Rapport de projet grammaires transformation

Yang Tianyi
22401398

Janvier 2025

Contents

1 Mode d'emploi du programme **2**

2 Structure de données pour la grammaire **2**

3 Algorithme de transformation de grammaire **3**

1 Mode d'emploi du programme

1.1 Dans `cfg.general`, stocker les grammaires algébriques à transformer

1.2 Utilisation de `make`:

- `make` ou `make run` exécute le programme `grammaire` pour générer les fichiers `alg.chomsky` et `alg.greibach`, qui sont respectivement les grammaires transformées en forme de Chomsky et Greibach.
- `make diff` utilise le programme `generer`, lit les fichiers `alg.chomsky` et `alg.greibach`, génère les fichiers `test_5_chomsky.res` et `test_5_greibach.res` qui ont une longueur de mot maximale de 5, et compare les deux fichiers `.res` pour vérifier s'il y a des différences.

1.3 Sans utiliser `make`:

- `python3 grammaire.py` : Génère les fichiers `alg.chomsky` et `alg.greibach`.
- `python3 generer.py alg.chomsky n` ou `python3 generer.py alg.greibach n` : Où `n` est la longueur maximale des mots générés par la grammaire.

2 Structure de données pour la grammaire

2.1 Grammaires algébriques

- Lettres majuscules A, B, S, X, ..., sauf E, les symboles non-terminaux (variables).
- Lettres minuscules a, b, ..., les terminaux.
- Lettres grecques $\alpha, \beta, \ldots$, les mots constitués de terminaux et de non-terminaux.
- S est toujours l'axiome.
- E représente la chaîne vide.
- Les règles ont la forme membre_gauche $\rightarrow$ membre_droit, où membre_gauche doit être un seul non-terminal.
- Deux formes de règles sont supportées : $S \rightarrow A$, $S \rightarrow B$ ou $S \rightarrow A \mid B$. Les deux formes produisent les mêmes résultats.

2.2 Forme normale de Chomsky

- Tous les non-terminaux sont formés par une combinaison de lettres majuscules et de chiffres, sauf E. Par exemple : A1, B1, ..., A9, B9. Afin d'éviter la confusion entre 0 et O, ainsi que pour la lisibilité de la grammaire générée, tous les non-terminaux entre A0 et Z0 sont abrégés sans le 0, et représentés simplement par A, B, ..., Z. Les non-terminaux commencent par les lettres majuscules A, B, ..., Z. Si cela ne suffit pas, des combinaisons de lettres et de chiffres sont utilisées (voir la fonction _generate_new_non_terminal).
- L'ensemble des terminaux est constitué des 26 lettres minuscules.
- S est toujours l'axiome.
- Les règles générées ont la forme membre_gauche $\rightarrow$ membre_droit, où :
  - membre_gauche est un seul non-terminal
  - membre_droit est soit deux non-terminaux, soit un seul terminal.

2.3 Forme normale de Greibach

- Les non-terminaux, terminaux et S sont les mêmes que ceux de la forme normale de Chomsky.
- Les règles générées ont la forme membre_gauche → membre_droit, où :
  - membre_gauche est un seul non-terminal.
  - membre_droit commence par un terminal, suivi par zéro ou plusieurs non-terminaux.

3 Algorithme de transformation de grammaire

3.1 Algorithme de transformation des grammaires algébriques en Forme normale de Chomsky[1]

3.1.1 Élimination des règles ϵ (Règles Epsilon)

Objectif : Supprimer les règles générant la chaîne vide ϵ, sans modifier le langage.

1. Identifier les non-terminaux pouvant produire ϵ (appelés non-terminaux *nullable*). Cela se fait par une définition récursive :
   - Si $A \to \epsilon$, alors A est *nullable*.
   - Si $A \to X_1 X_2 \dots X_n$, et si tous les X_i sont *nullable*, alors A est *nullable*.
2. Supprimer toutes les règles ϵ :
   - Parcourir chaque production $A \to \alpha$ et générer toutes les combinaisons possibles en remplaçant ou supprimant les symboles *nullable*.
   - Supprimer les productions $A \to \epsilon$.

3.1.2 Élimination des règles unitaires (Unit Rules)

Objectif : Supprimer les règles de la forme $A \to B$ (où le côté droit est un unique non-terminal).

1. Pour chaque règle unitaire $A \to B$, ajouter les productions de B à l'ensemble des productions de A.
2. Répéter jusqu'à ce qu'il n'y ait plus de règles unitaires.

3.1.3 Division des règles dont le côté droit a une longueur supérieure à 2

Objectif : Transformer toutes les règles dont le côté droit a une longueur supérieure à 2 (par exemple, $A \to BCD$) en une série de règles de longueur 2.

1. Pour une règle $A \to BCD$, introduire un nouveau non-terminal X et diviser la règle en :
   - $A \to BX$,
   - $X \to CD$.
2. Répéter le processus récursivement jusqu'à ce que toutes les règles aient un côté droit de longueur ≤ 2.

[1] Le programme n'utilise pas l'étape START standard (par exemple, introduire $S_0 \to S$) pour garantir que le symbole de départ original S reste conforme à la forme normale de Chomsky, car pendant la conversion (comme l'élimination des règles ϵ et des règles unitaires), le symbole de départ S est progressivement remplacé par les règles spécifiques d'autres non-terminaux. Cette méthode remplit implicitement le rôle de l'étape START sans ajouter explicitement un nouveau symbole de départ.

3.1.4 Extraction des terminaux dans des règles séparées

Objectif : Supprimer les règles où des terminaux et des non-terminaux sont mélangés dans le côté droit. Par exemple, $A \to aB$ ou $A \to Ba$.

1. Si une règle contient un terminal (et que le côté droit a une longueur > 1), introduire un nouveau non-terminal pour ce terminal. Par exemple :
   - $A \to aB$ devient $A \to XB$, avec une nouvelle règle $X \to a$.
   - $A \to Ba$ devient $A \to BX$, avec une nouvelle règle $X \to a$.

3.2 Algorithme de transformation des grammaires algébriques en Forme normale de Greibach

3.2.1 Élimination des règles ϵ et des règles unitaires

Objectif : Supprimer les règles qui génèrent la chaîne vide ϵ ainsi que les productions unitaires.

1. Cette étape réutilise deux étapes de la transformation en forme normale de Chomsky :
   - Élimination des règles ϵ,
   - Élimination des règles unitaires.

3.2.2 Élimination de la récursivité à gauche

Objectif : Éliminer toutes les récursivités à gauche (directes ou indirectes), de sorte que le côté droit des règles de chaque non-terminal ne commence pas par lui-même.

1. Pour chaque non-terminal A, diviser ses règles en deux catégories :
   - **Règles α :** Les règles récursives directes, de la forme $A \to A\alpha$,
   - **Règles β :** Les règles non récursives, de la forme $A \to \beta$.
2. Introduire un nouveau non-terminal A' pour gérer la récursivité :
   - Remplacer les règles de A par $A \to \beta A'$,
   - Définir les règles de A' : $A' \to \alpha A' \mid \epsilon$.
3. Répéter ces étapes jusqu'à ce que toutes les récursivités soient éliminées.

3.2.3 S'assurer que toutes les règles commencent par un terminal

Objectif : Adapter les règles pour que chaque règle commence par un terminal dans le côté droit.

1. Si une règle commence par un non-terminal (par exemple, $A \to BC$ ou $A \to BA$), remplacer le premier non-terminal B par ses productions.
2. Effectuer cette substitution récursivement jusqu'à ce que toutes les règles commencent par un terminal.

3.2.4 Suppression des non-terminaux inutiles

Objectif : Nettoyer la grammaire en supprimant les non-terminaux inutilisés ou les règles superflues.

1. Partir du axiome et marquer récursivement tous les non-terminaux accessibles,
2. Supprimer les non-terminaux non marqués ainsi que les règles associées.

References

[1] Moore, Robert C. (2000). *Removing Left Recursion from Context-Free Grammars*. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. URL: https://aclanthology.org/A00-2033.

[2] Dol Aher, Sunita and Halkude, Shraddha (2016). *JEET CRC T24S Modified TPS Activity for Mathematical Courses to Improve Students' Fundamental Knowledge*. URL: https://www.researchgate.net/figure/TPS-activity-for-converting-CFG-to-CNF-C-Improving-TPS-to-T24S_fig2_306082567