Théorie des langages

Projet Transformation de grammaires

Etudiantes :
Daria KURGANSKAYA
22201683
Maryatou KANTE
22201575

Professeur :
Franck Quessette

January 19, 2025

Contents

1 Objectifs du projet

L'objectif est de charger une grammaire algébrique depuis un fichier, de la convertir en formes normales de Greibach et de Chomsky, puis, pour chacune des formes normales obtenues, de générer tous les mots dont la longueur est inférieure à une limite spécifiée.

Les forme normales sont une représentation standard des grammaires hors-contexte ou algébriques en théorie des langages. es formes sont langement utilisées car simplifient la structure des grammaires, ce qui facilite l'analyse syntaxique et certaines opérations théoriques en rendant les algorhithmes plus explicites. C'est le cas de l'angorithme de la génération de mots de longeur donnée à partir d'une grammaire.

2 Notations

NonTer - non terminale
Ter - terminale
Val - terminale ou non terminale du membre droit
MbrDroit - membre droit
MbrGauche - membre gauche

3 Choix de la structure de données

de réaliser notre projet de transformation de grammaires en Python, nous avons opté pour la structure de donnés suivante : **les dictionnaires** dont:
- **les clés** sont des membres gauches des règles
- **les valeurs** sont la liste des membres droits associés au membre gauche donné. Chaque membre droit est aussi une liste dont les valeurs sont des ternimaux ou non terminaux qui composent la règle. De cette manière des valeurs du dictionnaire sont des liste des listes.

grammaire = MbrGauche (NonTer) : [[Val1 du MbrDroit1, Val2 du MbrDroit1], [Val1 du MbrDroit2]]

Afin d'eviter les confusions l'axiome est stokée dans une variable à part.
Cette structure de données confère plusieurs avantages qui facilitent grandement le traitement des règles de grammaires:

1. Accès rapide aux éléments
   Il est très simple de rechercher une règle particulière. Grâce au système clé/valeur la recherche d'une règle particulière est beaucoup plus efficace que dans des structures de données privilégiants des listes. En plus tous les membres droits sont groupés par membres gauches associés.

2. Les règles sont facilement organisables et modifiables.
   La modification, la suppression ou l'ajout de membres droits associés est très simple grâce au système de liste. Il est plus facile d'alterer une liste qui est une valeur du dictionnaire que la valeur elle-même. Enfin, il est aisé d'ajouter de nouveaux couples clé/valeur si nécessaire.

3. Lisibilité
   Cette structure de données étant très simple, elle est très lisible et donc facilement compréhensible. Étant donné qu'une grammaire peut contenir de nombreuses règles, il est crucial de pouvoir les distinguer clairement afin d'éviter les erreurs. Les dictionnaires n'ont pas ce problème grâce à leur organisation claire et structurée. Des règles ajoutées plus tardivement se trouveront en fin du dictionnaire.

4 Les algorithmes

4.1 Lecture/Ecriture

1. Lecture
   Le fichier lu en entrée doit obligatoirement avoir une extention *.general, .chomsky* ou *.greibach*. D'après l'énoncée, chaque ligne du fichier comprend une règle de la grammaire sous la forme : *MbrDroit : MbrGauche* .
   Le fichier est lu ligne par ligne. Grâce aux fonctions *split* et *strip* les lignes sont ensuite décomposées en membre gauche et membre droit et les espaces entre les symboles sont enlevés. Chaque membre droit est ensuite décomposé en liste de non terminaux et terminaux avant d'être ajouté au dictionnaire des regles. L'axiome est le membre droit de la première ligne.

   Tous les carateres du membre droit sont écrits sans espace entre eux. Il fallait donc convertir cette ligne de caractères en liste de valeurs terminaux ou pas. Afin de convertir l'information du fichier en format de données choisit, plusieurs méthodes on été suggérées:
   - A l'aide d'un programme écrit en lex transformer le fichier text en fichier json et ensuite lire le json obtenu par la library json de python. - Lire le fichier et convetir les membres droits en liste à l'aide de la librarie regex de python.
   - Lire le fichier et convetir les membres droits en liste grâce à la fonction *est_non_terminal()*.
   A chaque fois qu'une lettre minuscule est lue, elle est ajoutée au liste des caractères du membre droit, si "E" est lu, l'épsilon est ajoutée, sinon si majuscule est lue, cette lettre et le symbole numérique la suivant sont ajoutés.

   On a choisi d'implémenter la dernière méthode car elle semblait être la plus performante.

2. Ecriture
   Les règles normaux sont écrits dans le fichier qui porte le même nom que le fichier

de la grammaire et l'extention qui est le nom de la forme. L'ecriture est réalisée sous la forme
textitMbrGauche : MbrDroit et l'axiome est toujours écrite en premier lieu.

4.2 Algorithmes supplementaires

Les fonctions supplémentaires sont stoquées dans le fichier utils/utils.py.

1. Creation des non terminaux
   Afin de pouvoir créer de nouveaux non terminaux, 2 variables sont utilisées : la première stocke la lettre du non terminale courant (de A à Z hors E) et la deuxième stocke le numéro du non terminale courant (de 0 à 9).

   Après la lecture du fichier, la lettre et le numéro courants sont mis à jour en fonction du non terminale le plus grand utilisé, c'est à dire, celui ayant la lettre dont la valeur du code ascii est la plus importante et numéro le plus grand des numéros des membres droits commençants par cette lettre.

   Afin de creer un nouveau non ternimal le numéro du non terminale courrant est augmenté. S'il est impossible d'incrémenter le numéro car le numéro actuel est égal à 9, le code ascii de la lettre courrante est incrémenté et le numéro redevient égal à 0. Lorsqu'il est impossible de créer un nouveau non ternimale car le non terminal courant est Z9, une fonction de suppression des non terminaux non utilisés, dupliqués ou non accéssibles est utilisée et tous les non ternimaux sont renommés en commençant par A0. S'il est impossible de libérer des non terminaux le programme renvoie une erreur "Nombre de terminaux utilisés est trop important".

   ϵ est notée par "E". D'après l'enoncée les non terminaux ne peuvent pas commencer par "E". Cette règle est respectée lors de la création de nouveaux non terminaux.

   Cette structure a été choisie car elle est facilement changeable et applicable en cas de changement de règles de nommage des non ternimaux et en cas d'augmentation du nombre de non terminaux possibles (ex: nommer les non terminaux par 1 lettre et 5 nombres) cette méthode prends moins d'espaces mémoire que par exemple les tableaux de n$*$n valeurs qui indiquent si le terminale a été pris ou pas. Cette méthode peut nécessiter une renommage des non ternimaux si le fichier des règles initiales contients des non terminaux qui "sont grands" (ex: Z9) mais cela reste un fait rare.

2. Collecte des non terminaux L'algorithme de collecte des non terminaux supprime les règles non accessibles depuis l'axiome et les règles dont l'ensemble des membres droits sont strictement identiques. Puis il effectue le renommage de tous les non terminaux à partir de A0.

3. Affichage des non terminaux Les fonctions d'affichage des non terminaux des membres droits, membres gauches et d'union des deux existent et sont utilisées dans

d'autres fonctions.

4.3 Fonctions de changements de forme

1. Generalités
   Afin de faire correspondre la forme de la grammaire donnée à forme normale de Chomsky ou de Greibach, notre programme suit les algorithmes suivants:

   Conversion en forme normal de Chomsky
   1. retirer l'axiome des membres droits des règles ;
   2. supprimer les règles X $\rightarrow \epsilon$ sauf si X est l'axiome ;
   3. supprimer les règles unité X $\rightarrow$ Y ;
   4. supprimer les non-terminaux en tête des règles ;
   5. supprimer les symboles terminaux qui ne sont pas en tête des règles.

   Conversion en forme normal de Greibach
   1. retirer l'axiome des membres droits des règles ;
   2. supprimer les terminaux dans le membre droit des règles de longueur au moins deux ;
   3. supprimer les règles avec plus de deux non-terminaux ;
   4. supprimer les règles X $\rightarrow \epsilon$ sauf si X est l'axiome ;
   5. supprimer les règles unité X $\rightarrow$ Y.

   On a choisi d'éliminer également la reccursion gauche immédiate afin de minimiser le risque d'occurence de "boucles mortelles".

2. Retirer l'axiome des membres droits des règles
   L'axiome initial de la grammaire est remplacée par un nouveau non terminal généré. Cette fonction est effectuée même si l'axiome n'est pas présente dans des membres droit. En effet, cette fonction ne fait créer qu'un seul non terminale et le fait de vérifier la présence de l'axiome dans les membres droits a une compléxitée non négligeable: $O(nb\_Var)$.
   Afin de préserver la structure de la grammaire, les règle sont réorganisées :
   - Une nouvelle règle est ajoutée ou l'ancien axiome dérive le nouveau
   - Toutes les règles ayant comme membre gauche l'ancien axiome voient ce dernier remplacé par le nouveau axiome
   - Toutes les occurrences de l'ancien axiome dans les membres droits des règles sont remplacées pour le nouvel axiome.

3. Supprimer les règles X $\rightarrow \epsilon$ sauf si X est l'axiome

   Si une règle non axiome contient ϵ, l'algorithme sauvegarde sont membre gauche et parcourt les membres droits de toutes les règles pour détercter si le non terminale sauvegardé y est présent. Si une telle situation est détéctée, alors on ajoute à cette

règles toutes les combinaisons possibles de suppression ou de non suppression du non terminale sauvegardé.

L'algorithme d'énumération les toutes les combinaisons possibles contenant ou pas un élément donné, surtout si l'élément est présent en copies multiples, peut avoir une très grande complexité. Afin de la minimiser on utilise l'algorithme suivant:
- Déterminer les indices du non terminale sauvegardé dans la liste des valeurs faisant partie du membre droit.
- Déterminer le nombre de combinaisons possibles de suppréssion de cet élément soit $2^{nb-occurences-d-element}$.
- Pour i allant de 0 à nb_occurences faire :
- Pour j allant de 0 à nb_occurences faire : si le bit correspondant à la position j dans la représentation binaire de i est égale à 1 alors on la lettre à la position j est supprimée et le resultat est sauvegardé.
Cette méthode permet d'obtenir toutes les combinaisons en $O(nb\_occurences)^2$.

Pour la meilleur compréhension du processus, voir la fonction *change_ regles_ with_ eps()*.

4. Supprimer les non-terminaux en tête des règles

Si le premier caractère est un non terminale alors ce caractère, ainsi que la suite du membre droit sont sauvegardés ensuite la règle est supprimée. Après l'ensembre de règles {*MbrGauche de la règle initiale : MbdDroit des règles dont le non terminale sauvegardé est le membre gauche + suite sauvegardée*} sont crées.

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

Afin de supprimer les terminaux dans le membre droit des règles de longueur au moins deux on applique l'algorithme suivant:

- A chaque terminale on associe un nouveau non terminale et on stoque les valeurs dans un dictionnaire dont les clés sont les ternimaux et les valeurs sont une liste de non terminle associé et un boolean qui indique si ce terminale a été remplacé ou pas (initié à False)
- Pour chaque membre droit: si sa longeur est supérieure à 1:
- Pour chaque caractère du membre droit : si ce caractère est dans la liste des clés du dictionnaire défini auparavant, il est replacé par le non terminale associé.
- Pour chaque terminale replacé, la règle {*Non terminale assossié : terminale remplacé* } est crée.

6. Supprimer les règles avec plus de deux non-terminaux

Comme cette règle est toujours appliquée après avoir appliquer la fonction de supprésion des terminaux dans le membre droit des règles de longueur au moins deux, si

la longeur du membre droit choisi lors d'application de cette fonction est suppérieur à 2, il contient obligatoirement 2 non terminaux.

Afin de le faire on applique l'algorithme suivant:
- Pour chaque membre droit, si sa longeur est supérieure à 2:
- La regle actuelle est remplacée par la regle { *MbrGauche : [premier caractère du MbrDroit , Nouveau NonTerm N]* }
- Pour chaque caractères qui suivent juqu'au l'avant dernier (exclu) : créer une regle { *Nouveau NonTerm crée à l'étape N-1 : [caractère, Nouveau NonTerm de l'étape N]* }
- Enfin la règle { *Nouveau NonTerm crée à l'étape N-1 : [avant dernier caractère, dernier caractère]* } est crée.

7. Supprimer les règles unité X → Y

   Les règles unitaires de la forme X → Y avec X et Y comme non terminaux sont éliminées
   Toutes les règles de la grammaire sont parcourues et lorsque une règle unitaire de la forme X → Y, Y y est remplacé par l'ensemble des membres droits des règles dont Y est le membre gauche. Le membre droit de la règle unitaire est ainsi remplacé par la production de la règle qui contient Y est tant que mebre gauche.
   La grammaire résultante n'a ainsi plus de règle unitaire et contient uniquement des règles qui produisent des séquences de terminaux et/ou non-terminaux, ou des terminaux seuls.

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

   Tous les terminaux qui ne figurent pas en tête des règles sont remplacés par un non-terminal à leur position dans la règle. Une nouvelle règle est ensuite créée, où ce non-terminal devient le membre gauche et le terminal remplacé le membre droit. Ce processus est appliqué à chaque règle de la grammaire.

4.4 Génération de mots

Le code pour la génération des mots se trouve dans le fichier utils/forms.py, fonction *tous_mots()*. Son but est de génerer des mots de la longeur inférieure ou égale à la longeur L donnée.

Le même algorithme est utilisé pour la génération des mots à partir de la grammaire sous forme normale de Chomsky ou de Greibach car dans les deux cas, chaque non ternimale ajoute au plus un terminale à la longeur du mot.

Les membres droits des règles dont le membre gauche est l'axiome sont ajoutés à la liste "mots_possibles" si leur longeur est inférieure ou égale à la longeur des mots recherchés. Puis tant que cette liste est non vide, pour chaque mot possible est parcouru en recherche

de non ternimaux:

-si un non terminal T est trouvé, le mot est remplacé par l'ensemble de mots possibles où T est remplacé par les membres droits des règles dont T est le membre gauche et ayant une longeur inférieure ou égale à la longeur souhaitée.

- si le mot ne contient pas de non terminaux alors il est supprimé de la liste "mots_possibles" et sauvegardé dans la liste "mots_retenus".

Après tous les membres droits de "mots_retenus", qui sont dorénavant une liste de terminaux, sont transformés en chaîne de caractères grâce à la fonction *join()*.

- Les mots ainsi obtenus sont triés dans l'ordre lexicographique et écrits (*print*).

5 Fichiers d'exécution

5.1 Exécution de grammaire.py

Le fichier executable grammaire.py prend 3 paramètres:
- le nom du fichier contenant la grammaire et ayant l'extention *.general*
- le paramère non obligatoire d'entée en mode de débuggage
- le paramère non obligatoire de minimisation de non terminaux utilisés

Pour activer le débuggage passer 1 en deuxième paramètre. Par defaut le mode debeuggage n'est pas actif (0).
Pour activer l'écriture de la grammaire en minimisant le nombre de non ternimaux utilisés passer 1 en troisième paramètre. Par defaut ce mode n'est pas actif (0). Lorsque ce mode est activé, tous les non termiaux non accessibles depuis l'axiome ou ayant l'ensemble des membres droits strictement identiques sont supprimés. En plus les terminaux des membres gauches sont nommé dans l'ordre ascendant (de A0 pour axiome jusqu'à Z9).

Ce programme crée 2 fichiers *nom_fichier_d_entree.chomsky* contenant la grammaire sous forme normale de Chomsky et *nom_fichier_d_entree.greibach* contenant la grammaire sous forme normale de Graibach.

5.2 Exécution de generate.py

Le fichier executable generate.py prend 2 paramètres: - la longeur maximale des mots recherchés - le nom du fichier contenant la grammaire sous une forme normale et ayant l'extention *.chomsky* ou *.greibach*.

Ce programme écrit dans l'ordre lexicographique tous les mots dont la longeur est inférieure ou égale à la longeur donnée qu'on peut obtenir à partir de la grammaire donnée.

6 Options supplémentaires

6.1 Débuggage

Afin de vérifier le fonctionnement de la fonction le mode débuggage peut etre activé.

Dans ce cas le résultat de chaque fonction appliquée lors de la transformation de la grammaire dans sa forme normale sera écrit dans le terminal. En plus on vérifiera que les règles obtenus après ces transformations correspondent bien aux loix de la forme normale choisie (fonctions du fichier *utils/check_form.py*) sinon une erreur correspondante à la règle ne respectant pas la loix est levée.

6.2 Makefile

Le Makefile a pour but d'automatiser et de faciliter le chargement des grammaires, leur transformation en grammaires normales, la création des mots et la comparaison des mots obtenus selon la forme de la grammaire choisie.

- Charger : Chargement la grammaires afin d'en créer une grammaire qui respecte les règles de Chomsky, et une grammaire conforme aux règles de Greibach. Ensuite il génère l'ensemble des mots de longeurs inférieure ou égale à la langeur souhaitée.
- Comparer : Une fois l'ensemble de mots qu'on peut obtenir à partir la forme donnée de la grammaire est fait, les fichiers contenant les formes de Chomsky et de Greibach sont comparés pour identifier les différences entre les chaînes générées selon ces deux formes normales afin de visualiser de potentielles erreurs.

Cela permet de vérifier si les chaînes générées à partir des deux formes normales (Chomsky et Greibach) sont identiques ou présentent des différences, et d'analyser la conversion entre les différentes formes de grammaire.