# Transformation de Grammaires : Chomsky et Greibach et Génération de Mots

César Charbey 222027321, Ayoub Ourimi 22206283

January 18, 2025

## Sommaire

Ce compte-rendu décrit :

- Structure de Données
- Fonctions Utilitaires
- Transformation en Forme Normale de Chomsky
- Transformation en Forme Normale de Greibach
- Génération de Mots depuis une Grammaire

# 1 Structure de Données

Nous utilisons un **dictionnaire Python** pour représenter la grammaire. Chaque clé est un *non-terminal*, et sa valeur est une liste de *séquences* (listes de symboles).

- Un **terminal** est représenté par une lettre minuscule.
- Un **non-terminal** est représenté par une lettre majuscule (optionnellement suivie de chiffres), par exemple `S0`, `A1`, etc.
- L'**epsilon** est stocké comme la liste vide `[]`, et réécrit comme `E` en sortie.

## 1.1 Exemple de structure

Si l'on a la grammaire suivante (en notation informelle) :

$$S \to A\,b\,B \mid C \quad , \quad A \to a \mid \varepsilon \quad , \quad B \to A\,A \mid A\,C \quad , \quad C \to b \mid c$$

alors la structure Python correspondante est :

```
{
  "S": [ ["A", "b", "B"], ["C"] ],
  "A": [ ["a"], [] ],        // [] represente epsilon
  "B": [ ["A", "A"], ["A", "C"] ],
  "C": [ ["b"], ["c"] ]
}
```

# 2 Fonctions Utilitaires

## 2.1 Lecture et Écriture (`lire`, `ecrire`)

`lire(fichier)` :

- Lit un fichier texte ligne par ligne, de la forme `A0 -> A0a | E`.
- Le premier non-terminal rencontré devient l'axiome.
- Remplace `E` par la liste vide `[]`.
- Sépare les symboles en terminaux/minuscules et non-terminaux/majuscules (regex).

`ecrire(fichier)` :

- Parcourt toutes les règles.
- Convertit la liste vide `[]` en `E`.
- Concatène les symboles pour obtenir une sortie lisible.

## 2.2 `afficher_regles(message)`

Affiche la grammaire à des fins de **débogage**, en remplaçant `[]` par `E`, etc.

## 2.3 `GenerateurNonTerminaux`

Afin d'*éviter* de réutiliser un nom de variable déjà existant, on initialise ce générateur avec l'ensemble des non-terminaux déjà présents dans la grammaire, de façon à créer de nouveaux noms inédits (ex. `"A0"`, puis `"A1"`, `"B0"`, etc.) sans soucis.

# 3 Transformation en Forme Normale de Chomsky

La conversion se fait en plusieurs étapes, dans l'ordre standard (cf. **Wikipédia**) :

1. **Réduction** : suppression des symboles non-terminaux inutiles (inaccessibles ou non coaccessibles).
2. **START** : introduction éventuelle d'un nouvel axiome `S0 -> S`.
3. **TERM** : remplacement des terminaux dans les règles de longueur $\geq 2$ par des variables intermédiaires.
4. **BIN** : binarisation des règles trop longues.
5. **DEL** : suppression des règles $\varepsilon$ (sauf éventuellement l'axiome).
6. **UNIT** : suppression des règles unitaires $A- > B$ via une fermeture transitive.

Enfin, un **nettoyage** supprime les doublons.

## 3.1 Exemple : Grammaire cyclique en forme normale de chomsky

Il arrive qu'une grammaire en forme normale de Chomsky contienne des règles cycliques (ex. `A0 -> A0 A1`), pouvant générer des expansions infinies si l'on essaie d'énumérer tous les mots naïvement. C'est pourquoi lors de la génération, on introduit une **limite** sur la taille totale de la séquence.

## 3.2 Cas d'échec en forme normale de Greibach

Noter qu'en forme normale de Chomsky on peut gérer les grammaires purement récursives à gauche + epsilon, mais en **forme normale de Greibach stricte**, elles peuvent échouer (exemple : `A0 -> A0a | E`).

# 4 Transformation en Forme Normale de Greibach

**Objectif :** Chaque règle est de la forme

$$A \;\to\; a\,X_1 \cdots X_k \quad (\text{pour } k \geq 0, \text{ et } a \text{ un terminal}),$$

ou $\varepsilon$ si $A$ est l'axiome.

## 4.1 Phases Principales

1. **Suppression des $\varepsilon$-règles** (sauf axiome).
2. **Suppression des règles unitaires**.
3. **Mise en ordre** $(A_1, A_2, \ldots)$ :
   - *Substitution* : Pour $i = 1..n$, on remplace $A_j$ (avec $j < i$) s'il apparaît en tête dans les règles de $A_i$.
   - *Élimination récursivité gauche* : si $A_i{-}{>}\, A_i\alpha|\ldots$, on crée un nouveau $A'_i$.
4. **Placement d'un terminal en tête** : tant qu'une règle commence par un NT, on substitue.

## 4.2 Échec si non-axiome -> E

La **Forme normale de Greibach stricte** interdit toute production $\varepsilon$ hors axiome. Par exemple, la grammaire $A0{-}{>}\, A0\,a \mid \varepsilon$ échoue car on doit créer un auxiliaire $A0'{-}{>}\,\varepsilon$, et $A0'$ n'est pas l'axiome.

## 4.3 Validation

Une fois la forme Greibach obtenue (ou l'échec signalé), on vérifie :

$$\forall\, A \neq S_0, \quad A \to a\ldots \quad \text{et} \quad S_0 \to \varepsilon \text{ éventuellement.}$$

# 5 Génération de Mots depuis la Grammaire

## 5.1 But et Principes

Le module `Generer.py` permet d'**énumérer tous les mots** (en terminaux) que la grammaire peut produire, d'une **longueur maximale** donnée. On tient compte des symboles `E` (epsilon) qui s'effacent.

## 5.2 Lecture de la Grammaire

- Identique au module principal : on lit `nonTerminal -> Production | ....`
- On stocke dans un dictionnaire `regles` : {NT : [expansion1, expansion2, ...]}.
- `E` est converti en epsilon effectif lors de la génération (i.e. on supprime le symbole).

## 5.3 Algorithme d'Énumération

On utilise une **pile** ou une **file** (DFS ou BFS) de séquences.

1. **Démarre** avec une séquence contenant juste l'axiome ([Axiome]).
2. **À chaque étape** :
   - Calculer la longueur en terminaux de la séquence courante (ignorant `E`).
   - Si la longueur dépasse `longueur_max`, on **coupe**.
   - Si la séquence ne contient plus aucun non-terminal, on la convertit en mot final (enlevant `E`) et on l'ajoute au résultat.
   - Sinon, on **remplace** le premier non-terminal par chacune de ses expansions possibles, puis on empile/enfile ces nouvelles séquences.
3. **Déduplication** : on garde un ensemble (`deja_vu`) des séquences déjà développées pour éviter les boucles ou l'explosion combinatoire.

## 5.4 Limitation des Boucles Inifnies

Certaines formes normales de Chomsky comportent des règles cycliques (`A->A A`), engendrant des dérivations infinies. Pour éviter cela :

- On **limite** la taille totale de la séquence (en plus de la longueur des terminaux).
- Par exemple : `if len(sequence) > 2 * longueur_max : continue`.

## 5.5 Exemple de Génération

- Grammaire : $A0 -> A0\,a \mid \varepsilon$.
- `longueur_max` $= 3$.

Le programme énumère : $\varepsilon$, $a$, $aa$, $aaa$. Au-delà de 3 `a`, on coupe.