

1 Transformations de grammaire

1. Soit l'alphabet $X = \{+, =, a\}$. Donner une grammaire algébrique pour le langage L dont chaque mot représente une addition correcte de deux suites de caractères a . Par exemple L contient le mot $aa + aaaa = aaaaaa$.

Correction On constate que le langage demandé est de la forme $a^n + a^m = a^m a^n$.

D'où la grammaire :

$$\begin{aligned} S &\rightarrow aSa \\ S &\rightarrow a + Ma \\ M &\rightarrow aMa \\ M &\rightarrow a = a \end{aligned}$$

2. Soit le langage L_1 sur le vocabulaire $V = \{l', \text{homme}, \text{ours}, \text{qui}, a, \text{vu}\}$ formé de l'ensemble des phrases finies de la forme *l'homme qui a vu l'ours*, *l'homme qui a vu l'homme qui a vu l'ours*, *l'homme qui a vu l'ours*, *l'homme qui a vu l'homme qui a vu l'homme qui a vu ... qui a vu l'ours*.

- (a) Grammaire algébrique (*context-free*) engendrant L_1 :

$$\begin{aligned} S &\rightarrow \text{l'homme qui a vu } S \\ S &\rightarrow \text{l'homme qui a vu l'ours} \end{aligned}$$

- (b) Grammaire régulière engendrant L_1 .
- $$\begin{aligned} S &\rightarrow l' A \\ A &\rightarrow \text{homme } B \\ B &\rightarrow \text{qui } C \\ C &\rightarrow a D \\ D &\rightarrow \text{vu } E \\ &\quad | \text{vu } S \\ E &\rightarrow l' F \\ F &\rightarrow \text{ours} \end{aligned}$$

Soit L_2 le langage engendré par la grammaire \mathcal{G}_2 :

$$\begin{aligned} S &\rightarrow \text{NP Rel} \\ \text{NP} &\rightarrow \text{l'homme} \\ &\quad | \text{l'ours} \\ \text{Rel} &\rightarrow \text{qui a vu NP Rel} \\ &\quad | \text{qui a vu l'ours} \end{aligned}$$

- (c) Grammaire \mathcal{G}_2 sous forme normale de Chomsky :

$$\begin{aligned} S &\rightarrow \text{NP Rel} \\ \text{NP} &\rightarrow L H \mid L O \\ L &\rightarrow l' \\ H &\rightarrow \text{homme} \\ O &\rightarrow \text{ours} \\ \text{Rel} &\rightarrow Q a_vu_NP_Rel \mid Q a_vu_l'ours \\ Q &\rightarrow \text{qui} \\ a_vu_NP_Rel &\rightarrow A \text{ vu_NP_Rel} \\ A &\rightarrow a \\ \text{vu_NP_Rel} &\rightarrow V \text{ NP_Rel} \\ V &\rightarrow \text{vu} \\ \text{NP_Rel} &\rightarrow \text{NP Rel} \\ a_vu_l'ours &\rightarrow A \text{ vu_l'ours} \\ \text{vu_l'ours} &\rightarrow V \text{ l'ours} \\ l'ours &\rightarrow L O \end{aligned}$$

- (d) Grammaire \mathcal{G}_3 engendrant le langage L_3 , sur-ensemble de L_2 mais dans lequel les symboles *ours* et *homme* sont **strictement** interchangeables :
- Changer la règle : Rel \rightarrow qui a vu l'ours
 Par la règle : Rel \rightarrow qui a vu NP
- (e) Différences entre les langages L_1 et L_2 : L_2 doit finir par *l'ours*, mais *homme* et *ours* peuvent apparaître librement auparavant.
- (f) Grammaire algébrique du langage $L_2 \setminus L_1$:
- Il s'agit du langage où les chaînes finissent par *l'ours*, et où auparavant on a au moins une (autre) occurrence de *l'ours*. Avant d'avoir consommé cette occurrence 'intérieure' (non finale) de *l'ours*, on est comme en S. Après consommation, on peut soit terminer, soit consommer indifféremment *l'ours* ou *l'homme*.
- S \rightarrow l'homme qui a vu S
 S \rightarrow l'ours qui a vu A
 A \rightarrow l'ours
 A \rightarrow l'homme qui a vu A
 A \rightarrow l'ours qui a vu A

3. Mettre sous forme normale de Chomsky la grammaire définie par les règles de production suivantes
- $$S \rightarrow AB \mid aS \mid a$$
- $$A \rightarrow Ab \mid \varepsilon$$
- $$B \rightarrow AS$$

Correction Il faut commencer par nettoyer la grammaire. Suppression des ε -transitions :

$$S \rightarrow AB \mid B \mid aS \mid a$$

$$A \rightarrow Ab \mid b$$

$$B \rightarrow AS$$

Puis suppression des cycles éventuels : pour cela on supprime les productions singulières :

$$S \rightarrow AB \mid AS \mid aS \mid a$$

$$A \rightarrow Ab \mid b$$

$$B \rightarrow AS$$

Et enfin la forme quadratique :

$$S \rightarrow AB \mid AS \mid XS \mid a$$

$$X \rightarrow a$$

$$A \rightarrow AY \mid b$$

$$Y \rightarrow b$$

$$B \rightarrow AS$$

4. (a) Montrer que la grammaire suivante est ambiguë :
- $$S \rightarrow TU; T \rightarrow ST \mid a; U \rightarrow US \mid b$$
- On prouve facilement que *abab* est possible via 2 dérivations.
- (b) La grammaire suivante est-elle ambiguë ? $S \rightarrow aSSb \mid ab$
- Intuitivement, l'ambiguïté si elle existe proviendrait de l'utilisation à un moment des deux règles différentes pour les deux symboles S. C'est-à-dire dérivation 1 : $aSSb \rightarrow a(aSSb)(ab)b$ versus dérivation 2 $aSSb \rightarrow a(ab)(aSSb)b$. Or on voit que les chaînes engendrées par les deux versions sont différentes : les chaînes commencent par au moins 3 *a* pour la dérivation 1, alors que les chaînes commencent par *aab* pour la dérivation 2.

5. **Grammaires propres :**

- (a) Caractériser le langage reconnu par la grammaire suivante $S \rightarrow cB \mid cS$
 $B \rightarrow aBb \mid \varepsilon$

Définir l'algorithme qui rend une grammaire ε -libre et l'appliquer à la grammaire ci-dessus.

Correction Le langage est $c * a^n b^n$ avec $n \geq 0$

Algorithme rendant une grammaire ε -libre : On commence par calculer l'ensemble des non terminaux dérivant epsilon directement ou indirectement :

```

calculer_epsilon_plus(grammaire)
{
  ( ## initialiser epsilon_plus : au départ l'ensemble vide)
  epsilon_plus := ∅
  epsilon_plus_grossit := vrai
  ( ## tant que epsilon_plus grossit)
  tant que epsilon_plus_grossit est vrai
  {
    epsilon_plus_grossit := faux
    pour chaque règle de grammaire
    {
      si la partie gauche n'est pas déjà dans epsilon_plus
      {
        si tous les symboles de la partie droite ∈ epsilon_plus
        (y compris partie droite vide)
        {
          Ajouter la partie gauche à epsilon_plus
          epsilon_plus_grossit := faux
        }
      }
    }
  }
  return epsilon_plus
}

```

```

supprimer_epsilon_productions(grammaire)
{
  epsilon_plus := calculer_epsilon_plus(grammaire)
  new_rules := ∅
  pour chaque old_rule de grammaire
  {
    On remplace old_rule par un ensemble de nouvelles règles sigma_rules,
    construit de la manière suivante :
      Pour chaque symbole x de la partie droite de old_rule,
      tel que  $x \in \textit{epsilon\_plus}$ , on considère l'alternative :
        garder x dans la partie droite, ou bien l'enlever
      sigma_rules est constitué de toute la combinatoire de ces alternatives
      moins la chaîne vide.
      C'est à dire que si  $\varepsilon$  apparaît dans la combinatoire,
      on ne le retient pas dans sigma_rules
      D'où l'algorithme :
      sigma_rules := { $\varepsilon$ }
      Pour chaque symbole x de la partie droite de old_rule
      {
        Pour chaque chaîne de sigma_rules
        {
          si  $x \in \textit{epsilon\_plus}$ 
          remplacer dans sigma_rules,
          chaîne par le couple chaîne et chaîne+x
          sinon remplacer dans sigma_rules, chaîne par chaîne+x
        }
      }
    }
  }
  Ajouter à new_rules les règles sigma_rules, sauf celles de la forme  $X \rightarrow \varepsilon$ 
  Si  $\varepsilon$  appartient au langage engendré par grammaire
  {
    Créer un nouvel axiome S' distinct de S (avec S = l'axiome de grammaire)
    Et Ajouter à new_rules les règles  $S' \rightarrow \varepsilon | S$ 
    return nouvelle grammaire (axiome = S', règles = new_rules)
  }
  Sinon
  {
    return nouvelle grammaire (axiome = S, règles = new_rules)
  }
}
    
```

Pour la grammaire citée, cela donne : $\textit{epsilon_plus} = B$ et la grammaire devient : $S \rightarrow cB \mid c \mid cS$
 $B \rightarrow aBb \mid ab$