Université de Versailles – Saint-Quentin-en-Yvelines
Université Paris-Saclay
Licence 3 Informatique

2024-2025

Projet – Transformation de grammaires

Version du 6 janvier 2025

Pierre Coucheney – pierre.coucheney@uvsq.fr
Kossi-Roland Etse – kossi-roland.etse@uvsq.fr
Franck Quessette – franck.quessette@uvsq.fr

Table des matières

Le but de ce projet est de programmer différentes transformations entre des grammaires algébriques puis d'utiliser ces grammaires pour générer des mots de leur langage.

1 Les formes de grammaires algébriques

Dans tout ce document, on note :

— par des **lettres majuscules** (éventuellement indicées) A, B, S, X, ... les symboles non-terminaux (variables) ;

— par des **lettres minuscules** a, b, ... les terminaux ;

— par des **lettres grecques** α, β, ... les mots constitués de terminaux et de non-terminaux. ;

— de plus S est toujours l'axiome.

Les différentes formes normales ne s'appliquent qu'aux grammaires algébriques.

1.1 Forme générale des grammaires algébriques

Un grammaire est algébrique si toutes les règles sont de la forme :

— $X \to \alpha$

Le membre gauche d'une règle est toujours un unique non-terminal et le membre droit un mot quelconque. Les formes normales ci-dessous donnent des contraintes sur le membre droit.

1.2 Forme normale de Greibach

Une grammaire algébrique est sous forme normale de Greibach si toutes les règles sont de la forme :

— $X \to aA_1A_2...A_n$ avec $n \geq 1$

— $X \to a$

— $S \to \varepsilon$ seulement si ε appartient au langage.

1.3 Forme normale de Chomsky

Une grammaire algébrique est sous forme normale de Chomsky si toutes les règles sont de la forme :

— $X \to YZ$

— $X \to a$

— $S \to \varepsilon$ seulement si ε appartient au langage.

2 But du projet

Le but est de lire une grammaire algébrique depuis un fichier et de transformer cette grammaire dans les formes normales de Greibach et de Chomsky. Puis à partir de chacune de ces nouvelles grammaires donner tous les mots dont la longueur est inférieure à une longueur donnée.

On a les conventions suivantes :

— L'ensemble des terminaux est l'ensemble des 26 lettres minuscules.

— La lettre `E` majuscule représente ε.

— Un non-terminal est une des 25 lettres majuscules (toutes sauf `E`), suivie d'un des 10 chiffres. Par exemple `T8`. Il y a donc 250 non terminaux possibles.

— Dans le fichier lu en entrée chaque ligne comprend une règle de la grammaire sous la forme : `membre_gauche : membre_droit`. Des espaces peuvent être présents mais doivent être ignorés lors de l'analyse lexicale de la règle. De plus l'axiome est le `membre_gauche` de la première règle du fichier.

3 Travail à faire

Vous devez trouver (livres, internet) les algos pour transformer une grammaire algébrique quelconque dans les formes normales. https://fr.wikipedia.org/wiki/Forme_normale_de_Chomsky https://fr.wikipedia.org/wiki/Forme_normale_de_Greibach peut-être un bon point d'entrée.

4 Code à faire

Votre programme doit :

— Définir une structure de données pour stocker une grammaire.

— Avoir une fonction `lire` pour lire un fichier contenant une grammaire et la stocker dans votre structure de données. L'extension du fichier doit être `.general` pour général.

— Avoir des fonctions `greibach`, `chomsky`, qui transforment une grammaire algébrique dans les formes normales dont elle ont les noms. Ces fonctions doivent donc générer une structure de données à partir d'une autre en aucun ne faire de l'affichage (sauf debugage).

— Avoir une fonction `ecrire` pour écrire une grammaire depuis votre structure de donnée vers un fichier. Le fichier aura comme extension un mot correspondant au nom de la forme normale, comme le nom des fonctions.

— Votre programma exécutable devra s'appeler `grammaire`.

— On doit pouvoir lancer le programme avec la commande : `grammaire xxx.general` qui doit générer les fichiers : `xxx.greibach` et `xxx.chomski`. `xxx` pouvant être n'importe quel nom.

— Faire un second programme qui doit s'appeler `generer` qui prend en argument un fichier contenant une des formes normales et un entier n et donne en sortie tous les mots de longueur inférieure ou égale à n générés par la grammaire contenue dans le fichier et triés en ordre lexicographique, un par ligne et sans espace.

Vous pouvez bien sûr définir toutes autre fonction utiles à votre programme.

5 Travail à rendre

Le travail est à faire **en binôme**. Vous devez rendre sur e-campus un fichier `NOM1_NOM2.zip` tel que :

— `NOM1` et `NOM2` sont les noms de famille en majusule des deux membres du binôme.
— Si le fichier n'est pas un `.zip` : `-3 points`
— Si le nom du fichier n'a pas la forme demandée : `-3 points`
— Le fichier `.zip` doit contenir un dossier qui doit s'appeler `NOM1_NOM2`, sinon `-3 points`.
— Dans ce dossier il doit y avoir vos programmes en Python ou en langage C ainsi qu'un fichier `makefile` permettant avec la commande `make` de compiler et d'exécuter sur 3 fichiers d'exemples qui vous seront fournis et éventuellement deux autres de votre choix. Pas de `makefile` `-3 points`. Commande `make` qui ne marche pas `-3 points`.
— Dans le dossier, il y aura les fichiers d'exemple de la forme `xxx.general`.
— Dans le dossier il y aura un fichier `binome.txt` contenant les noms, prénoms et numéros d'étudiants des deux membres du binôme. pas de fichier ou fichier pas clair : `-3 points`.
— Dans le dossier un fichier `NOM1_NOM2.pdf` expliquant votre structure de données ainsi que les algos que vous avez programmés. Si ce document est rédigé en LaTeX `+2 points`. Dans ce document les noms prénoms et numéros d'étudiants des deux memebres du binômes doivent être sur la première page avec le titre, sinon : `-3 points`.
— Si le fichier `.zip` n'est pas remis sur ecampus : `-3 points`.
— La date limite de remise du projet est **dimanche 19 janvier 2025** à **23h59**. `-1 point` par heure de retard.

6 Conseils

— Prenez le temps de bien comprendre les différents algos avant de commencer à programmer.
— Exécuter à la main ces algos sur deux exemples simples.
— Définissez une structure de données bien adaptée à vos différents algos.
— Utilisez un Makefile ou tout autre outil vous permettant de compiler et d'exécuter rapidement.
— Utilisez github (ou autre) pour sauvegarder vos programme.
— Utiliser `Lex` pour lire les fichiers facilement.
— Tester votre code sur les deux exemples simples qui permettent de facilement vérifier vos résultats «à la main».
— Utilisez vos programmes pour qu'ils comparent leurs résultats. Par exemple, si vous avez un fichier `test.general` la suite de commandes suivantes (si le code est en Pyhton) :

```
$> python grammaire test.general
$> python generer 4 test.chomsky > test_4_chomsky.res
$> python generer 4 test.greibach > test_4_greibach.res
$> diff test_4_chomsky.res test_4_greibach.res
```

La commande `diff` affiche les différence entre deux fichiers. Si elle n'affiche rien c'est que les deux fichiers sont identiques. Si la commande `diff` affiche quelque chose c'est qu'un de vos programme au moins n'est pas correct.