I Sujet 0 CCP MPI

1. Rappeler la définition d'un langage régulier.
2. Les langages suivants sont-ils réguliers? Justifier.

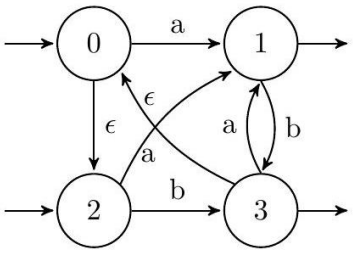
(a) $L_1 = \{a^n b a^m \mid n, m \in \mathbb{N}\}$

(b) $L_2 = \{a^n b a^m \mid n, m \in \mathbb{N}, n \leq m\}$

(c) $L_3 = \{a^n b a^m \mid n, m \in \mathbb{N}, n > m\}$

(d) $L_4 = \{a^n b a^m \mid n, m \in \mathbb{N}, n + m \equiv 0 \bmod 2\}$

3. On considère l'automate non déterministe suivant :



(a) Déterminiser cet automate.

(b) Construire une expression régulière dénotant le langage reconnu par cet automate.

(c) Décrire simplement avec des mots le langage reconnu par cet automate.

II Extrait Centrale 2022 : Palindromes et rationalité

Soit $w \in \Sigma^\star$. On dit que le mot w est un palindrome si $\tilde{w} = w$.

1. Écrire une fonction palindrome de signature `string -> bool` qui teste, en temps linéaire, si un mot est un palindrome.

Pour un alphabet Σ, on note $\mathrm{Pal}(\Sigma)$ l'ensemble des palindromes de $\Sigma^\star$.

2. Montrer que si Σ est un alphabet à une lettre, alors $\mathrm{Pal}(\Sigma)$ est rationnel.
3. Montrer que si Σ contient au moins deux lettres, alors $\mathrm{Pal}(\Sigma)$ n'est pas rationnel.
   On pourra utiliser un automate et un mot de $\mathrm{Pal}(\Sigma) \cap a^\star b a^\star$.

Soit $L \subset \Sigma^\star$ un langage reconnu par l'automate $A = (Q, I, F, T)$.

Pour $(q, q') \in Q^2$, on note $L_{q,q'}$ le langage de tous les mots w qui étiquettent un chemin dans A partant de q et arrivant en q'.

4. Montrer que $L_{q,q'}$ est reconnaissable et exprimer le langage L_A en fonction de langages $L_{q,q'}$.
5. Montrer que $\mathrm{Pal}(\Sigma) \cap \left(\Sigma^2\right)^\star = \{u\tilde{u} \mid u \in \Sigma^\star\}$.

Soit L un langage rationnel reconnu par un automate $A = (Q, I, F, T)$. On définit les langages $D(L) = \{w\tilde{w} \mid w \in L\}$ et $R(L) = \{w \in \Sigma^\star \mid w\tilde{w} \in L\}$.

6. Décrire simplement les langages $D\left(a^\star b\right)$ et $R\left(a^\star b^\star a^\star\right)$.
7. Les langages $D(L)$ et $R(L)$ sont-ils reconnaissables? On pourra faire intervenir les langages $L_{q,q'}$, définis ci-dessus.

A12 – Plus grands facteurs

On fixe l'alphabet $\Sigma := \{a, b\}$. Pour un langage L sur Σ, étant donné un mot $w \in \Sigma^*$, on veut calculer la longueur du plus grand facteur de w dans le langage L, notée $\mathrm{lpgf}(L, w)$; on convient que cette longueur est de -1 si aucun facteur de w n'est dans L. On a donc $-1 \leq \mathrm{lpgf}(L, w) \leq |w|$ où $|w|$ dénote la longueur du mot w.

Question 0. On considère le langage L_0 défini par l'expression rationnelle $(ab)^*$, et le mot $w_0 := baabababa$. Calculer $\mathrm{lpgf}(L_0, w_0)$.

Question 1. Étant donné un langage L, représenté comme un automate fini déterministe complet, et un mot $w \in \Sigma^*$, proposer un algorithme naïf pour calculer $\mathrm{lpgf}(L, w)$. Donner un pseudo-code pour cet algorithme, et déterminer sa complexité en temps et en espace.

Question 2. Proposer un algorithme plus efficace pour ce problème qui s'exécute en temps et en espace $O(|Q| \times |w|)$. Est-il important que l'automate soit déterministe ?

Question 3. On dit que $w' \in \Sigma^*$ est un *sous-mot* de w s'il existe une fonction strictement croissante ϕ de $\{1, \ldots, |w'|\}$ dans $\{1, \ldots, |w|\}$ telle que $w'_i = w_{\phi(i)}$ pour tout $1 \leq i \leq |w'|$, où w'_i dénote la i-ème lettre de w' et de même pour w.

Étant donné un langage L représenté comme un automate et un mot w, comment calculer la longueur du plus grand *sous-mot* de w dans le langage L ?

Question 4. On pose le langage $L_4 := \{a^n b^n \mid n \in \mathbb{N}\}$. Proposer un algorithme efficace qui calcule $\mathrm{lpgf}(L_4, w)$ pour un mot d'entrée w.

Question 5. On pose le langage $L_5 := \{uu \mid u \in \Sigma^*\}$. Proposer un algorithme qui calcule $\mathrm{lpgf}(L_5, w)$ pour un mot d'entrée w en temps $O(|w|^2)$.

Question 6. On pose le langage L_6 des mots bien parenthésés sur Σ défini inductivement comme suit : le mot vide ϵ est bien parenthésé, la concaténation de deux mots bien parenthésés est bien parenthésée, et si $w \in \Sigma^*$ est bien parenthésé alors awb l'est aussi. Proposer un algorithme efficace qui calcule $\mathrm{lpgf}(L_6, w)$ pour un mot d'entrée w.

Corrigé

Question 0. Il est clair que $(ab)^3$ est un facteur de w_0 mais que $(ab)^4$ ne l'est pas, donc $\mathrm{lpgf}(L_0, w_0)$ est 6.

Question 1. On a d'abord l'algorithme naïf de complexité en temps cubique (et de complexité en espace constante) :

```
T := tableau de taille n contenant les lettres du mot (de 0 à n-1)
lpgf := -1

// i inclus, j exclu, pour également tester le mot vide
Pour i de 0 à n-1:
  Pour j de i à n:
    q := q_0 // état initial de l'automate
    Pour k de i à j-1:
      q := delta(T[k], q) // fonction de transition de l'automate
    Fin Pour
    Si F[q]: // états finaux de l'automate
      lpgf := max(lpgf, j-i)
    Fin Si
  Fin Pour
Fin Pour

Renvoie lpgf
```

On peut rendre l'algorithme quadratique (et conserver la complexité constante en espace) en changeant la boucle comme suit :

```
...
Pour i de 0 à n-1:
  q := q_0 // état initial de l'automate
  Si F[q]:
    lpgf := max(lpgf, 0)
  Fin Si
  Pour j de i à n-1:
    q =: delta(T[k], q) // fonction de transition de l'automate
    Si F[q]: // états finaux de l'automate
      lpgf := max(lpgf, j-i+1)
    Fin Si
  Fin Pour
Fin Pour
...
```

Question 2. On considère un automate non-déterministe $A = (Q, I, F, \delta)$, et on construit son graphe produit avec le mot w, c'est-à-dire le graphe ayant pour sommets $\{(q, i) \mid q \in Q, i \in \{0, \ldots, |w|\}\}$ et ayant, pour chaque $0 \leq i < |w|$ et chaque transition (q, a, q') où a est la lettre à la position $i + 1$ de w, une arête de (i, q) à $(i + 1, q')$. Un sommet est dit *initial* si sa seconde composante est dans I, et *final* si sa seconde composante est dans F. Il est clair que, pour tous $q, q' \in Q$ et $0 \leq i \leq j \leq |w|$, il existe un chemin de (q, i) à (q', j) dans le graphe si et seulement s'il existe un chemin dans A de q à q' étiqueté par $w[i + 1 \cdots j]$. Ainsi, il y a un facteur de longueur k accepté par l'automate ss'il y a un chemin de

longueur k d'un sommet initial à un sommet final. Il suffit donc de calculer le plus long chemin dans ce graphe acyclique, ce que l'on peut faire facilement par programmation dynamique, et on en déduit la longueur du plus long facteur accepté par l'automate.

```
T := tableau de taille n contenant les lettres du mot
D := tableau d'entiers de taille (n+1) fois |Q|
lpgf := -1

Pour i de 0 à n:
  Pour q de 0 à |Q|-1:
    Si F[q]:
      D[i][q] := 0
    Sinon:
      D[i][q] := -1
    Fin Si
  Fin Pour
Fin Pour

Pour i de (n-1) à 0:
  Pour q de 0 à |Q|-1:
    Pour chaque transition (q, T[i], q') partant de q:
      Si D[i+1][q'] >= 0:
        D[i][q] := max(D[i][q], 1+D[i+1][q'])
      Fin Si
    Fin Pour
    Si I[q]: // q est initial
      lpgf := max(lpgf, D[i][q])
  Fin Pour
Fin Pour

Renvoie lpgf
```

Pour un automate déterministe, la complexité en temps et en mémoire est de $O(|Q| \times |w|)$ comme annoncé. Pour un automate non-déterministe, la complexité en temps passe à $O(|\delta| \times |w|)$ (en supposant que chaque état de Q apparaît dans δ, puisque si tel n'est pas le cas on peut supprimer les autres états sans perte de généralité). On peut aboutir à une meilleure complexité en espace, i.e., $O(|Q|)$, en ne conservant le tableau $D[i]$ que pour la valeur courante et la valeur précédente de i.

Question 3. On procède comme en question 2 mais on s'autorise à sauter des lettres. Concrètement, quand on calcule `L[i][q]`, on ajoute l'instruction :

```
...
L[i][q] := max(L[i][q], L[i+1][q])
...
```

Question 4. On peut écrire w en regroupant les a contigus et les b contigus, i.e., $w = a^{p_1}b^{q_1} \cdots a^{p_k}b^{q_k}$, où p_1 et q_k sont éventuellement nuls. Il est clair qu'un facteur de w de la forme $a^n b^n$ pour un certain $n \in \mathbb{N}$ doit être formé d'un suffixe (non nécessairement strict) d'un groupe de a dans cette décomposition, et d'un préfixe (non nécessairement strict) du groupe suivant de b dans cette décomposition : et ainsi $\mathrm{lpgf}(L_4, w) = \max_{1 \leq i \leq k} \min(p_i, q_i)$.

En pseudo-code :

```
T := tableau de taille n contenant les lettres du mot
p := 0
lpgf := -1

Tant que p < n:
  n_a := 0
  n_b := 0
  Tant que p < n et T[p] == 'a':
    n_a++
    p++
  Fin Tant que
  Tant que p < n et T[p] == 'b':
    n_b++
    p++
  Fin Tant que
  lpgf := max(lpgf, min(n_a, n_b))
Fin Tant que

Renvoie lpgf
```

Question 5. L'algorithme naïf en temps cubique est de tester chaque facteur (de taille paire), et vérifier si c'est un carré : on vérifie si le facteur entre i inclus et j exclu est un carré en lisant simultanément les $T[i+k]$ et les $T[i+(j-i)/2+k]$ pour k de 0 inclus à $(j-i)/2$ exclu et en vérifiant que c'est le même mot.

Pour le faire en temps quadratique, on peut plus astucieusement tester chaque décalage possible (la moitié de la longueur du facteur), et lire avec ce décalage pour voir si un facteur convient. En pseudo-code :

```
T := tableau de taille n contenant les lettres du mot
lpgf := 0 // le mot vide est toujours un facteur

Pour d de 1 à \lfloor n/2 \rfloor:
  n_ok := 0
  Pour i de 0 à n-d-1:
    Si T[i] == T[i+d]:
      n_ok++
    Sinon:
      n_ok := 0
    Fin Si
    Si n_ok == d:
      // T[i-d+1..i] == T[i+1..i+d] (bornes incluses)
      // donc on a trouvé un facteur carré de longueur 2d
      lpgf := 2d
      Sortir de la boucle Pour interne // optimisation
    Fin Si
  Fin Pour
Fin Pour

Renvoie lpgf
```

[Il existe un algorithme plus sophistiqué pour résoudre ce problème en temps linéaire [GS04].]

Question 6. On remarque d'abord qu'on peut tester en temps linéaire si un mot est bien parenthésé en vérifiant que chaque préfixe contient au moins autant de a que de b, et que le mot au total contient autant de a que de b. En pseudo-code :

```
T := tableau de taille n contenant les lettres du mot
d := 0

Pour i de 0 à n-1:
  Si T[i] == 'a':
    d++
  Sinon // T[i] == 'b'
    d--
  Fin Si
  Si d < 0:
    Renvoie faux
  Fin Si
Fin Pour

Renvoie (d == 0)
```

L'algorithme très naïf en $O(n^3)$ est donc de tester tous les facteurs, et de tester en temps linéaire pour chaque facteur s'il est bien parenthésé.

L'algorithme un peu moins naïf en temps $O(n^2)$, comme en question 1, est de tester toutes les positions de départ :

```
T := tableau de taille n contenant les lettres du mot
lpgf := 0 // le mot vide est toujours un facteur

Pour i de 0 à n-1:
  d := 0
  Pour j de i à n-1:
    Si T[j] == 'a':
      d++
    Sinon: // T[j] == 'b'
      d--
    Fin Si
    Si d == 0:
      lpgf := max(lpgf, j-i+1)
    Fin Si
    Si d < 0:
      Sortir de la boucle Pour interne
    Fin Si
  Fin Pour
Fin Pour

Renvoie lpgf
```

Pour un algorithme en temps linéaire, l'idée générale est de parcourir le mot une seule fois avec une pile où on conserve les parenthèses ouvrantes (a) et leur position : ainsi, quand on lit une parenthèse fermante (b), on peut mettre à jour lpgf en considérant le plus long facteur bien parenthésé qui se finit à cet endroit. Le piège est que, sur un mot comme *abab*, à la lecture du dernier b, le plus long facteur

n'est pas celui qui commence à la parenthèse ouvrante correspondante (i.e., *ab*), mais c'est *abab*. Ainsi, plutôt que de mettre les parenthèses fermantes en correspondante avec leur parenthèse ouvrante, on va les mettre en correspondance avec la position de la parenthèse ouvrante *du niveau précédent* (si elle existe, auquel cas le facteur le plus long commence une position après), ou, s'il n y en a pas, avec une parenthèse fermante qui n'a pas de correspondance dans ce qui précède.

Formellement, on va dire qu'une parenthèse fermante dans le mot w est *abyssale* si elle n'est pas en correspondance avec une parenthèse ouvrante dans les positions précédentes. On va considérer qu'à la position -1 il y a une parenthèse fermante abyssale qui va servir de sentinelle ; ceci ne change clairement pas la valeur du lpgf. Maintenant, l'observation clé est que, pour toute parenthèse fermante qui n'est pas abyssale, le facteur bien parenthésé le plus long qui lui correspond commence juste après la parenthèse ouvrante du niveau inférieur qui est dans la pile à ce moment-là, ou (s'il n'y en a pas) commence juste après la dernière parenthèse fermante abyssale que l'on a vu. La raison pour cela est que, si on revient en arrière à partir de cette parenthèse fermante, on va trouver sa parenthèse ouvrante correspondante, et puis il y a trois cas : (i) la position d'avant est une parenthèse ouvrante auquel cas le plus long facteur commence juste après ; (ii) la position d'avant est une parenthèse fermante abyssale (y compris la parenthèse fermante abyssale sentinelle à la position -1) auquel cas le plus long facteur commence aussi juste après ; (iii) la position d'avant est une parenthèse fermante non-abyssale, auquel cas elle a une parenthèse ouvrante correspondante jusqu'à laquelle on peut remonter pour répéter l'argument.

On en déduit donc l'invariant à garantir : à tout moment de la lecture de w, quand on a fini de lire un préfixe, la pile doit contenir les parenthèses ouvrantes actuellement ouvertes et leur position (dans l'ordre), et au fond de la pile doit se trouver la dernière parenthèse fermante abyssale que l'on a vue et sa position (initialement il s'agit de la sentinelle à la position -1). Quand on lit une parenthèse ouvrante, on l'empile simplement et l'invariant est préservé. Quand on lit une parenthèse fermante, soit la pile ne contient que le fond, et auquel cas on le remplace par la parenthèse fermante courante (car elle est abyssale) ; soit elle contient autre chose, auquel cas il suffit de pop la parenthèse ouvrante au sommet car elle vient de se finir. La valeur du lpgf est mise à jour quand on lit une parenthèse fermante non abyssale, et elle vaut la distance entre la position courante et la position du sommet de pile (c'est soit la dernière parenthèse abyssale vue, soit la position de la dernière parenthèse ouvrante qui n'a pas encore été fermée, et on a expliqué au paragraphe précédent pourquoi le facteur bien parenthésé maximal qui se termine à la parenthèse fermante courante doit commencer juste après cette position).

On parvient donc au code suivant :

```
T := tableau de taille n contenant les lettres du mot
P := pile vide
lpgf := 0 // le mot vide est toujours un facteur
P.push(-1) // sentinelle

Pour i de 1 à n:
  Si T[i] == 'a':
    P.push(i)
  Sinon: // T[i] == 'b'
    P.pop()
    Si P n'est pas vide:
      lpgf := max(lpgf, i - P.top())
    Sinon:
      P.push(i)
    Fin Si
  Fin Si
Fin Pour

Renvoie lpgf
```

III Automate à pile

Soit Σ un alphabet fini, et Γ un autre alphabet fini appelé alphabet de pile. Un automate à pile sur Σ est un quintuplet $A = (Q, q_0, \gamma_0, \delta, F)$, où Q est un ensemble fini d'états, $q_0 \in Q$ est l'état initial, $\gamma_0 \in \Gamma$ est le symbole de pile initial, δ est une fonction de transition de $Q \times \Sigma \times \Gamma$ vers l'ensemble des parties de $Q \times \Gamma^*$, et $F \subseteq Q$ est un ensemble d'états finaux.

Une configuration de A est un couple (q, z) où $q \in Q$ est l'état et où z est un mot non-vide sur Γ appelé pile. La configuration initiale est (q_0, γ_0), et une configuration (q, z) est acceptante si $q \in F$. Lorsque l'automate est dans une configuration (q, z) et lit une lettre $a \in \Sigma$, il décompose $z = z'\gamma$ avec $\gamma \in \Gamma$ le sommet de pile, il choisit un $(q', g) \in \delta(q, a, \gamma)$ tel que $z'g$ soit non-vide, et il aboutit à la configuration $(q', z'g)$. L'automate à pile A accepte un mot de Σ^* s'il peut lire ses lettres dans l'ordre à partir de la configuration initiale pour parvenir à une configuration acceptante suivant ces règles.

Question 0. Étant donné un automate A sur Σ sans pile, expliquer comment construire un automate à pile A' sur Σ qui reconnaît le même langage que A.

Question 1. On prend dans cette question $\Sigma = \{a, b\}$. Proposer un automate à pile qui reconnait le langage $\{a^n b^n \mid n \geq 2\}$. Qu'en déduire?

Question 2. On prend toujours $\Sigma = \{a, b\}$. Proposer un automate à pile qui reconnait le langage $\{w \in \Sigma^* || w|_a = |w|_b\}$, où $|w|_a$ et $|w|_b$ désignent respectivement le nombre de a et de b de w.

Question 3. Pour $\eta \in \mathbb{N}^*$, étant donné un mot w de longueur n et un automate à pile A, les configurations η-tronquées sont les triplets (q, z) où $q \in Q$ et z est un mot non-vide sur Γ de longueur $\leq \eta$. Un calcul η-tronqué de A sur un mot est une séquence de configurations η-tronquées : la définition est comme auparavant sauf que, si $|z'g| \geq \eta$, on le remplace par son suffixe de longueur η. Le langage η-tronqué $L_{\leq\eta}(A)$ de A est le langage des mots sur lesquels A a un calcul η-tronqué acceptant.

Quelle est la relation entre $L_{\leq\eta}(A)$ et $L(A)$? Montrer que, pour tout automate à pile A et $\eta \in \mathbb{N}^*$, le langage η-tronqué de A est un langage régulier, et expliquer comment construire un automate sans pile qui le reconnaît.

Question 4. Soit w un mot de longueur n et A un automate à pile. Soit $\chi = (q_0, z_0), \ldots, (q_p, z_p)$ un calcul de A sur w, et notons $h_i := |z_i|$ pour tout $0 \leq i \leq p$. Pour $\eta \in \mathbb{N}^*$, une η-montagne de χ est un triplet d'indices $0 \leq l < m < r \leq p$ tels qu'on ait $h_l = h_r$, on ait $h_m - h_l \geq \eta$, et on ait $h_j > h_l$ pour tout $l < j < r$. Une montagne est une η-montagne pour un certain $\eta \in \mathbb{N}^*$.

On appelle G la plus grande longueur d'un mot de Γ^* dans une image de la fonction de transition δ. Montrer que, pour tout $\eta > G$, si $w \in L(A) \backslash L_{\leq\eta}(A)$, alors tout calcul acceptant de A sur w a une $(\eta - G)$-montagne.

Question 5. L'état de base d'une η-montagne $l < r$ dans un calcul est le couple $\beta(l, r) := (q_l, q_r, \gamma)$ où q_l, q_r sont les états des étapes l et r du calcul respectivement, et γ est le dernier symbole de la pile z_l.

Montrer que, pour tout automate à pile A, il existe $\eta_0 \in \mathbb{N}^*$ avec la propriété suivante : pour tout mot w, pour tout calcul χ de A sur w, si χ a une η_0-montagne (l, m, r), alors il existe une montagne (l', m, r') et une montagne (l'', m, r'') avec $l < l' < l'' < m < r'' < r' < r$ telles que $\beta(l', r') = \beta(l'', r'')$.

Question 6. Déduire des trois questions précédentes une forme affaiblie du lemme de pompage pour les automates à pile : pour tout langage L reconnu par un automate à pile, il existe un entier $p \in \mathbb{N}$ tel que, pour tout mot $w \in L$ avec $|w| > p$, on peut écrire $w = uvxyz$ où les mots u, v, x, y, z satisfont:

$-|vy| \geq 1$

- Pour tout $n \in \mathbb{N}$, on a $uv^n xy^n z \in L$

Suite des questions

Question 3. Pour $\eta \in \mathbb{N}^*$, étant donné un mot w de longueur n et un automate à pile A, les *configurations η-tronquées* sont les triplets (q, z) où $q \in Q$ et z est un mot non-vide sur Γ de longueur $\leq \eta$. Un *calcul η-tronqué* de A sur un mot est une séquence de configurations η-tronquées : la définition est comme auparavant sauf que, si $|z'g| \geq \eta$, on le remplace par son suffixe de longueur η. Le *langage η-tronqué* $L_{\leq\eta}(A)$ de A est le langage des mots sur lesquels A a un calcul η-tronqué acceptant.

Quelle est la relation entre $L_{\leq\eta}(A)$ et $L(A)$? Montrer que, pour tout automate à pile A et $\eta \in \mathbb{N}^*$, le langage η-tronqué de A est un langage régulier, et expliquer comment construire un automate sans pile qui le reconnaît.

Question 4. Soit w un mot de longueur n et A un automate à pile. Soit $\chi = (q_0, z_0), \ldots, (q_p, z_p)$ un calcul de A sur w, et notons $h_i := |z_i|$ pour tout $0 \leq i \leq p$. Pour $\eta \in \mathbb{N}^*$, une *η-montagne* de χ est un triplet d'indices $0 \leq l < m < r \leq p$ tels qu'on ait $h_l = h_r$, on ait $h_m - h_l \geq \eta$, et on ait $h_j > h_l$ pour tout $l < j < r$. Une *montagne* est une η-montagne pour un certain $\eta \in \mathbb{N}^*$.

On appelle G la plus grande longueur d'un mot de Γ^* dans une image de la fonction de transition δ. Montrer que, pour tout $\eta > G$, si $w \in L(A) \setminus L_{\leq\eta}(A)$, alors tout calcul acceptant de A sur w a une $(\eta - G)$-montagne.

Question 5. L'*état de base* d'une η-montagne $l < r$ dans un calcul est le triplet $\beta(l, r) := (q_l, q_r, \gamma)$ où q_l, q_r sont les états des étapes l et r du calcul respectivement, et γ est le dernier symbole de la pile z_l.

Montrer que, pour tout automate à pile A, il existe $\eta_0 \in \mathbb{N}^*$ avec la propriété suivante : pour tout mot w, pour tout calcul χ de A sur w, si χ a une η_0-montagne (l, m, r), alors il existe une montagne (l', m, r') et une montagne (l'', m, r'') avec $l < l' < l'' < m < r'' < r' < r$ telles que $\beta(l', r') = \beta(l'', r'')$.

Question 6. Déduire des trois questions précédentes une forme affaiblie du *lemme de pompage pour les automates à pile* : pour tout langage L reconnu par un automate à pile, il existe un entier $p \in \mathbb{N}$ tel que, pour tout mot $w \in L$ avec $|w| > p$, on peut écrire $w = uvxyz$ où les mots u, v, x, y, z satisfont :

— $|vy| \geq 1$

— Pour tout $n \in \mathbb{N}$, on a $uv^n xy^n z \in L$.

Corrigé

Question 0. On choisit un alphabet de pile Γ arbitraire, et $\gamma_0 \in \Gamma$ arbitraire. Soit $A = (Q, q_0, F, \delta)$ un automate sans pile sur Σ, non nécessairement complet ou déterministe, où $\delta : Q \times \Sigma \to 2^Q$. On définit l'automate $A' = (Q, q_0, \gamma_0, \delta', F)$, où on définit $\delta(q, a, \gamma) := \{(q', \gamma_0) \mid q' \in \delta(q)\}$ pour tous $q \in Q$, $a \in \Sigma$, et $\gamma \in \Gamma$.

Observons d'abord que, dans toutes les configurations accessibles pour l'automate A', la *pile* est exactement γ_0. En effet, à chaque transition, on la laisse à γ_0. De plus, la valeur du sommet de pile n'a jamais d'impact sur les transitions. Ainsi, les séquences de configurations possibles de A' sur n'importe quel mot $w \in \Sigma^*$ correspondent exactement aux séquences d'états possibles de A sur w en ajoutant une pile qui vaut toujours γ_0. En particulier, la configuration initiale correspond à l'état initial, les configurations acceptantes correspondent aux états finaux, et les transitions entre configurations correspondent aux transitions de A. Ainsi, A et A' reconnaissent le même langage.

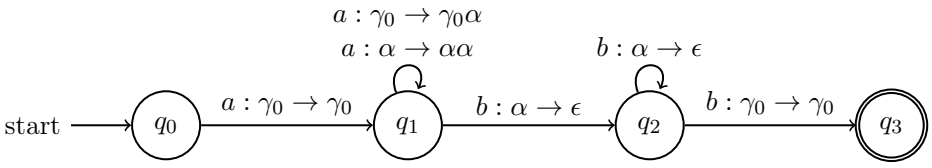
Question 1. On choisit l'alphabet de pile $\Gamma = \{\alpha, \gamma_0\}$. On définit l'automate $A = (\{q_0, q_1, q_2, q_3\}, q_0, \gamma_0, \delta, \{q_3\})$ avec δ comme suit (c'est $\emptyset$ pour les cas non spécifiés) :

— $\delta(q_0, a, \gamma_0) := \{(q_1, \gamma_0)\}$

— $\delta(q_1, a, \gamma) := \{(q_1, \gamma\alpha)\}$ pour $\gamma \in \{\gamma_0, \alpha\}$;

— $\delta(q_1, b, \alpha) := \{(q_2, \epsilon)\}$

— $\delta(q_2, b, \alpha) := \{(q_2, \epsilon)\}$;
— $\delta(q_2, b, \gamma_0) := \{(q_3, \gamma_0)\}$.
Graphiquement :



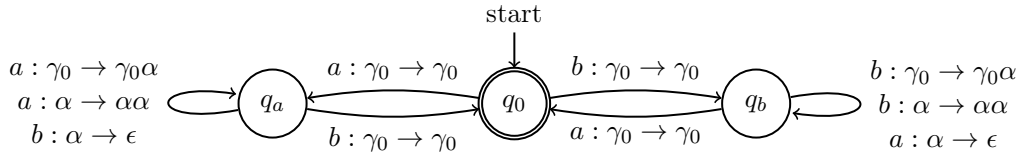
Intuitivement, l'automate passe dans l'état 1 quand il voit un a ; il reste dans l'état 1 tant qu'il lit des a et empile un α par a lu. Lorsqu'il lit des b, il passe dans l'état 1 et dépile un α par b lu. Il atteint l'état final lorsque le fond de la pile γ_0 est atteint en lisant le dernier b, auquel cas il accepte précisément si c'est la fin du mot.

Formellement, montrons que, pour tout mot $w = aa^n bb^n$ avec $n \in \mathbb{N}^*$, l'automate accepte w. L'automate suit la séquence de configurations $(q_0, \gamma_0), (q_1, \gamma_0), (q_1, \gamma_0\alpha), \ldots, (q_1, \gamma_0\alpha^n), (q_2, \gamma_0\alpha^{n-1}), \ldots, (q_2, \gamma_0\alpha), (q_2, \gamma_0), (q_3, \gamma_0)$, et il accepte car q_3 est final.

À l'inverse, considérons un chemin de configurations, et montrons que le mot reconnu est de la forme $aa^n bb^n$ avec $n \in \mathbb{N}^*$. On commence à la configuration (q_0, γ_0), et on lit nécessairement un a pour parvenir à la configuration (q_1, γ_0). On lit alors un certain nombre de a, notons n ce nombre, et on parvient à la configuration $(q_1, \gamma_0\alpha^n)$. Il faut ensuite lire un b pour parvenir à la configuration $(q_1, \gamma_0\alpha^{n-1})$ ce qui impose que $n > 0$. Ensuite, pour parvenir à l'état final q_3, il faut que le sommet de pile soit γ_0, ce qui impose de lire $n - 1$ fois un b pour parvenir à la configuration (q_2, γ_0). Ensuite, il faut lire un autre b pour effectuer la transition et parvenir à (q_3, γ_0). On a alors lu $n + 1$ fois a, suivi de $1 + (n - 1) + 1$ fois b, avec $n > 0$, donc un mot de la forme attendue. Le mot se termine forcément à ce moment, car il n'y a plus de transition sortante. Ainsi, tout mot accepté par l'automate est de la bonne forme, ce qui conclut la preuve.

On en déduit qu'il existe des langages non rationnels reconnus par un automate à pile, car le langage $\{a^n b^n \mid n \geq 2\}$ n'est pas rationnel : ceci se montre sans difficulté par le lemme de l'étoile.

Question 2. On construit l'automate suivant sur l'alphabet de pile $\Gamma = \{\alpha, \gamma_0\}$:



Il est clair que tout mot avec autant de a que de b est accepté. En effet, considérons la fonction δ qui à toute position $0 \leq i \leq n$ d'un mot d'entrée w de longueur n associe $|w'|_a - |w'|_b$ pour w' le préfixe de longueur i du mot. On montre facilement par induction sur la position que, pour tout $0 \leq i \leq n$, si l'on considère la configuration où on se trouve après avoir lu le préfixe w' de longueur i de w :
— on a $\delta(i) = 0$ ssi la configuration est (q_0, γ_0)
— on a $\delta(i) > 0$ ssi la configuration est $(q_a, \gamma_0\alpha^{|\delta(i)|-1})$
— on a $\delta(i) < 0$ ssi la configuration est $(q_b, \gamma_0\alpha^{|\delta(i)|-1})$
Ainsi, le mot est accepté si et seulement si $\delta(|w|) = 0$, c'est-à-dire si et seulement s'il est dans le langage.

Question 3. Il est clair que $L_{\leq\eta}(A) \subseteq L(A)$ pour tout η, car tout calcul η-tronqué permet d'obtenir un calcul : la seule différence est qu'un calcul η-tronqué peut se bloquer parce que la pile est devenue vide, alors qu'elle n'aurait pas été vide dans un vrai calcul (i.e., on a dépilé plus profond que η).

Pour la seconde partie de la question, on construit simplement un automate fini sur les configurations η-tronquées : chaque configuration η-tronquée correspond à un état. Le point clé est que les configurations η-tronquées sont en nombre fini, donc on obtient bien un automate fini.

Question 4. Fixons w un mot de longueur n, l'automate à pile A, et $\eta \in \mathbb{N}^*$. Supposons que $w \in L(A) \setminus L_{\leq\eta}(A)$, et considérons un calcul acceptant $\chi = (q_0, z_0), \ldots, (q_p, z_p)$ de A sur w.

On observe d'abord que, clairement, χ est un calcul acceptant η-tronqué à moins qu'il existe un point $0 \leq m \leq p$ et un point $m < r' \leq p$ où $h_m - h_{r'} \geq \eta$. En effet, la seule façon que χ échoue en tant que calcul η-tronqué est qu'on atteigne la pile vide à un moment où la pile n'est pas véritablement vide, ce qui veut dire qu'on avait la pile à une hauteur h_m et qu'on atteint ensuite une hauteur de $h_m - \eta$ ou moins. En fait, vu que l'on ne peut dépiler les symboles que un par un, quitte à prendre $r' > m$ aussi petit que possible, on peut supposer que $h_m - h_{r'} = \eta$, et que r' est la plus petite valeur $> m$ pour laquelle c'est le cas. Ainsi, comme $w \notin L_{\leq\eta}(A)$, on sait qu'il existe un tel témoin $0 \leq m < r \leq p$ tels que $h_m - h_{r'} = \eta$ pour le calcul χ, et pour tout $m \leq j < m$, on a $h_j > h_{r'}$ par minimalité de r'.

Étant donné ce témoin, on cherche à construire une $(\eta - G)$-montagne. Comme la pile est initialement de hauteur 1, et qu'elle croît de G à chaque étape, il est clair qu'il existe $0 \leq l < m$ tel que $h_{r'} \leq h_l \leq h_{r'} + G$. On prend le plus grand $l < m$ tel que h_l ait cette propriété. On diminue ensuite r' en r de sorte à garantir que $h_r = h_l$ (ce qui assure par l'encadrement de h_l que $r' - r \leq G$) : ceci est possible car, vu que la taille de pile décroît de h_m à $h_{r'}$ et qu'on dépile au plus un symbole à chaque fois, toutes les valeurs intermédiaires sont forcément atteintes au cours de la descente (au contraire de ce qu'il se passait précédemment pour la montée). On prend le plus petit r avec cette propriété. On a maintenant construit $0 \leq l < m < r \leq p$ tels que $h_l = h_r$ et $h_r \geq \eta - G$. Il reste juste à observer que $h_j > h_l$ pour tout $l < j < r$, mais pour $j \leq m$ c'est une conséquence de la maximalité de l, et pour $m \leq j$ c'est une conséquence de la minimalité de r.

On a donc construit une $(\eta - G)$-montagne (l, m, r).

Question 5. On observe tout d'abord que, par les propriétés d'une montagne, le dernier symbole de la pile z_r est également γ, car ce symbole n'a pu être dépilé entre l et r.

Le nombre d'états de base est de $|Q|^2 \times |\Gamma|$, et on définit η_0 comme $|Q|^2 \times |\Gamma| \times (G+1)$, où G est défini à partir de A comme à la question précédente. Considérons un mot w et un calcul χ de A sur w qui a une η_0-montagne (l, m, r). De z_l à z_m, la pile passe d'une hauteur h_l à h_m ; comme on empile au plus G symboles à chaque étape de calcul, par définition de η_0, il y a au moins $v := 1 + |Q|^2 \times |\Gamma|$ hauteurs entre h_l et h_m qui sont atteintes à un indice entre l et m. Pour chacune de ces hauteurs, comme à la question précédente, il existe une position entre m et r où la même hauteur de pile est atteinte (toutes les hauteurs de pile sont atteintes pendant la descente). Ainsi, à partir de la η_0-montagne (l, m, r), on peut définir v montagnes imbriquées : spécifiquement, on a la séquence $h_l < h'_1 < \cdots < h'_v < h_m$ des hauteurs atteintes, et les indices $l < l'_1 < \cdots < l'_v < m$ où elles le sont, que l'on choisit chacun maximaux parmi les indices $< m$ qui atteignent la hauteur respective. On a ensuite la séquence des indices $m < r'_1 < \cdots < r'_v < r$ tels que $h_{l'_i} = h_{r'_i}$ pour tout $1 \leq i \leq v$, que l'on choisit chacun minimaux parmi les indices $> m$ qui atteignent la hauteur respective (il est clair qu'ils satisfont bien la relation d'ordre indiquée). Pour tout $1 \leq i \leq v$, il est ainsi clair que (l'_i, m, r'_i) est une η_i-montagne pour un certain $\eta_i > 0$: la seule condition à vérifier est celle sur les h_j, qui est vraie comme précédemment par maximalité des l'_i et minimalité des r'_i.

À présent, par définition de v et par le principe des tiroirs, il y a nécessairement deux de ces montagnes qui ont le même état de base, ce qui permet de conclure.

Question 6. On pose $\eta := \eta_0 + G$, pour η_0 comme défini à la question précédente. On sait par la question 3 que $L_{\leq\eta}(A)$ est un langage régulier : on pose p la longueur de pompage classique pour un automate A (sans pile) qui le reconnaît. Prenons un mot $w \in L$ avec $|w| > p$. Si $w \in L_{\leq\eta}(A)$, comme

$|w| > p$, on peut appliquer le pompage pour A et décomposer $w = uvz$ avec $|v| \geq 1$ (correspondant au cycle dans l'automate) et $uv^n z \in L$ pour tout $n \in \mathbb{N}$, et on conclut en posant $x = y = \epsilon$.

Si $w \in L(A) \setminus L_{\leq \eta}(A)$, alors on considère un calcul acceptant de A sur w, on note p sa longueur. D'après la question 4, ce calcul a une η_0-montagne. D'après la question 5, il existe deux montagnes (l', m, r') et (l'', m, r'') avec $l' < l'' < m < r'' < r'$ telles que $\beta(l', r') = \beta(l'', r'')$. On décompose $w = uvxyz$ suivant $0 \leq l' < l'' < r'' < r' \leq p$, ce qui garantit la première condition. Pour la seconde, on observe qu'on peut construire à partir de χ un calcul acceptant pour $uv^n xy^n z$ pour $n \in \mathbb{N}$. L'idée est que l'on peut reproduire les étapes du calcul correspondant à v et y : pour v, on part et on arrive au même état q_l avec le même sommet de pile qu'on peut lire mais qu'on ne peut jamais dépiler, et ce faisant on empile un certain mot z ; et pour y, on part et on arrive au même état q_r avec le même sommet de pile γ, en dépilant z sans descendre au-delà.

[La classe des langages reconnus par les automates à pile s'appelle également classe des langages algébriques*, et le résultat démontré à cette question est le pendant pour les automates à pile du théorème de Bar-Hillel, Perles et Shamir, cf [Wik17], avec un petit affaiblissement (on ne montre pas la condition $|vxz| \leq p$). La preuve proposée suit [AJ13].]*

Références

[AJ13] Antoine Amarilli and Marc Jeanmougin. A Proof of the Pumping Lemma for Context-Free Languages Through Pushdown Automata, 2013.

[Wik17] Wikipedia. Pumping lemma for context-free languages, 2017. `https://en.wikipedia.org/wiki/Pumping_lemma_for_context-free_languages`.

IV Inégalité de Kraft

Dans ce qui suit, on se fixe un alphabet fini Σ de cardinalité $K \geq 2$. On rappelle qu'un mot $u \in \Sigma^*$ est préfixe d'un mot $w \in \Sigma^*$ s'il existe $u' \in \Sigma^*$ tel que $uu' = w$. On dit qu'un ensemble de mots $\Gamma \subseteq \Sigma^*$ est sans préfixe si aucun mot u de Γ n'est préfixe d'un autre mot v de $\Gamma (v \neq u)$.

1. Donner un exemple d'ensemble Γ sans préfixe sur l'alphabet $\Sigma = \{a, b, c\}$, contenant au moins 6 mots. Donner maintenant un exemple d'ensemble sans préfixe infini.

2. Soit $\Gamma \subseteq \Sigma^*$ un ensemble de mots sans préfixe ne contenant pas le mot vide. Montrer que Γ possède la propriété de décomposabilité unique : si un mot w peut s'écrire à la fois $w = u_1 \dots u_n$ et $w = v_1 \dots v_m$ avec tous les u_i et v_j dans Γ, alors $n = m$ et $u_i = v_i$ pour tout i.

3. Etant donné un ensemble fini de mots Γ, proposer un algorithme naïf pour tester si Γ est sans préfixe. Quelle est sa complexité?

4. Un arbre K-aire est un arbre dont chaque noeud a au plus K fils. Une feuille de l'arbre est un noeud qui n'a aucun fils. On appelle profondeur d'un noeud sa distance à la racine (cette dernière a donc profondeur 0). Dit autrement, la racine a profondeur 0 et si un noeud a profondeur h, ses fils ont hauteur $h + 1$.

   Soit T un arbre K-aire fini. Montrer que

   $$\sum_{\sigma \text{ feuille de } T} K^{\text{-profondeur } (\sigma)} \leq 1$$

   Dans quel cas a-t-on égalité?

5. En utilisant la question précédente, montrer que si $\Gamma \subseteq \Sigma^*$ est un ensemble fini sans préfixe, alors

   $$\sum_{w \in \Gamma} K^{-|w|} \leq 1$$

   On appelle cette inégalité l'inégalité de Kraft. Dans quel cas a-t-on égalité? Est-ce que l'inégalité de Kraft reste vraie pour les ensembles infinis (sans préfixe)?

6. Proposer maintenant un algorithme en temps linéaire pour tester si un ensemble de mots finis est sans préfixe.

7. On va maintenant donner une preuve algorithmique de la réciproque de l'inégalité de Kraft : si $(l_i)_{i=1}^n$ est une famille d'entiers telle que

   $$\sum_{i=1}^{n} K^{-l_i} \leq 1$$

   alors il existe une famille de mots distincts $(u_i)_{i=1}^n$ de Σ^* formant un ensemble sans préfixe avec $|u_i| = l_i$ pour tout i.

   Donner un algorithme prenant en entrée une telle suite l_i qui retourne une telle suite u_i. On demande de plus que l'algorithme soit 'en ligne', c'est-à-dire que l'algorithme lit les l_i un par un et produit u_i immédiatement après la lecture de l_i, sans attendre la valeurs des l_j pour $j > i$. Indication: Utiliser un invariant faisant intervenir, à l'étape $t + 1$, l'écriture en base K de $1 - \sum_{i=1}^{t} K^{-l_i}$.

8. Montrer que l'inégalité de Kraft reste vraie pour tout ensemble de mots Γ (fini ou infini) vérifiant la propriété de décomposabilité unique de la

9. Indication: On pourra considérer la série entière $\sum_n a_n x^n$ ò a_n est le nombre de mots de Γ de longueur n - que sait-on de son rayon de convergence? - et étudier $\left(\sum_n a_n x^n\right)^N$ pour un 'grand' entier N.

Corrigé

Question 1. Trouver des ensembles finis sans préfixe est trivial. Pour un ensemble infini, on peut prendre par exemple

$$\Gamma = \{a^n b \mid n \in \mathbb{N}\}$$

Question 2. Par récurrence sur la longueur de w. Pour $|w| = 0$ aucune décomposition n'est possible car Γ ne contient pas le mot vide, la propriété est donc satisfaite de facto. Supposons maintenant qu'on a deux décompositions

$$w = u_1 \dots u_n = v_1 \dots v_m$$

(avec les u_i et v_j dans Γ, donc en particulier non-vides). Sans perte de généralité on a $0 < |u_1| \leq |v_1|$, et l'égalité ci-dessus implique que u_1 est donc un préfixe de v_1, donc $u_1 = v_1$ car Γ est sans préfixe. Il suit que $u_2 \dots u_n = v_2 \dots v_m$ et on conclut par récurrence, le mot $w' = u_2 \dots u_n$ étant de longueur strictement inférieure à celle de w.

Question 3. On teste pour toute paire (u, v) de mots distincts de Γ si u est un préfixe de v. Chaque comparaison prend un temps $O(|u| + |v|)$ donc on a une complexité totale en $O\left(n^2\right), n$ étant la taille mémoire de l'entrée Γ.

Question 4. Par récurrence sur la hauteur h de l'arbre, on montre en même temps l'inégalité et le fait qu'on a égalité si et seulement si l'arbre est localement complet (tout noeud est soit une feuille soit a exactement K fils). On notera $f(T) = \sum_{\sigma \text{ feuille de } T} K^{-hauteur}(\sigma)$.

Pour $h = 0$, on a une seule feuille de hauteur 0 donc $\sum_{w \in \Gamma} K^{-|w|} = 1$ et l'arbre est bien localement complet.

Pour $h + 1$, on se observe que

$$f(T) = \frac{1}{K} \sum_i f\left(T_i'\right)$$

la somme étant prise sur les sous-arbres non-vides de T. En effet la hauteur d'un noeud dans un T_i' est un de moins que sa hauteur dans T. Par récurrence, on a $f\left(T_i'\right) \leq 1$ pour tout i et on a au plus K sous-arbres, d'où l'inégalité $f(T) \leq 1$. Pour avoir l'égalité, il faut avoir exactement K sous-arbres et $f\left(T_i'\right) = 1$ pour chacun d'entre eux, ce qui oblige par récurrence a avoir chaque T_i' localement complet et donc T doit être lui-même localement complet.

Question 5. Il y a une interprétation assez évidente d'ensemble préfixe (non-vide) en terme d'arbre. Soit h la longueur du plus long mot de Γ. On peut identifier chaque mot de Γ comme un noeud de l'arbre K-aire parfait T de hauteur h, où la racine correspond au mot vide et inductivement si un noeud correspond au mot w, ses fils correspondent aux mots $\{wa \mid a \in \Sigma\}$. Le fait d'être sans préfixe est équivalent à ce que parmi les noeuds de Γ on n'en ait pas un qui soit préfixe d'un autre. On peut alors considérer l'arbre $T_\Gamma = \{u \mid u \text{ est préfixe d'un mot } w \in \Gamma\}$. Si Γ est sans préfixe, chaque $w \in \Gamma$ est une feuille de T_Γ, avec hauteur $(w) = |w|$. L'inégalité de Kraft suit.

De plus, on a égalité quand Γ est maximal, c'est-à-dire contenu dans aucun ensemble de mots sans préfixe strictement plus grand. Par la Question 4, il suffit de montrer que Γ est maximal si et seulement si son arbre T_Γ est localement parfait. En effet, si T_Γ n'est pas localement parfait, un noeud non feuille w a l'un de ses fils wa manquants, on peut donc rajouter le mot wa à Γ en préservant la propriété d'être sans préfixe. A l'inverse, si Γ n'est pas maximal, considérons le mot le plus court parmi les mots qu'on peut ajouter à Γ. Ecrivons ce mot va avec $v \in \Sigma^*$ et $a \in \Sigma$ (pour être complètement précis il faut traiter le cas dégénéré où ce mot est le mot vide; ceci arrive seulement quand $\Gamma = \emptyset$, et dans ce cas il n'y a rien à démontrer). Par hypothèse, v est préfixe d'un mot de Γ donc v est dans T_Γ mais pas $va : T_\Gamma$ n'était donc pas parfait.

Montrons enfin que l'inégalité de Kraft reste vraie pour les ensembles sans préfixe infini. Soit Γ un tel ensemble, et notons Γ_n l'ensemble des n-premiers éléments de Γ. Comme on a à faire à une série de termes positifs uniquement, par le théorème de converge monotone,

$$\sum_{w \in \Gamma} 2^{-|w|} = \lim_n \sum_{w \in \Gamma_n} 2^{-|w|}$$

Or Γ_n est sans préfixe pour tout n (un sous-ensemble d'un ensemble sans préfixe est sans préfixe), donc $\sum_{w \in \Gamma_n} 2^{-|w|} \leq 1$ pour tout n et la limite est donc elle-même ≤ 1. [Par contre, pour les ensembles infinis on n'a plus égalité ssi maximalité].

Question 6. Il suffit de placer les mots un par un dans un arbre K-aire comme dans la question précédente. A chaque ajout on parcourt l'arbre et si on rencontre un noeud déjà pris, alors Γ n'est pas sans préfixe. Si aucun conflit n'a lieu durant la construction, alors Γ est sans préfixe. Clairement cet algorithme tourne en temps linéaire.

Question 7. Là encore on va placer les mots u_i dans un arbre K-aire dont ils seront des feuilles (ce qui garantira la propriété d'être sans préfixe). On commence avec l'arbre réduit à la racine, qui va grandir au fur et à mesure. A chaque étape on aura deux types de feuilles : les feuilles déjà prises par un mot de w et les feuilles disponibles. De plus, on va s'arranger pour qu'après l'étape t (c'est-à-dire après avoir construit et placé $(u_i)_{i=1}^t$) en écrivant le réel $1 - \sum_{i=1}^{t} K^{-l_i}$ en base K :

$$1 - \sum_{i=1}^{t} K^{-l_i} = \sum_{n=0}^{\infty} a_n K^{-n}$$

où a_n est nulle à partir d'un certain rang, on ait exactement a_n feuilles de disponibles à la hauteur n. Initialement c'est bien ce qu'on a : la racine a hauteur 0 et est disponible.

Supposons qu'on ait jusqu'à maintenant respecté cette propriété après t étapes. On lit maintenant la valeur l_{t+1} et o distingue deux cas :

- Soit il y a au moins une feuille de disponible à la hauteur l_{t+1}. Dans ce cas on prend une telle feuille et elle devient notre u_{t+1} (et donc devient non-disponible)

- Soit aucune feuille de hauteur l_{t+1} n'est disponible. Dans ce cas on cherche le plus grand $n < l_{t+1}$ tel qu'on a une feuille w de disponible à la hauteur n. On agrandit alors l'arbre sous cette feuille, en ajoutant les feuilles disponibles suivantes:

$$\{wa^m b \mid b \in \Sigma, b \neq a, m \in [0, l_{t+1} - n - 1]\}$$

où a est une lettre fixée de Σ, ainsi que les noeuds internes

$$\{wa^m \mid m \in [0, l_{t+1} - n - 1]\}$$

et en ajoutant également la feuille $wa^{l_{t+1}-n}$, qu'on attribue à u_{t+1}, et qui a bien longueur $t+1$. Dans les deux cas on a bien préservé l'écriture en base K de $1 - \sum_{i=1}^{t} K^{-l_i}$: le premier cas correspond à

(avec $a_n > 0$) tandis que le second correspond à la propagation de la retenue

Ceci prouve la correction de l'algorithme. [Il fonctionne, comme le test de la question précédente, en temps linéaire].

Question 8. Formons la série entière proposée $H(x) = \sum_n a_n x^n$. Comme a_n est le nombre de mots de Γ de longueur n, on a $a_n \leq K^n$. Donc si $x < 1/K$, on a convergence absolue de la série, le rayon de convergence est donc $\geq 1/K$. De plus, tous les a_n étant positifs, on a, encore une fois par le théorème de convergence monotone,

$$\sum_{w \in \Gamma} K^{-|w|} = \sum_n a_n K^{-n} = \lim_{x \to (1/K)^-} \sum_n a_n x^n$$

On va maintenant montrer que $\lim_{x \to (1/K)^-} \sum_n a_n x^n \leq 1$. Soit $x \in [0, 1/K\ [$ et N un entier quelconque. On a

$$\left(\sum_n a_n x^n \right)^N = \sum_n \left(\sum_{\substack{i_1, \ldots, i_N \\ i_1 + \ldots + i_N = n}} a_{i_1} + \ldots + a_{i_N} \right) x^n$$

Mais le terme

$$\sum_{\substack{i_1, \ldots, i_N \\ i_1 + \ldots + i_N = n}} a_{i_1} + \ldots + a_{i_N}$$

est exactement le nombre de façons de choisir un N-uplet $(u_1, \ldots, u_N)$ avec tous les u_i dans Γ et tels que $|u_1| + \ldots + |u_N| = n$. Mais par la propriété d'unique décomposabilité, à chaque tel choix correspond un unique $w = u_1 \ldots u_N$ de longueur n. On a donc

$$\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N} \leq K^n$$

pour tout n. Donc, pour tout $x < 1/K$ fixé, on a

$$H(x)^N = \left(\sum_n a_n x^n \right)^N = \sum_n \left(\sum_{\substack{i_1,\ldots,i_N \\ i_1+\ldots+i_N=n}} a_{i_1} + \ldots + a_{i_N} \right) x^n \leq \sum_n K^n x^n < \infty$$

En appelant $G(x) = \sum_n K^n x^n$, qui est aussi une série entière de rayon de convergence $\geq 1/K$, on a établi :

$$H(x)^N \leq G(x)$$

pour tout $|x| < 1/K$ et tout ă N. Cela implique nécessairement $H(x) \leq 1$ pour tout $|x| < 1/K$, car sinon l'inégalité ci-dessus serait violée pour N assez grand. On a donc bien

$$\lim_{x \to (1/K)^-} H(x) \leq 1$$

V Exercices langages

Soit Σ un alphabet. Si $u = u_1...u_n$ et $v = v_1...v_n$ sont deux mots de même longueur sur Σ, leur distance de Hamming est:

$$d(u, v) = |\{i \mid u_i \neq v_i\}|$$

1. Montrer que la distance de Hamming est une distance sur Σ^*.
2. Écrire une fonction `dist : 'a list -> 'a list -> int` calculant la distance de Hamming de deux mots de même longueur, sous forme de listes.

Étant donné un langage L sur Σ, on définit son voisinage de Hamming $\mathcal{H}(L) = \{u \in \Sigma^* \mid \exists v \in L,\ d(u, v) = 1\}$.

3. Donner une expression rationnelle du voisinage de Hamming de $L(0^*1^*)$.
4. Définir par récurrence une fonction H telle que, si e est une expression rationnelle d'un langage L sur $\Sigma = \{0, 1\}$, $H(e)$ est une expression rationnelle de $\mathcal{H}(L)$.
5. Écrire la fonction H précédente en Caml.

1. À quelle condition nécessaire et suffisante simple le langage reconnu par un automate est vide? Décrire un algorithme pour le savoir.
2. À quelle condition nécessaire et suffisante simple le langage reconnu par un automate est fini? Décrire un algorithme pour le savoir.
3. Décrire un algorithme pour déterminer si deux automates admettent le même langage.
4. Soit A un automate à n états. Montrer que si $L(A) \neq \emptyset$ alors $L(A)$ contient un mot de longueur au plus $n - 1$.