récurrence sur le nombre d'étapes de la dérivation $X \Rightarrow^* \varepsilon$ qu'on obtient bien $\text{Nul}(X) =$ `true` par l'algorithme. Si $X \Rightarrow \varepsilon$, c'est qu'il existe une production $X \to \varepsilon$ et on aura bien $\text{Nul}(X) =$ `true` par l'algorithme. Sinon, $X \Rightarrow X_1X_2 \dots X_p$ avec $X_i \Rightarrow^* \varepsilon$ et par hypothèse de récurrence, on aura $\text{Nul}(X_i) =$ `true` par l'algorithme, pour tout i, et donc $\text{Nul}(X) =$ `true`.

Les premiers et les suivants.

Question 7.

$$\text{Premiers}(S) = \{\texttt{sym}, \texttt{(}, \texttt{\#}\} \qquad \text{Suivants}(S) = \{\}$$
$$\text{Premiers}(L) = \{\texttt{sym}, \texttt{(}\} \qquad \text{Suivants}(L) = \{\texttt{)}, \texttt{\#}\}$$
$$\text{Premiers}(E) = \{\texttt{sym}, \texttt{(}\} \qquad \text{Suivants}(E) = \{\texttt{sym}, \texttt{(}, \texttt{)}, \texttt{\#}\}$$

Question 8. On comprend que $\mathcal{G}$ reconnaît les listes par la gauche et $\mathcal{G}'$ par la droite, le reste étant identique. On va montrer la double inclusion des langages.

Montrons tout d'abord que si $L \Rightarrow^* w \in T^* \setminus \{\varepsilon\}$ alors $w = w'u$ avec $L \Rightarrow^* w'$ et $E \Rightarrow^* u$. On prouve ce résultat par récurrence sur la longueur de la dérivation. La dérivation commence nécessairement par $L \Rightarrow EL$ car $w \neq \varepsilon$. Ensuite, elle se poursuit avec $E \Rightarrow^* w_1$ et $L \Rightarrow^* w_2$ et $w = w_1w_2$. Par hypothèse de récurrence, $w_2 = w_3u$ avec $L \Rightarrow^* w_3$ et $E \Rightarrow^* u$. Il vient donc $L \Rightarrow EL \Rightarrow^* w_1w_3$ et on conclut en posant $w = w_1w_3$.

Montrons maintenant que $X \Rightarrow^* w$ implique $X' \Rightarrow^* w$ pour chaque des trois non terminaux. On procède par récurrence sur la longueur de la dérivation $X \Rightarrow^* w$.

- Si $X = S$, la dérivation est $S \Rightarrow L\texttt{\#}$ avec $L \Rightarrow^* w'$ et $w = w'\texttt{\#}$. Par HR, $L' \Rightarrow^* w'$ et donc $S' \Rightarrow L'\texttt{\#} \Rightarrow^* w'\texttt{\#}$.
- Si $X = L$ et si $L \Rightarrow^* \varepsilon$ alors $L' \Rightarrow^* \varepsilon$. Sinon, on peut appliquer le résultat établi ci-dessus et $w = w'u$ avec $L \Rightarrow^* w'$ et $E \Rightarrow^* u$. Par HR, on a $L' \Rightarrow^* w'$ et $E' \Rightarrow^* u$, et donc $L' \Rightarrow L'E' \Rightarrow^* w'u = w$.
- Si $X = E$, on a deux cas. Si $E \Rightarrow^* sym$ alors $E' \Rightarrow \texttt{sym}$ également. Si $E \Rightarrow (L) \Rightarrow^* (w')$ alors par HR $L' \Rightarrow^* w'$ et donc $E' \Rightarrow (L') \Rightarrow^* (w') = w$.

L'autre inclusion se prouve de façon tout à fait similaire (avec un résultat similaire).

Question 9. Pour cette grammaire, on trouve :

$$\text{Premiers}(S') = \{\texttt{sym}, \texttt{(}, \texttt{\#}\} \qquad \text{Suivants}(S') = \{\}$$
$$\text{Premiers}(L') = \{\texttt{sym}, \texttt{(}\} \qquad \text{Suivants}(L') = \{\texttt{sym}, \texttt{(}, \texttt{)}, \texttt{\#}\}$$
$$\text{Premiers}(E') = \{\texttt{sym}, \texttt{(}\} \qquad \text{Suivants}(E') = \{\texttt{sym}, \texttt{(}, \texttt{)}, \texttt{\#}\}$$

et on obtient la table suivante.

	`sym`	`(`	`)`	`#`
S'	$L'\texttt{\#}$	$L'\texttt{\#}$		$L'\texttt{\#}$
L'	$\varepsilon/L'E'$	$\varepsilon/L'E'$	ε	ε
E'	`sym`	(L)		

Dans ce tableau, deux choix apparaîssent dans deux cases. Ce qui ne permet pas l'analyse.