Corrigé TD 1
Langages Formels

Luc Lapointe
luc.lapointe@ens-paris-saclay.fr
home.lmf.cnrs.fr/LucLapointe/

Exercice 1 – Dyck à n paires de parenthèses

Soit $\Sigma = \{a_1, \cdots, a_n\} \cup \{b_1, \cdots, b_n\}$ l'alphabet formé de n paires de parenthèses. Un mot est bien parenthésé s'il se réduit au mot vide via les règles,
pour tout $i \in [1, n]$, $a_i b_i \to \varepsilon$.

Donner une grammaire engendrant le langage de Dyck

$$D_n^\star = \{w \in \Sigma^\star \mid w \text{ est bien parenthésé}\}.$$

Solution

Une idée qui fonctionne :

$$S \to SS \mid \varepsilon$$
$$\forall i \in [1, n], S \to a_i S b_i$$

Montrer qu'**un mot généré par cette grammaire est un mot de Dyck** peut se faire par exemple par récurrence sur la taille de la dérivation :
Soit w généré. Si la dérivation est de taille 1, $w = \varepsilon$. Sinon, plusieurs cas :

- Si la dernière règle utilisée est $S \to \mathbb{S}$, alors par hypothèse de récurrence w est une concaténation de deux mots de Dyck, donc en est aussi un, car les réécritures utilisables pour le côté droit seul ou le côté gauche seul peuvent être réutilisées à l'identique sur la concaténation.
- Si la dernière règle utilisée est $S \to a_i S b_i$, alors par hypothèse de récurrence, le mot généré par S peut être réécrit, puis $a_i S b_i$ également.

Montrer qu'**un mot de Dyck est généré par cette grammaire** peut se faire par récurrence sur la longueur de réécriture. Si la réécriture est de longueur 0, alors le mot est ε. Sinon, en appliquant toutes les étapes sauf la dernière, on en déduit que w est de la forme $w_1 a_i w_2 b_i w_3$, où w_1, w_2 et w_3 se réduisent à ε. Par hypothèse de récurrence, w_1, w_2 et w_3 sont donc générés par la grammaire, et on peut générer w avec

$$S \to SS \to SSS \to Sa_i Sb_i S$$

puis en appliquant les dérivées de l'hypothèse de récurrence.

Cette première grammaire est ambigüe. Il n'était pas particulièrement demandé de proposer une grammaire non ambigüe, mais en voici une :

$$S \to \varepsilon$$
$$\forall i \in [1, n], S \to a_i S b_i S$$

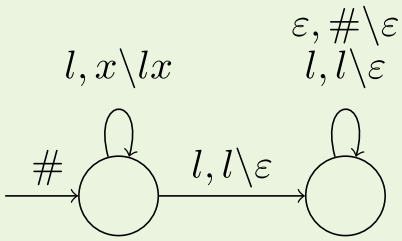
Exercice 2 – Exemples d'automates à pile

1. Construire un automate à pile reconnaissant le langage $L_1 = \{u\tilde{u} : u \in \Sigma^\star\}$.
2. Construire un automate à pile reconnaissant le langage de Dyck D_n^*.
3. Construire un automate à pile reconnaissant le langage $L_2 = \{w \in \Sigma^* : |w|_a = 2|w|_b\}$.
4. Construire un automate à pile reconnaissant par pile vide le langage $L_3 = \{a^n b^p : 1 \leq n \leq p \leq 2n\}$.

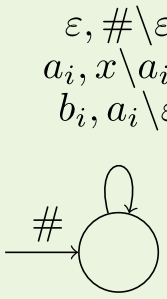
Solution

Les transitions décrites valent pour tout $l \in \Sigma, x \in Z \cup \Sigma$ et pour tout $i \leq n$. Tous les automates décrits acceptent par pile vide.

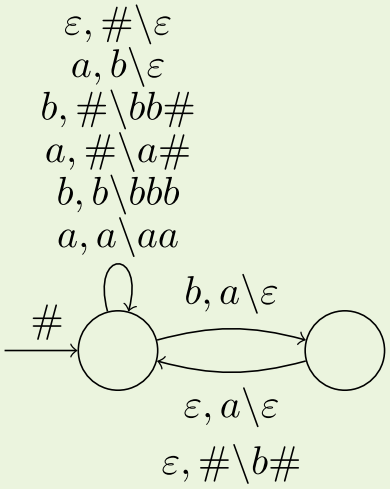
1.



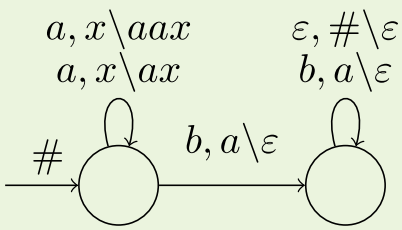
2.



3. La pile représente l'excédent d'une lettre, qui doit être compensé par la lecture de la lettre opposée. Il n'y a jamais simultanément de a et de b dans la pile.



4.



Exercice 3 – Langage linéaire et automates à pic

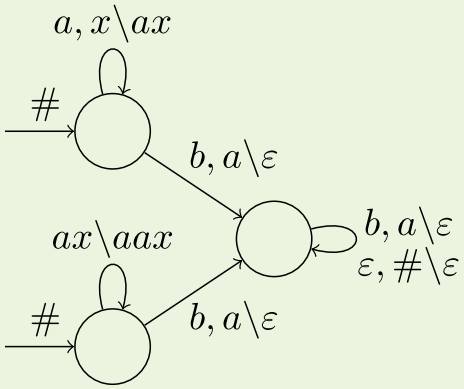
Un automate à un pic est un automate à pile tel que dans tout calcul valide, la taille de la pile n'augmente plus une fois qu'elle a diminué. La taille de la pile peut donc augmenter (au sens large) pendant une première partie du calcul, puis elle ne fait que diminuer (au sens large). Un langage est à un pic s'il peut être accepté par pile vide par un automate à un pic.

1. Montrer que le langage $L = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$ est un langage à un pic.
2. Montrer que le langage $K = \{ba^{i_1}ba^{i_2}b \cdots ba^{i_n}b | n \geq 1 \text{ et } \exists j, i_j \neq j\}$ est un langage à un pic.
3. Montrer que tout langage linéaire est un langage à un pic.

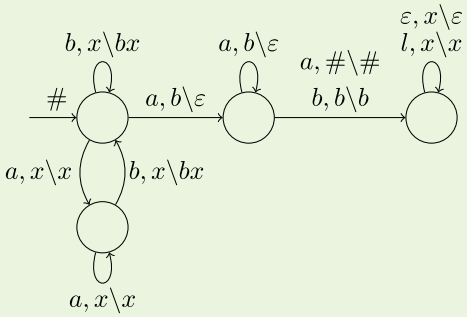
Idée de solution

Les transitions décrites valent pour tout $l \in \Sigma$, et pour tout $x \in Z$.

1. L'automate commence par choisir de manière non-déterministe s'il devra avoir autant, ou deux fois plus, de b que de a.



2. L'automate commence par avancer jusqu'au b qui l'intéresse en ignorant les a et comptant les b, puis fait le traitement pour tester si $i_j \neq j$, et vider la pile si c'est le cas.



3. L'idée est d'utiliser le non-déterminisme de l'automate pour choisir quelle règle appliquer à chaque fois. Les états de l'automate sont les symboles non-terminaux de l'alphabet, plus des états « étapes » qui serviront à faire des chemins entre les états « non-terminaux ». Pour chaque règle

$$A \to w_1 B w_2$$

on ajoute à l'automate des transitions de A vers B qui passent par $|w_1|$ états étape pour lire w_1, puis empilent w_2 sur la pile.

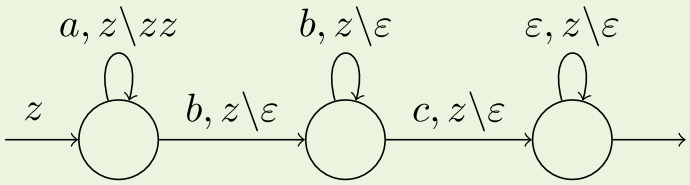
On ajoute aussi à l'automate un état « zéro symbole non terminal », duquel il doit s'assurer que les lettres restantes à lire dans le mot correspondent à ce qu'il a dans sa pile.

Exercice 4 – Unique symbole de pile

On s'intéresse ici à des automates dont l'alphabet de pile Γ est un singleton $\{z\}$.

1. Montrer que le langage $L = \{a^n b^m c : 1 \leq m \leq n\}$ peut être accepté par pile vide et état final par un automate dont l'alphabet de pile est un singleton.
2. Montrer que le langage L ne peut pas être accepté par pile vide par un automate dont l'alphabet de pile est un singleton.

Idée de solution

1. 

2. L'idée est de remarquer qu'il faut dépiler en lisant des b, mais que ça implique d'accepter notamment des mots qui ne terminent pas forcément par un c.

   Supposons par l'absurde qu'il existe un automate acceptant par pile vide. Regardons la famille de mots $a^i b^i c$, et pour chacun de ces mots, considérons un calcul acceptant. On note c_i la configuration atteinte par ce calcul en lisant a^i. S'il existe $i < j$ tels que $c_i = c_j$, alors $a^i b^j c$ est accepté. Donc tous les c_i sont différents. Il existe en particulier une famille I d'indices tels que toutes les configurations c_i dont l'indice est dans I arrivent à un même état e : elles sont de la forme $c_0 \xrightarrow{a^i} ez^{k_i}$ avec k_i qui peut être arbitrairement grand.

   Considérons un tel i grand. Découpons le chemin acceptant, en exhibant le premier cycle qui fait décroître la taille de la pile (un tel cycle existe si i est grand).

   $$c_0 \xrightarrow{w_1}{}^* qz^{k_1+k_2} \xrightarrow{w_2}{}^* qz^{k_1} \xrightarrow{w_3}{}^* q'\varepsilon$$

   Si w_2 est non-vide et ne contient pas de c, on peut obtenir une contradiction. Idem si w_2 contient un c. Donc w_2 doit être vide.

   Comme w_2 est vide, il existe un préfixe de w_1 qui est accepté. *« Préfixe de w_1 » plutôt que « w_1 » car même s'il n'y a pas de cycle qui réduit la taille de*

la pile dans w_1, il peut y avoir un chemin qui réduit un peu la taille de la pile. Donc w_3 doit également être vide.

En prenant un i assez grand, on a dans w_1 une boucle qui lit des b sans réduire la taille de la pile. À partir de là on peut obtenir une contradiction.

Contrôle continu – Exemples d'automates à pile

Construire un automate à pile reconnaissant les langages suivants. Inutile de donner de preuve, une phrase par état pour expliquer leur rôle dans l'automate suffit.

1. $$L_1 = \{a^i b^j c^k : i + j = k\}$$
2. $$L_2 = \{a^i b^j c^k : i + k = j\}$$
3. Le langage des palindromes $\{u \in \Sigma^\star : \tilde{u} = u\}$ où $\tilde{u}$ est l'image miroir de u.