I Miroir

On note $\widetilde{u} = u_n...u_1$ le miroir d'un mot $u = u_1...u_n$ et $\widetilde{L} = \{\widetilde{u} \mid u \in L\}$ le miroir d'un langage L.
Soit L un langage hors-contexte. Montrer que $\widetilde{L}$ est un langage hors-contexte.

II Trouver une grammaire

Montrer que les langages suivants sont non-contextuels :

1. $L_1 = (ab)^*$
2. $L_2 = \{a^n b^p \mid n \geq p\}$.
3. L_3 : représentations des multiples de 3 en base 2.
4. $L_4 = \{a^i b^j c^k \mid i = j + k\}$.

III Trouver le langage engendré

Déterminer les langages engendrés par les grammaires suivantes avec S comme symbole initial, en le prouvant :

1. G_1 :

$$\begin{aligned} S &\to X \mid Y \\ X &\to aX \mid aZ \\ Y &\to Yb \mid Zb \\ Z &\to \varepsilon \mid aZb \end{aligned}$$

2. G_2 :

$$\begin{aligned} S &\to 0A1 \mid \varepsilon \\ A &\to 1S0 \mid \varepsilon \end{aligned}$$

3. G_3 :

$$\begin{aligned} S &\to X \mid XaS \\ X &\to aXbX \mid bXaX \mid \varepsilon \end{aligned}$$

4. G_4 :

$$\begin{aligned} S &\to X \mid Y \\ X &\to Z0X \mid Z0Z \\ Y &\to Z1Y \mid Z1Z \\ Z &\to \varepsilon \mid 1Z0Z \mid 0Z1Z \end{aligned}$$

IV Régulier $\implies$ hors-contexte

Montrer par induction structurelle que tout langage régulier est un langage hors-contexte (ce qui est une preuve alternative passant par un automate, donnée en cours).

V Lemme de l'étoile algébrique, intersection et complémentaire

On admet la version suivante du lemme de l'étoile pour les langages non-contextuels :

Théorème : Lemme d'Ogden

Si L est un langage hors-contexte alors il existe un entier n tel que, pour tout mot $t \in L$ tel que $|t| \geq n$, on peut écrire $t = uvwxy$ avec :

- $|vwx| \leq n$;
- $vx \neq \varepsilon$;
- $\forall i \in \mathbb{N}, uv^i wx^i y \in L$.

Soient $L_1 = \{a^n b^n c^p \mid n, p \in \mathbb{N}\}$ et $L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

1. Montrer que L_1 est un langage hors-contexte.
2. Montrer que L_2 n'est pas un langage hors-contexte.
3. Montrer que l'ensemble des langages hors-contextes n'est pas stable par intersection ni par passage au complémentaire.

VI CCP 2023

On considère la grammaire algébrique G sur l'alphabet $\Sigma = \{a, b\}$ et d'axiome S dont les règles sont : $S \to SaS \mid b$

1. Cette grammaire est-elle ambiguë ? Justifier.
2. Déterminer (sans preuve pour cette question) le langage L engendré par G. Quelle est la plus petite classe de langages à laquelle L appartient ?
3. Prouver que $L = L(G)$.
4. Décrire une grammaire qui engendre L de manière non ambiguë en justifiant de cette non ambiguité.
5. Montrer que tout langage dans la même classe de langages que L peut être engendré par une grammaire algébrique non ambiguë.

VII Forme normale de Chomsky

Une grammaire $G = (\Sigma, V, R, S)$ est en forme normale de Chomsky si toutes ses règles sont de la forme $X \longrightarrow YZ$ (où $Y, Z \in V$), $A \longrightarrow a$ (où $a \in \Sigma$) ou $S \longrightarrow \varepsilon$.
Soit G une grammaire qui n'engendre pas ε. Montrer qu'il existe une grammaire G' en forme normale de Chomsky telle que $L(G') = L(G)$.

VIII Mots de Dyck

Soit $\Sigma = \{a, b\}$. Un mot u sur Σ est un mot de Dyck (ou : mot bien parenthésé) si :

a) u contient autant de a que de b

b) chaque préfixe de u contient au moins autant de a que de b

On note D l'ensemble des mots de Dyck.

1. Montrer que D n'est pas un langage régulier.
2. Montrer que tout mot de Dyck non-vide se décompose de manière unique sous la forme $aubv$, où u et v sont des mots de Dyck.
3. Soit G la grammaire donnée par $S \to SS \mid aSb \mid \varepsilon$. Montrer que $L(G) = D$.
4. Montrer que G est ambigüe.
5. Donner une grammaire non-ambigüe engendrant D.
6. Donner une bijection entre les mots de Dyck et les arbres binaires stricts (tels que tout nœud possède 0 ou 2 fils).
7. Soit C_n le nombre de mots de Dyck de longueur $2n$. Trouver une équation de récurrence sur C_n.
8. Après avoir fait le cours de mathématiques sur les séries entières, montrer que $C_n = \dfrac{1}{n+1}\dbinom{2n}{n}$.
9. Dans cette question, on peut utiliser des parenthèses différentes (par exemple, `{}` et `[]`). Décrire un algorithme en complexité linéaire pour savoir si un mot est bien parenthésé.
10. Décrire un algorithme en complexité linéaire pour trouver la longueur du plus long facteur bien parenthésé d'un mot sur Σ.

On pourra résoudre les deux dernières questions sur LeetCode : Valid Parentheses et Longest Valid Parentheses.