Chapitre II

Langages libres de contexte

1 Introduction

Le cours de compilation est un prérequis pour ce chapitre.

Il suppose que vous connaissez déjà ce qu'est un langage libre de contexte (GLC) et un automate à pile (AP).

Si vous n'êtes pas familier avec ces notions, vous devez rattraper vos lacunes.

Dans cette leçon, nous étudierons :
- une formalisation de ces modèles,
- les limites de ces modèles,
- une version du théorème de l'étoile pour les GLCs.

2 Grammaire libre de contexte

Exemple : soit une grammaire libre de contexte G.

$$\begin{cases} \mathbf{A} \to 0A1 \\ A \to B \\ B \to \# \end{cases} \qquad \begin{cases} \mathbf{A} \to 0A1 \mid B \\ B \to \# \end{cases}$$

Définition :
- un ensemble de variables : $\{A, B\}$.
- un ensemble de symboles : $\{0, 1, \#\}$ (dit symboles terminaux).
- des règles de réécriture des variables : $\mathbf{A} \to 0A1 \mid B, B \to \#$
- une variable de départ : A (en général, le lhs [1] de la première règle).

Utilisation :

1. prendre comme expression initiale la variable de départ.
2. choisir une des variables de l'expression, choisir l'une des règles de réécriture, et l'appliquer sur la variable choisie.
3. répéter l'état 2 jusqu'à ce qu'il n'y ait plus de variable à dériver.

Dérivation : $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

1. Left-Hand Side = l'expression à gauche de l'équation.

2.1 Définition formelle

Définition 1 : Grammaire libre de contexte (GLC)

Une grammaire G libre de contexte est un 4-tuple (V, Σ, R, S) où :

— V est un ensemble fini de variables.

— Σ est un ensemble fini de symboles terminaux.

— R est un ensemble fini de règles (= méthodes de réécriture d'une variable $v \in V$ par une concaténation finie de variables de V (y compris v), de symboles terminaux de Σ, ou ϵ).

— S est la variable de départ.

Définition 2 : Langage libre de contexte

Un langage libre de contexte est l'ensemble des mots possibles engendrés par une grammaire libre de contexte.

Définition 3 : Classe des langages libres de contexte

La classe des langages libres de contexte est constitué de l'ensemble des langages libres de contexte.

2.2 Modèle de traitement

Notations :

$u \Rightarrow v$ **:** s'il existe une règle r de réécriture de R qui transforme u en v.

$u \overset{*}{\Rightarrow} v$ **:** si $u = v$ ou s'il existe une suite de réécritures $u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k$ telle que $u_1 = u$ et $u_k = v$.

Exemple :

soit la grammaire $G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1 \mid \epsilon\}, S)$

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 0000S1111 \Rightarrow 00001111$

$S \overset{*}{\Rightarrow} 00001111$

$\mathcal{L}(G) = \{0^n 1^n \mid n \geq 0\}$

Définition 4 : Modèle de traitement

$G = (V, \Sigma, R, S)$ accepte la chaine w s'il existe une suite de règle de R telle que $S \overset{*}{\Rightarrow} w$.

Remarque : Le processus de dérivation est non déterministe.

A savoir que si la dérivation d'une variable est constitué de plusieurs règles, alors un choix non-déterministe doit être effectué.

2.3 Conception d'une GLC

Construction d'une GLC $G = (V, \Sigma, R, S)$

1. Avec des sous-chaînes de longueur dépendante :

   utiliser une règle de la forme $R \rightarrow uRv$ pour forcer la dépendance entre les sous-chaînes (exemple : $u^n \# v^n$).

2. à partir l'union de plusieurs GLCs $G_i = (V_i, \Sigma_i, R_i, S_i)$
   — renommer les variables des V_i tels qu'ils soient disjoints entre eux (réappellation à propager dans R_i et S_i).
   — $V = \bigcup_i V_i, \quad \Sigma = \bigcup_i \Sigma_i, \quad R = \bigcup_i R_i$
   — ajouter une nouvelle règle à $R : S \rightarrow S_1 | S_2 | \cdots | S_n$ et prendre S comme symbole de départ.

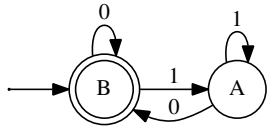
Exemple :

$L_1 = \{0^n 1^n \mid n \geq 0\}$ généré par $S \rightarrow 0S1|\epsilon$
$L_2 = \{1^n 0^n \mid n \geq 0\}$ généré par $S \rightarrow 1S0|\epsilon$
$L = L_1 \cup L_2$ généré par $S \rightarrow S_1|S_2$
$S_1 \rightarrow 0S_1 1 \mid \epsilon$
$S_2 \rightarrow 1S_2 0 \mid \epsilon$

3. à partir d'un ADF $L = (Q, \Sigma, \delta, q_0, F)$ (langage régulier) :
   — associer une variable $V_i \in V$ à chaque état $q_i \in Q$.
   — l'ensemble des symboles est évidemment le même.
   — pour chaque transition $\delta(q_i, a) = q_j$, ajouter à R la règle $V_i \rightarrow aV_j$.
   — pour chaque état final $q_i \in F$, ajouter la règle $V_i \rightarrow \epsilon$.
   — faire de V_0 l'état de départ (associé à l'état q_0).
   on l'appelle une **grammaire linéaire**.

Exemple :

ADF



GLC
$G = (\{V_0, V_1\}, \{0, 1\}, R, V_0)$
où $V_0 \rightarrow 0V_0 \mid 1V_1 \mid \epsilon$
$V_1 \rightarrow 1V_1 \mid 0V_0$

2.4 Fermeture d'une GLC

Théorème 1 : fermeture
Les GLCs sont fermés par les opérateurs d'union, de concaténation et étoile.
Démonstration :
— **concaténation et union** :
déjà démontré pour l'union, même construction pour la concaténation. □
— **étoile** : (opétateur unaire)
construire la GLC $G' = (V, \Sigma, R', S')$ à partir $G = (V, \Sigma, R, S)$, en ajoutant, un nouveau symbole S', et $R' = R \cup \{S \rightarrow \epsilon \mid SS\}$ □

Théorème 2 : Langage régulier et GLC
Tout langage régulier peut être décrit par une GLC.
Démonstration : voir la preuve constructive des grammaires linéaires. □

2.5 Forme normale de Chomsky

Définition Forme normale de Chomsky (FNC)

une GLC est sous forme normale de Chomsky si chaque règle est de la forme :

$A \rightarrow BC$
$A \rightarrow a$
$S \rightarrow \epsilon$

où : — a est n'importe quel terminal.
— A, B, C, S sont des variables :
— S est la variable de départ.
— A est n'importe quelle variable (y compris S).
— B et C sont des variables différentes de S.

Exemple : **GLC (pas FNC)** $S \to uSv \mid \epsilon$

GLC sous FNC (la même)
$S \to AB \mid UV \mid \epsilon$
$A \to UU \mid AU$
$B \to VV \mid BV$
$U \to u$
$V \to v$

Raisons pour lesquelles une CGF n'est pas sous FNC :

1. la variable de départ apparait dans la RHS.
2. règle ϵ : par exemple $A \to \epsilon$ (où A n'est pas la variable de départ).
3. règle unitaire : tel que $A \to A$ ou $B \to C$.
4. la règle n'a pas exactement 2 variables ou un terminal dans le RHS.

Exemple 1 : $S \to uSv \mid \epsilon \equiv$ $S \to uSv$ cas 1 et 4.
$S \to \epsilon$ ok.

Exemple 2 : $S \to ASA \mid aB \equiv$ $S \to ASA$ cas 1 et 4.
$A \to B \mid S$ $S \to aB$ cas 4.
$B \to b \mid \epsilon$ $A \to B$ cas 3.
$A \to S$ cas 1 et 3.
$B \to b$ ok.
$B \to \epsilon$ cas 2.

Quel est l'intérêt pour une GLC d'être sous une FNC ?

Sous une FNC,
— l'application d'un règle de type $A \to BC$ fait grandir la dérivation d'une variable,
— l'application d'un règle de type $A \to a$ remplace une variable par un terminal.

En conséquence, pour obtenir une chaîne de longueur n, il faut utiliser :
— $n - 1$ règles de type $A \to BC$,
— n règles de type $A \to a$.

Cette propriété est algorithmiquement importante afin d'évaluer le nombre de règle à appliquer afin d'obtenir une chaine de longueur n.

Exemple :

Permet de répondre à la question : combien de mots de longueurs 5 peuvent être obtenu avec la GLC sous FNC suivante ?

$S \to AB|AA$, $A \to AB|AA|a$, $B \to b$

D'où l'importance du théorème suivant :

Théorème 3 : conversion GLC vers CNF

Tout langage libre du contexte peut être généré par une grammaire libre du contexte sous forme normale de Chomsky.

Démonstration :

Méthode constructive pour obtenir une FNC à partir d'une CGF.

Trivialement, toutes les règles de transformation de la grammaire données ci-dessous ne modifient pas le langage généré.

Étape 1 (cas 1) pour supprimer S dans les RHS, ajouter une nouvelle variable de départ S_0 et une règle $S_0 \to S$ où S est la variable de départ de G.
Assure que la nouvelle variable de départ n'est pas RHS.

Étape 2 (cas 2) pour supprimer une règle $A \to \epsilon$, pour chaque occurrence de A dans une RHS, ajouter une nouvelle règle, en supprimant l'occurrence de A.
exemple : $R \to uAvAw \Rightarrow R \to uAvw \mid uvAw \mid uvw$

Étape 3 (cas 3) supprimer une règle unitaire $A \to B$,
pour chaque règle $B \to u$ (où u est une chaîne de variables et de terminaux), ajouter une règle $A \to u$ et supprimer la règle $A \to B$.
exemple : pour $A \to B, B \to aC, B \to CC$
ajouter $A \to aC, A \to CC$ et supprimer $A \to B$.
Étape 4 (cas 4) supprimer une règle $A \to u_1 u_2 \dots u_k$, pour $k > 2$ où chaque u_i est une variable ou un terminal.
remplacer cette règle par $A \to u_1 A_1, A_1 \to u_2 A_2, \dots, A_{k-2} \to u_{k-1} u_k$.
exemple : pour $A \to aABb$
remplacer par $A \to aA_1, A_1 \to AA_2, A_2 \to Bb$.
Étape 4' (cas 4') supprimer une règle $A \to u_1 u_2$ où u_1 ou u_2 est un terminal.
remplacer chaque terminal u_i par une variable U_i et ajouter une règle $U_i \to u_i$.
exemple : pour $A \to aA|Bb|ab$
remplacer par $A \to U_a A|BU_b|U_a U_b$, $U_a \to a$, $U_b \to b$.

□

Exemple : de transformation d'une CGF sous FNC.

$$\begin{array}{l} S \to ASA \mid aB \\ A \to B \mid \underline{S} \\ B \to b \mid \epsilon \end{array} \xrightarrow[S_0]{\text{étape 1}} \begin{array}{l} S_0 \to S \\ S \to ASA \mid aB \\ A \to B \mid S \\ B \to b \mid \underline{\epsilon} \end{array} \xrightarrow[B \to \epsilon]{\text{étape 2(a)}} \begin{array}{l} S_0 \to S \\ S \to ASA \mid aB \mid a \\ A \to B \mid S \mid \underline{\epsilon} \\ B \to b \end{array}$$

$$\xrightarrow[A \to \epsilon]{\text{étape 2(b)}} \begin{array}{l} S_0 \to S \\ S \to ASA \mid aB \mid a \mid SA \mid AS \mid \underline{S} \\ A \to B \mid S \\ B \to b \end{array} \xrightarrow[S \to S]{\text{étape 3(a)}} \begin{array}{l} S_0 \to \underline{S} \\ S \to ASA \mid aB \mid a \mid SA \mid AS \\ A \to B \mid S \\ B \to b \end{array}$$

$$\xrightarrow[S_0 \to S]{\text{étape 3(b)}} \begin{array}{l} S_0 \to ASA \mid aB \mid a \mid SA \mid AS \\ S \to ASA \mid aB \mid a \mid SA \mid AS \\ A \to \underline{B} \mid S \\ B \to b \end{array} \xrightarrow[A \to B]{\text{étape 3(c)}} \begin{array}{l} S_0 \to ASA \mid aB \mid a \mid SA \mid AS \\ S \to ASA \mid aB \mid a \mid SA \mid AS \\ A \to b \mid S \\ B \to b \end{array}$$

$$\xrightarrow[\text{retirer } A \to S]{\text{étape 3(d)}} \begin{array}{l} S_0 \to \underline{ASA} \mid aB \mid a \mid SA \mid AS \\ S \to \underline{ASA} \mid aB \mid a \mid SA \mid AS \\ A \to b \mid \underline{ASA} \mid aB \mid a \mid SA \mid AS \\ B \to b \end{array} \xrightarrow[\text{chaîne } ASA]{\text{étape 4}} \begin{array}{l} S_0 \to AU \mid \underline{a}B \mid a \mid SA \mid AS \\ S \to AU \mid \underline{a}B \mid a \mid SA \mid AS \\ A \to b \mid AU \mid \underline{a}B \mid a \mid SA \mid AS \\ B \to b \\ U \to SA \end{array}$$

$$\xrightarrow[\text{chaîne } aB]{\text{étape 4'}} \begin{array}{l} S_0 \to AU \mid VB \mid a \mid SA \mid AS \\ S \to AU \mid VB \mid a \mid SA \mid AS \\ A \to b \mid AU \mid VB \mid a \mid SA \mid AS \\ B \to b \\ U \to SA \\ V \to a \end{array}$$

La grammaire obtenue est sous Forme Normale de Chomsky et génère exactement le même langage.

3 Automate à pile

Rappel :

Un Automate à Pile (AP) est un Automate Non-déterministe Fini (ANF) auquel a été ajouté pile LIFO de taille infinie.

A chaque transition, un automate à pile :

— lit un caractère de la chaine d'entrée (sauf s'il s'agit d'une transition ϵ)
— dépile (éventuellement) le symbole au sommet de la pile.
— empile (éventuellement) un symbole sur la pile.

La notation $a, b \rightarrow c$ signifie que la machine lit a depuis l'entrée, dépile b et pousse c sur la pile. ϵ signifie une absence d'opération.

Exemple :

Fonctionnement :

tant que l'on lit un 0 : le pousser sur la pile.

tant que l'on lit un 1 : dépiler un 0 de la pile, et s'il n'y en a pas, rejeter.

si la pile est vide, alors accepter, sinon rejeter.

Le langage reconnu est $L = \{0^n 1^n \mid n \geq 0\}$.

Remarques :

— on ne peut pas tester si la pile est vide.
Une solution consiste à utiliser un symbole particulier (par exemple \$).
On commence avant le début de la lecture par placer un \$ au sommet de la pile (transition de la forme $\epsilon, \epsilon \rightarrow \$$).
Si on lit le symbole \$ sur le sommet de la pile, alors la pile est vide.
— on ne peut pas tester si la chaîne d'entrée a été traitée.
Comme pour un ADF, la chaîne d'entrée est acceptée si après le dernier symbole lu, on se trouve dans un état acceptant.
— Dans l'exemple précédent (reproduit ci-dessous), le non-détermisme est utilisé :
  — pour empiler autant de 0 à la lecture des 0s qu'il sera nécessaire à d'en dépiler à la lecture des 1,
  — pour s'assurer que la pile est vide lorsque l'on a fini de lire tous les symboles

3.1 Définition

Définition 5 : Définition formelle d'un automate à pile

Un AP est un 6-uple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ où :

- Q un ensemble fini d'états.
- Σ un alphabet.
- Γ les symboles de la pile.
- δ une fonction de transition de la forme :
  $\delta : Q \times \Sigma' \times \Gamma' \to \mathcal{P}(Q \times \Gamma')$
  où $\Sigma' = \Sigma \cup \{\epsilon\}$ et $\Gamma' = \Gamma \cup \{\epsilon\}$.
- q_0 est l'état de départ.
- F est l'ensemble des états acceptants.

Le 6-uple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ définit complètement l'automate à pile.

Remarques :

- les symboles de la pile peuvent être différents des symboles de l'alphabet.
- la fonction de transition,
  - a trois paramètres en entrée : l'état courant, le caractère à lire sur l'entrée, et le symbole à dépiler au sommet de la pile.
  - a un couple de paramètre en sortie : l'état dans lequel on passe (transition) et le symbole à empiler au sommet de la pile.

Exemple : AP qui reconnaît $L = \{0^n 1^n \mid n > 0\}$ (on notera que le cas ϵ est traité car $q_s \in F$).

États : $Q = \{q_s, q_0, q_1, q_e\}$
État de départ : q_s
États acceptants : $F = \{q_s\}$
Alphabet de l'entrée : $\Sigma = \{0, 1\}$
Alphabet de la pile : $\Gamma = \{0, \$\}$

Fonction de transition :

$\delta(q_s, \epsilon, \epsilon) = \{(q_0, \$)\}$
$\delta(q_0, 0, \epsilon) = \{(q_0, 0)\}$
$\delta(q_0, 1, 0) = \{(q_1, \epsilon)\}$
$\delta(q_1, 1, 0) = \{(q_1, \epsilon)\}$
$\delta(q_1, \epsilon, \$) = \{(q_e, \epsilon)\}$

Fonction de transition : δ

Σ'	0			1			ϵ		
Γ'	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_s	{}	{}	{}	{}	{}	{}	{}	{}	$\{(q_0, \$)\}$
q_0	{}	{}	$\{(q_0, 0)\}$	$\{(q_1, \epsilon)\}$	{}	{}	{}	{}	{}
q_1	{}	{}	{}	$\{(q_1, \epsilon)\}$	{}	{}	{}	$\{(q_e, \epsilon)\}$	{}
q_e	{}	{}	{}	{}	{}	{}	{}	{}	{}

3.2 Modèle de traitement

Définition : modèle formel de traitement informatique d'un automate à pile

Un AP $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepte la chaîne d'entrée w si :

- w s'écrit comme une chaîne de symboles $w = w_1 w_2 \dots w_n$ où $w_i \in \Sigma'$.
- il existe des états $r_0, r_1, \dots, r_m$ dans Q,
- il existe des états $s_0, s_1, \dots, s_m$ dans Γ^* (= état de la pile)

qui vérifient :

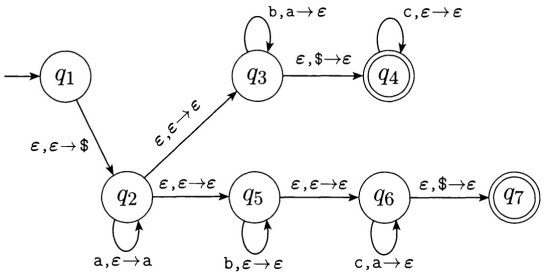
1. $r_0 = q_0$, $s_0 = \epsilon$ (assure que l'AP commence en q_0, avec une pile vide).

2. pour $i = 0, 1, \ldots, m-1$, on a $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ où $s_i = at$, $s_{i+1} = bt$ où $a, b \in \Gamma'$ et $t \in \Gamma^*$. (assure que la transition de l'AP et la mise à jour de la pile est correcte).
3. $r_m \in F$ (assure que l'AP est dans un état acceptant après avoir traité l'entrée).

3.3 Conception

Exemple : $L = \left\{a^i b^j c^k \mid i, j, k \geq 0 \text{ et } i = j \text{ ou } i = k\right\}$

Comment construire l'automate à pile qui reconnaît ce langage ?
— pousser les a lus sur la pile.
— deviner si le nombre de a doit être comparé avec le nombre de b ou de c (fait par une transition non déterministe).



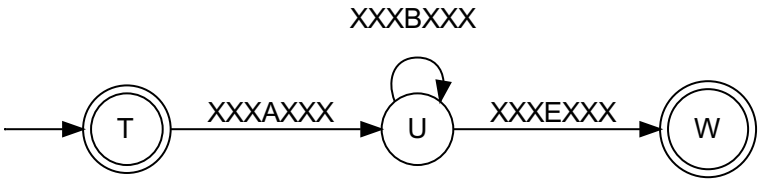
3.4 Équivalence avec GLC

Lemme 4 : AP ⇒ GLC
un langage est libre de contexte s'il est reconnu par un AP.

Démonstration : (preuve constructive)

Supposons qu'un langage L est engendré par une GLC $G = (V, \Sigma, R, S)$.

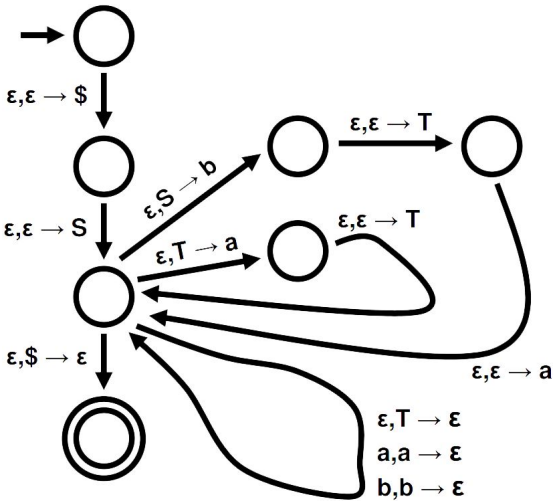
Construisons un AP $P = (Q, \Sigma', \Gamma', \delta, q_0, F)$ qui reconnaît L.



— placer le marqueur \$ et la variable de départ S dans la pile.
— répéter
  — si le sommet de la pile est une variable, sélectionner une règle associée à cette variable **de façon non déterministe**, et l'empiler.
  — si le sommet de la pile est un terminal, le dépiler et le comparer au symbole d'entrée. S'ils sont différents, alors rejeter.
  — si le sommet de la pile contient \$, alors entrer l'état acceptant.

L'automate à pile P ainsi défini simule alors trivialement la GLC G. □

Exemple : de simulation d'une GLC sur un AP.



GLC : $S \rightarrow aTb,\ T \rightarrow Ta \mid \epsilon$
Construire l'AP associé ainsi :

1. sans lecture, empiler le marqueur de fin de pile \$ et le symbole de départ du GLC.
2. sans lecture, pour chaque règle, empiler les variables et les symboles de la règle dans le sens inverse (FIFO).
3. pour chaque symbole s de l'alphabet, créer une règle qui lit s et dépile s.

Le choix des règles s'effectue grâce au fonctionnement non déterministe de l'AP.
La lecture de la bande est faite au fur et à mesure que les terminaux sont placés au sommet de la pile.

Lemme 5 : GLC ⇒ AP
si un automate à pile reconnait un langage, alors ce langage est libre de contexte.

Démonstration : (preuve constructive)

Normalisation de l'AP : tout AP N peut être transformé en un AP N' respectant les **3** règles suivantes :

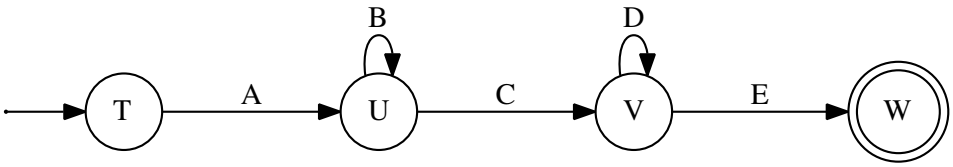
1. Il n'existe qu'un seul état acceptant q_a
   solution : créer un nouvel état acceptant q'_a, et relier les anciens états acceptants à q'_a par une transition ϵ sans mouvement de pile.
2. La pile est vide avant d'accepter.
   solution :
   — Ajouter un caractère \$ de détection de fin de pile à Γ'.
   — Ajouter un nouvel état de départ q'_s relié à l'ancien q_s par une transition qui sans lire l'entrée, place \$ sur la pile.
   — Ajouter un nouvel état q''_a relié à q'_a par une transition qui dépile \$. Ajouter des transitions qui relient q'_a à lui-même et qui dépilent tous les caractères autres que \$.
3. chaque transition empile **ou (exclusif)** dépile un symbole.
   solution :
   — Ajouter un symbole # à l'alphabet de la pile.
   — Ajouter un état intermédiaire dans les cas suivants :
   — **si une transition empile et dépile** : le faire en 2 temps
   $$q_i \xrightarrow{x,b\rightarrow c} q_j \quad \Rightarrow \quad q_i \xrightarrow{x,b\rightarrow\epsilon} q'_{ij} \text{ et } q'_{ij} \xrightarrow{\epsilon,\epsilon\rightarrow c} q_j$$
   — **si une transition n'empile ni ne dépile** : empiler et dépiler immédiatement #.
   $$q_i \xrightarrow{x,\epsilon\rightarrow\epsilon} q_j \quad \Rightarrow \quad q_i \xrightarrow{\epsilon,\epsilon\rightarrow\#} q'_{ij} \text{ et } q'_{ij} \xrightarrow{x,\#\rightarrow\epsilon} q_j$$

On peut donc supposer que tout AP N' vérifie ces 3 conditions.

On construit alors une GLC équivalente en transformant les transitions de l'AP en règles de la GLC.

Règles de la GLC équivalente : (Q = ensemble des états de l'AP).
— **règle initiale** : $S \to S_{\text{start}} S_{\text{end}}$
— **règles de terminaison** : $\forall q \in Q, S_{qq} \to \epsilon$
— **règles de mouvements de pile** : pour chaque couple de transitions
  — $p_1 \xrightarrow{x,\epsilon \to z} p_2$ (on empile z en lisant x)
  — $q_2 \xrightarrow{y,z \to \epsilon} q_1$ (on dépile z en lisant y)
  par $S_{p_1,q_1} \to x S_{p_2,q_2} y$.
— **règles de transition** : $\forall p, q, r \in Q$ tels qu'il existe un chemin de p à q passant par r, $S_{p,q} \to S_{p,r} S_{r,q}$.
  note : lien entre états n'ayant pas de mouvements de pile.
On peut montrer que la GLC ainsi générée engendre le même langage que l'AP (par récurrence, en montrant que toute exécution de l'AP est simulée par cette GLC). □

Exemple : $L = \{0^n 1^n \mid n > 0\}$



— **règles initiales** : $S \to S_{q_s q_e}$
— **règles de terminaison** : $S_{q_0 q_0} \to \epsilon$
$S_{q_1 q_1} \to \epsilon$
— **règles de mouvements de pile** :

$q_s \xrightarrow{\epsilon,\epsilon \to \$} q_0$	$q_1 \xrightarrow{\epsilon,\$ \to z} q_e$	$S_{q_s q_e} \to \epsilon S_{q_0 q_1} \epsilon$
$q_0 \xrightarrow{0,\epsilon \to 0} q_0$	$q_0 \xrightarrow{1,0 \to \epsilon} q_1$	$S_{q_0 q_1} \to 0 S_{q_0 q_0} 1$
$q_0 \xrightarrow{0,\epsilon \to 0} q_0$	$q_1 \xrightarrow{1,0 \to \epsilon} q_1$	$S_{q_0 q_1} \to 0 S_{q_0 q_1} 1$

— **règles de transition** : $S_{q_s q_e} \to S_{q_s q_0} S_{q_0 q_e}$
$S_{q_s q_e} \to S_{q_s q_1} S_{q_1 q_e}$
$S_{q_0 q_e} \to S_{q_0 q_1} S_{q_1 q_e}$

4 Lemme de l'étoile

Lemme de l'étoile pour les GLCs

Pour toute grammaire libre du contexte G, il existe un entier p, tel que pour toute chaîne $w \in G$ de longueur au moins p, on peut trouver u, v, x, y, z alors $w = uvxyz$ et :
— pour tout $i \geq 0$, $uv^i xy^i z$ est dans L,
— $|vy| > 0$
— $|vxy| \leq p$

Démonstration :

soit G une grammaire libre du contexte, et A le langage engendré par G.
— soit b le nombre maximum d'enfants dans l'arbre d'analyse grammaticale de n'importe quelle chaîne de L.
  i.e. le RHS de n'importe quelle règle a au plus b terminaux et variables.
— soit $n = \#V$ où $\#V$ est le nombre de variables dans G.
— on prend comme longueur de pompage $p = b^{n+2}$
  rappel : à la profondeur d, on a au plus b^d fils.
Cette profondeur p implique qu'une variable a été utilisée au moins 2 fois.

Plus formellement, si une chaîne s a une longueur $\#s \geq p$, alors l'arbre T d'analyse grammaticale de s a une profondeur $\geq n + 2$.

Donc, il existe au moins une branche de l'arbre de longueur $\geq n + 2$.

On suppose que T est de plus petite profondeur possible (ne peut pas être réduit en supprimant des transitions $S' \to \epsilon S'\epsilon \mid \epsilon S' \mid S'\epsilon$). Si cela n'était pas le cas, on ne pourrait pas lier la profondeur de l'arbre n à la longueur de la chaîne produite p.

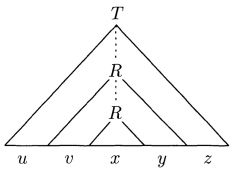
Une descente dans l'arbre implique l'utilisation d'une règle remplaçant une variable par une autre. Comme ceci arrive $n + 2$ fois, et qu'il n'y a que n variables, une variable R a été répétée au moins une fois.

Découpons la chaîne s comme $s = uvxyz$:

chaque occurrence de R produit une chaîne :

— la première occurrence produit la chaîne vxy.

— la seconde occurrence produit la chaîne x.



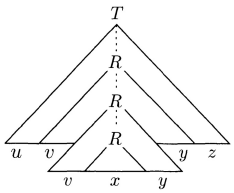
Or si la seconde occurence de R produit la même chaîne que la première, on obtient :

$s = uv^2xy^2z$

En réitérant le processus, on obtient :

$s = uv^ixy^iz$ pour $i > 0$

et $s \in \mathcal{L}(T)$.



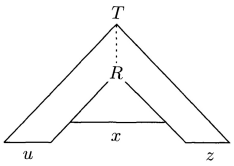
Inversement, si la première occurence de R produit la même chaîne que la seconde, on obtient une version réduite :

$s = uxz$

Par conséquent, si :

$s = uv^ixy^iz$ pour $i \geq 0$

alors $s \in \mathcal{L}(T)$.



Vérifions maintenant les 2 conditions supplémentaires du théorème :

— $|vy| > 0$ ($uvxyz$)

si v et y étaient tous deux ϵ (i.e. $|vy| = 0$), on serait en contradiction avec le choix que l'arbre T est minimum (sinon $uvxyz$ et uxz représentent la même chaîne).

— $|vxy| \leq p$

La première occurrence de R génère vxy. Or R a été choisi tel que les 2 occurrences de R tombent dans un chemin de longueur $\leq n + 1$ (le chemin se termine nécessairement par un terminal).

Le sous-arbre qui génère vxy a donc une hauteur maximale $n + 2$.

Un arbre de cette profondeur génère une chaîne de longueur au plus $p = b^{n+2}$. □

Remarque : comme dans le cas des ADFs,

— on ne peut l'utiliser que pour montrer qu'un langage n'est pas libre du contexte.

— il faut montrer qu'**aucune** écriture sous cette forme n'est possible.

Exemple 1 : $A = \{a^n b^n c^n \mid n \geq 0\}$ ne peut pas être généré par une GLC.

Soit k la longueur critique, considérons $s = a^k b^k c^k$.

Montrons qu'aucune écriture de s sous la forme $uvxyz$ ne permet de pomper s. Si v ou y contiennent :

— **plus d'un symbole** : les symboles de uv^2xy^2z ne sont pas dans le bon ordre (exemple : si $v = ab$, $v^2 = abab$).

— **un seul symbole** : si $uvxyz \in A$, alors $uv^2xy^2z \notin A$ car le nombre a, b et c n'est plus équilibré.

Exemple 2 : $B = \{ww \mid w \in \{0,1\}^*\}$ ne peut pas être généré par une GLC.

Soit k la longueur critique, et considérons $s = 0^k 10^k 1 \in B$.

Supposons que s s'écrit sous la forme $s = uvxyz$ avec $u = 0^{k-1}$, $v = 0$, $x = 1$, $y = 0$ et $z = 0^{k-1}1$. Alors,

$uvxyz = 0^{k-1}0100^{k-1}1 = 0^k10^k1 \in B$

$uv^{p+1}xy^{p+1}z = 0^{k-1}0^{p+1}10^{p+1}0^{k-1}1 = 0^{p+k}10^{k+p}1 \in B$

Donc, 0^k10^k1 peut être pompé, cherchons un autre exemple.

Prenons maintenant $s = 0^k1^k0^k1^k \in B$ où k est la longueur critique.

Supposons que $s = uvxyz$ et $|vxy| \leq k$. Comme pour le cas précédent, v ou y ne peuvent contenir qu'un seul symbole. Alors si vxy :

— **est la partie gauche de** s : alors pour $s = uv^pxy^pz$, la partie droite n'est plus symétrique (de la forme $0^i1^i0^p1^p$).

— **est la partie droite de** s : inversement, la partie gauche n'est plus symétrique (de la forme $0^p1^p0^i1^i$).

— **est la partie centrale de** s : par pompage, on obtient une expression de la forme $0^p1^i0^j1^p$ où i et j sont différents de p (puisqu'issus du pompage).

En conséquence, aucune forme de $s = uvxyz$ ne peut être pompée, et B ne peut pas être générée par une GLC.

5 Complément sur la fermeture

Théorème 6 : GLC et fermeture par intersection
Les GLCs ne sont pas fermés par l'opérateur d'intersection.

Démonstration : en exhibant un contre-exemple.

Soient G_1 et G_2 les deux GLCs suivantes :

$G_1 = (\{S_1, A_1, B_1\}, \{0,1\}, R_1, S_1)$
où $R_1 = \begin{array}{l} S_1 \to A_1B_1 \\ A_1 \to 0A_11 \mid 01 \\ B_1 \to 2B_1 \mid \epsilon \end{array}$
$L_1 = \mathcal{L}(G_1) = 0^n1^n2^*$

$G_2 = (\{S_2, A_2, B_2\}, \{0,1\}, R_2, S_2)$
où $R_2 = \begin{array}{l} S_2 \to A_2B_2 \\ A_2 \to 0A_2 \mid \epsilon \\ B_2 \to 1B_22 \mid 12 \end{array}$
$L_2 = \mathcal{L}(G_2) = 0^*1^n2^n$

Alors $L = L_1 \cap L_2 = 0^n1^n2^n$.

Si G_1 et G_2 sont des GLCs, on a vu dans l'exemple 1 juste avant que L ne pouvait pas être généré par un GLC. Donc, $G_1 \cap G_2$ n'est pas une GLC. □

Théorème 7 : GLC et fermeture par complémentation
Les GLCs ne sont pas fermés par l'opérateur de complémentation.

Démonstration : comme $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, la non-fermeture par intersection, implique la non-fermeture par complémentation. □

Exemple : d'un langage tel que L ne soit pas généré par une GLC mais tel que $\overline{L}$ le soit ?

Soit $L = \{ww \mid w \in \{0,1\}^*\}$.
Alors $\overline{L}$ peut être vu comme l'union des deux langages L_1 et L_2 suivants :

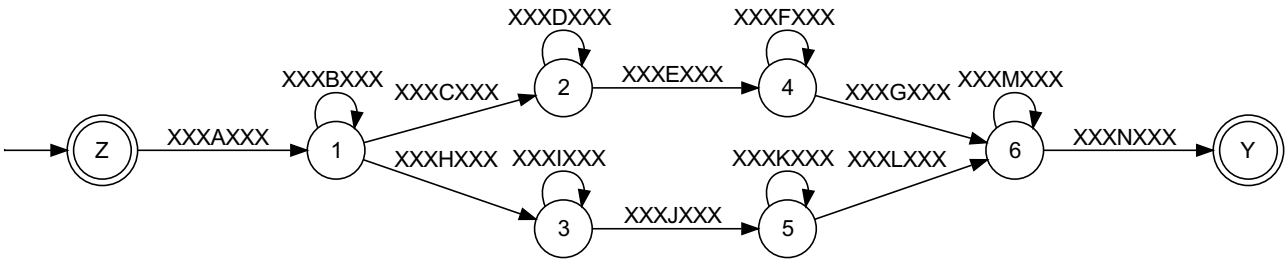
— L_1 : pour tout $y \in L_1$, la longueur de y est impaire.
car toute chaîne de L est de longueur paire.
exemple de grammaire générant L_1 : $S_1 \to TS_2$, $S_2 \to TTS_2 \mid \epsilon$ et $T \to 0 \mid 1$.

— L_2 : pour tout $y \in L_2$ de longueur paire $2p$, alors il existe $1 \leq i \leq p$, $y_i \neq y_{i+p}$.
Pour accepter y s'assurer que y_i et y_{i+p} sont différents. La chaîne y peut alors se voir comme :

$w_1, \ldots, w_{i-1}$	w_i	$w_{i+1}, \ldots, w_p$	$w_{p+1}, \ldots, w_{p+i-1}$	w_{p+i}	$w_{p+i+1}, \ldots, w_{2p}$

Autrement dit, le nombre de caractères qu'il y a **entre** w_i et w_{p+i} est égal à la somme du nombre de caractères qu'il y a **avant** w_i et de ceux qu'il y a **après** w_{p+i}.

L'AP suivant reconnait alors le langage L_2 (on note Σ tout symbole de $\{0,1\}$) :



En 6 étapes :

— q_1 : lecture (non déterministe) de n_1 symboles, empile n_1 # sur la pile.
— $q_1 \to q_{2x}$: transition spécifique pour "enregistrer" le symbole $(n_1 + 1)$.
— $q_{2x} + q_{2x} \to q_{3x}$: dépile complètement le contenu de la pile (n_1 fois).
— q_{3x} : lecture (non déterministe) de n_2 symboles, empile n_2 # sur la pile.
— $q_{3x} \to q_4$: transition spécifique différente du symbole enregistré.
— q_4 : dépile complètement le contenu de la pile (donc n_2 fois).

Le non-déterminisme fait le reste !

Comme AP et GLC sont équivalents, et que par fermeture, l'union de deux GLCs est une GLC, $\overline{L}$ est une GLC.

6 Résumé

— un AP est un ANF avec de la mémoire.
— les APs et les GLCs reconnaissent la même classe de langage (= langages libres de contexte)
— la FNC est une transformation d'une GLC rendant les algorithmes sur les GLCs plus facile à mettre en œuvre.
— les GLCs sont fermés par union, opérateur étoile mais pas par intersection ni complémentation.
— le lemme de l'étoile pour les GLCs est une condition nécessaire (mais pas suffisante) permettant de vérifier qu'une grammaire n'est pas libre de contexte.
— il existe des langages non libre de contexte (reconnu par aucun GLC/AP).