UVSQ université PARIS-SACLAY

UFR des Sciences

CAMPUS DE VERSAILLES

Projet – Transformation de grammaires

Rapport

Les étudiants :

BOUAROUA Slimane 22402112

ASSIS Hugo 22200574

Sous la supervision de :

Mr Franck Quessette

Année Universitaire : 2024/2025

Table des Matières

Introduction

Ce projet porte sur la transformation de grammaires formelles, un domaine fondamental en théorie des langages et des automates. L'objectif principal est d'appliquer des techniques de transformation pour convertir des grammaires en formes normales, notamment la forme normale de Chomsky (CNF) et la forme normale de Greibach (GNF).

Dans ce projet, nous avons implémenté :

- ➢ Une méthode systématique pour convertir les grammaires en forme normale de Chomsky (CNF), où chaque règle a une structure standardisée.
- ➢ Une approche pour transformer les grammaires en forme normale de Greibach (GNF), permettant d'obtenir des règles utiles pour les analyseurs descendants récursifs.

Structure de Données

Pour implémenter efficacement les transformations des grammaires en forme normale de Chomsky (CNF) et forme normale de Greibach (GNF), nous avons conçu une structure de données adaptée qui permet de manipuler les productions et les non-terminaux de manière flexible et performante.

Nous avons choisi la structure de données Dictionnaire (HashMap) pour représenter les règles de la grammaire, ça ressemble un peu a Json en JavaScript, parce que la grammaire contient de partie : partie droite qui contient un seul non-terminal dans notre cas et la partie gauche qui contient ensemble des règles de production, ensuite on va les stocker avec la notion clé-valeur :

- Chaque non-terminal est une clé.
- La valeur associée est une liste des productions possibles pour ce non-terminal et son type sera une structure de données liste.

Exemples en python :

```
grammaire = {
    "S": ["AB", "a"],
    "A": ["aA", "B"],
    "B": ["b"]
}
```

Nous l’avons utilisé avec plusieurs raisons :

- Accès rapide aux règles d'un non-terminal.
- Facilité d'ajout, de suppression ou de modification des productions.

Algorithmes Implémentés

Chaque algorithme que nous avons programmé pour transformer une grammaire en forme normale de Chomsky (FNC) ou en forme normale de Greibach (FNG) est structuré en cinq étapes. Ces étapes sont interdépendantes, chaque étape s'appuyant sur les résultats obtenus à l'étape précédente pour garantir une transformation correcte et cohérente de la grammaire, pour cette raison, nous avons conçu un algorithme (fonction) dédié pour chaque étape de la transformation.

Forme normale de Chomsky (FNC)

1. Retirer l'axiome des membres droits des règles ;

Avant	Après
S -> aSa | bSb	S -> B B -> aBa | bBb

L'algorithme prend en entrée une grammaire et son axiome, génère un nouveau non-terminal, et modifie les règles pour remplacer les occurrences récursives de l'axiome par ce nouveau non-terminal, en sortie, il produit une grammaire transformée où l'axiome initial est remplacé par une règle pointant vers le nouveau non-terminal, simplifiant ainsi la gestion des règles.

2. Supprimer les terminaux dans le membre droit des règles de longueur au moins deux ;

Avant	Après
S -> aBb | a | b| bb | aB	S -> A1 B B1 | a | b | B1B1 | aB1 A1 -> a B1-> b B -> c

L'algorithme prend en entrée une grammaire et remplace tous les terminaux apparaissant dans des règles de longueur ≥2 par de nouveaux non-terminaux, en générant des règles intermédiaires associant chaque nouveau non-terminal à son terminal, produisant en sortie une grammaire transformée où toutes les règles de longueur ≥2 contiennent uniquement des non-terminaux.

3. Supprimer les règles avec plus de deux non-terminaux ;

Avant	Après
S -> UVC	S-> US1 S1 -> VC

L'algorithme prend en entrée une grammaire et transforme les règles contenant plus de deux symboles en les décomposant en règles binaires (contenant au maximum deux symboles), en générant de nouveaux non-terminaux pour chaque groupe de deux symboles, et produit en sortie une grammaire où toutes les règles sont binaires, conformément aux exigences de la forme normale de Chomsky (CNF).

4. Supprimer les règles $X \rightarrow \varepsilon$ sauf si X est l'axiome ;

Avant	Après
S -> AbB | C B -> AA | AC C -> b | c A -> a | ε	S -> AbB | Ab | C | b | bB B -> A | AA | AC | C C -> b | c A -> a

L'algorithme prend en entrée une grammaire et son axiome, identifie les non-terminaux capables de produire ϵ (epsilon), et génère toutes les combinaisons possibles des règles en supprimant les symboles ϵ productifs de manière exhaustive. Pour chaque règle, il produit toutes les variantes possibles où chaque occurrence d'un non-terminal ϵ productif peut être soit supprimée, soit conservée, garantissant ainsi que toutes les variantes valides des règles sont explorées et ajoutées à la grammaire transformée. En sortie, la grammaire ne contient plus de ϵ productions, sauf éventuellement pour l'axiome, si cela est nécessaire.

5. Supprimer les règles unité $X \rightarrow Y$;

Avant	Après
A -> aA | B B -> bB | C C -> cC	A -> aA | bB | cC B -> bB | cC C -> cC

L'algorithme prend en entrée une grammaire et élimine les **règles unitaires** (règles où un non-terminal pointe directement vers un autre non-terminal, par exemple A→B). En sortie, il produit une grammaire modifiée où chaque non-terminal contient uniquement des règles finales (produisant des terminaux ou des séquences de non-terminaux) sans aucune règle unitaire.

Forme normale de Greibach (FNG)

1. Retirer l'axiome des membres droits des règles ; Elle a été expliquée dans la FNC.
2. Supprimer les règles $X \rightarrow \varepsilon$ sauf si X est l'axiome ; Elle a été expliquée dans la FNC.
3. Supprimer les règles unité $X \rightarrow Y$; Elle a été expliquée dans la FNC.
4. Supprimer les non-terminaux en tête des règles ;

Avant	Après
A -> Az | x	A - > x B B - > z B | ε

L'algorithme transforme une grammaire en s'assurant que chaque règle commence par un terminal. Si une règle commence par un non-terminal, les productions de ce non-terminal sont substituées récursivement à la place. En sortie, il produit une grammaire modifiée où toutes les règles commencent par un terminal ou sont des règles vides (ϵ).

5. Supprimer les symboles terminaux qui ne sont pas en tête des règles.

Avant	Apres
S -> aAa | ab A -> c	S -> aAB | aC A -> c B -> a C -> b

L'algorithme prend en entrée une grammaire et remplace tous les terminaux qui ne sont pas en tête des règles (c'est-à-dire après le premier symbole) par des non-terminaux nouvellement générés. Les règles de longueur 1 contenant un terminal ou les règles vides (ϵ) restent inchangées. En sortie, il produit une grammaire où chaque terminal situé après la tête est remplacé par un non-terminal, avec des règles supplémentaires associant ces nouveaux non-terminaux aux terminaux correspondants.

Conclusion

Ce projet nous a permis d'explorer en profondeur les concepts fondamentaux liés aux grammaires formelles et leurs transformations en formes normales, notamment la forme normale de Chomsky (FNC) et la forme normale de Greibach (FNG). À travers l'implémentation de plusieurs algorithmes, nous avons développé une approche structurée et modulaire pour traiter les étapes clés, telles que l'élimination des ϵ productions, des règles unitaires, et des terminaux ou non-terminaux non conformes.