I Sujet 1

I.1 Langage non rationnel

Montrer que $L = \{a^p : p \text{ premier } \}$ n'est pas rationnel.

I.2 Distance de Hamming

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.

II Sujet 2

II.1 Hauteur d'étoile

La hauteur d'étoile h d'une expression régulière est définie récursivement de la manière suivante :

- $h(e) = 0$ si e est $\emptyset$, ε ou une lettre.
- $h(e_1 + e_2) = \max(h(e_1), h(e_2))$.
- $h(e_1 e_2) = \max(h(e_1), h(e_2))$.
- $h(e^*) = h(e) + 1$.

1. Quelle est la hauteur d'étoile de $(ba^*b)^*$?
2. Écrire la fonction h : `'a regexp -> int` en Caml.

La hauteur d'étoile d'un langage L est la plus petite hauteur d'étoile d'une expression rationnelle e de langage L.

3. Que peut-on dire des langages de hauteur d'étoile 0?
4. Montrer que la hauteur d'étoile de $L((ba^*b)^*)$ est 1.

II.2 Questions sur les automates

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 $n - 1$.

III Sujet 3

III.1 Nombres rationnels

1. Si $x \in \mathbb{R}$, on note $L(x)$ l'ensemble des préfixes des chiffres de x après la virgule. Par exemple, $L(\pi) = \{\varepsilon, 1, 14, 141, 1415...\}$. En sachant que $\frac{1}{6} = 0.1666...$ et $\frac{1}{7} = 0.142857142857...$, montrer que $L(\frac{1}{6})$ et $L(\frac{1}{7})$ sont rationnels.
2. Montrer que $L(x)$ est rationnel si et seulement si $x \in \mathbb{Q}$.

III.2 Stabilité des langages reconnaissables

Si $m = m_1...m_n$ est un mot, on définit son miroir $\widetilde{m} = m_n...m_1$.
Si L est un langage, on définit son miroir $\widetilde{L} = \{\widetilde{m} \mid m \in L\}$.

1. Montrer que le miroir d'un langage reconnaissable est reconnaissable.

Si L est un langage sur Σ, on définit:

- $Pref(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*,\ uv \in L\}$: ensemble des préfixes des mots de L.
- $Suff(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*,\ vu \in L\}$: ensemble des suffixes des mots de L.
- $Fact(L) = \{u \in \Sigma^* \mid \exists v, w \in \Sigma^*,\ vuw \in L\}$: ensemble des facteurs des mots de L.

2. Montrer que si L est reconnaissable alors $Pref(L)$, $Suff(L)$, $Fact(L)$ le sont aussi. Décrire un algorithme pour obtenir les automates correspondants.
3. Montrer que si L est rationnel alors $Pref(L)$, $Suff(L)$, $Fact(L)$ le sont aussi (puisqu'on va montrer que rationnel = reconnaissable, c'est une preuve alternative à la précédente).

A2 – Langages réguliers épars

On fixe l'alphabet fini $\Sigma = \{a, b\}$. Un *mot* $w \in \Sigma^*$ est une suite finie d'éléments de Σ, et un langage $L \subseteq \Sigma^*$ est un ensemble de mots. La *densité* de L est la fonction $\delta_L : \mathbb{N} \to \mathbb{N}$ où $\delta_L(n) := |L \cap \Sigma^n|$ pour $n \in \mathbb{N}$ est le nombre de mots de longueur n dans L. On dit que L est *épars* si on a $\delta_L = O(P)$ pour un polynôme P.

Un langage *régulier* est un langage dénoté par une expression rationnelle, ou reconnu par un automate déterministe fini (on admet que ces caractérisations sont équivalentes). Tous les automates sont supposés déterministes.

Question 0. Donner un exemple d'un langage régulier épars, et d'un langage régulier non-épars.

Question 1. Est-il vrai que l'union de deux langages réguliers épars est un langage régulier épars ? Que penser de la concaténation ? Que penser de l'itération (étoile de Kleene) ?

Question 2. Est-il vrai qu'un langage est épars si et seulement si son complémentaire ne l'est pas ?

Question 3. Soient u, v, v', w des mots de Σ^*. Supposons qu'un langage L contienne tous les mots du langage dénoté par l'expression régulière $u(av|bv')^*w$. Montrer que L n'est pas épars.

Question 4. En s'inspirant de la question précédente, proposer une condition suffisante sur un automate fini pour que le langage accepté par l'automate ne soit pas épars.

Question 5. Écrire le pseudocode d'un algorithme naïf pour tester le critère de la question 4 sur un automate fourni en entrée, et discuter de sa complexité.

Question 6. On suppose qu'étant donné l'automate, on dispose d'une fonction permettant de calculer les composantes fortement connexes de son graphe en temps linéaire. En déduire un algorithme pour tester le critère de la question 4 en temps linéaire.

Question 7. Montrer que le critère identifié en question 4 est en réalité une caractérisation des automates reconnaissant des langages qui ne sont pas épars.

Question 8. Conclure à la caractérisation suivante : les langages réguliers épars sont exactement les unions finies de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_k^* u_k$ pour des mots $u_0, \ldots, u_k, v_1, \ldots, v_k \in \Sigma^*$.

Suite des questions

Question 9. Quels sont les régimes possibles pour la fonction de densité d'un langage régulier, et quels sont les langages les réalisant ?

Question 10. Quelle est la complexité, étant donné un automate, de déterminer le régime de la fonction de densité de son langage ?

Corrigé

Question 0. Le langage vide est épars. Le langage Σ^* ne l'est pas puisque δ_{Σ^*} est la fonction qui à n associe 2^n, qui n'est pas dominée par un polynôme.

Question 1. On note d'abord que les langages réguliers sont clos par ces opérateurs, donc il suffit de vérifier si les langages épars le sont. *[On s'attend bien à ce que la clôture des langages réguliers par ces opérations soit mentionnée.]*

Les langages épars sont clos par union : pour L_1 et L_2 deux langages épars, si on a $\delta_{L_1} = O(P_1)$ et $\delta_{L_2} = O(P_2)$ pour deux polynômes P_1 et P_2, alors comme pour tout $n \in \mathbb{N}$ on a $|(L_1 \cup L_2) \cap \Sigma^n| = |L_1 \cap \Sigma^n \cup L_2 \cap \Sigma^n| \leq |L_1 \cap \Sigma^n| + |L_2 \cap \Sigma^n|$, on a $\delta_{L_1 \cup L_2} = O(P_1 + P_2)$ et $P_1 + P_2$ est également un polynôme. *[On exigera la preuve formelle.]*

Les langages épars sont clos par concaténation : soient L_1 et L_2 deux langages épars. Pour simplifier, on les suppose dominés par un polynôme commun, de sorte à ce que $\delta_{L_1} = O(P)$ et $\delta_{L_2} = O(P)$ pour un polynôme P, que pour simplifier l'on supposera également croissant. Pour tout $n \in \mathbb{N}$, on a $|L_1 L_2 \cap \Sigma^n| = \left|\cup_{0 \leq i \leq n} (L_1 \cap \Sigma^i) \times (L_2 \cap \Sigma^{n-i})\right|$. Ceci est majoré par $\sum_{0 \leq i \leq n} \left|L_1 \cap \Sigma^i\right| \times \left|L_2 \cap \Sigma^{n-i}\right|$, majorable par hypothèse par $\sum_{0 \leq i \leq n} P(i)P(n-i)$. Comme P est croissant ceci se majore grossièrement par $(n+1)P(n)^2$. C'est également un polynôme.

Les langages épars ne sont manifestement pas clos par étoile : le langage fini $a + b$ est épars mais son itération est Σ^* qui ne l'est pas.

Question 2. C'est faux, car il se peut qu'aucun des deux ne soit épars : le complémentaire de $a\Sigma^*$ est $b\Sigma^*|\epsilon$ et aucun des deux n'est épars.

En revanche, il est bien sûr impossible qu'un langage et son complémentaire soient tous deux épars, puisque leur union serait alors éparse par la question précédente, or il s'agit de Σ^* qui ne l'est pas.

Question 3. Soit L' le langage de l'expression rationnelle. Il suffit de montrer que L' n'est pas épars pour conclure. *[La formulation de la question avec L et L' est pour aider pour la question suivante.]*

Posons n_0 la somme des longueurs de u et w, posons m le PPCM des longueurs de av et bv' de sorte que $m = p\,|av| = q\,|bv'|$, et étudions $\delta_{L'}$ aux valeurs $n_0 + km$ pour $k \in \mathbb{N}$. Pour chaque "bloc" de taille m, on a au moins deux possibilités (manifestement différentes) : le remplir par des av, ou le remplir par des bv'. (Bien sûr, on a potentiellement davantage de possibilités en réalité.) Mais ceci assure que $\delta_{L'}(n_0 + km) \geq 2^k$, manifestement non dominable par un polynôme.

Question 4. *[Attention, pour le bon déroulement de la suite du sujet, il faut parvenir exactement à la formulation proposée.]*

La question 3 suggère que si l'on considère un automate déterministe alors il reconnaîtra un langage non-épars s'il y a un état q satisfaisant les conditions suivantes :

— Il y a un chemin (correspondant à u) allant de l'état initial à q ;

— Il y a une transition a allant de q à un état q', et un chemin (correspondant à v) revenant de q' à q ;

— Il y a une transition b allant de q à un état q'', et un chemin (correspondant à v') revenant de q'' à q ;

— Il y a un chemin (correspondant à w) allant de q à un état final.

On peut reformuler : il y a un état q dans l'automate qui est accessible et co-accessible (attention, ces deux termes ne sont pas au programme) avec deux boucles différentes allant de q à lui-même, l'une commençant par a et l'autre par b.

C'est une condition suffisante : si un automate présente ce motif alors il reconnaît un langage non épars par la question précédente. La question 7 montre qu'il s'agit en réalité d'une condition nécessaire et suffisante : un automate sans ce motif reconnaît un langage épars.

Question 5. Attention, le graphe correspondant à l'automate est en réalité un multi-graphe, avec potentiellement des boucles. Les détails de la représentation d'entrée ne sont pas spécifiés pour pouvoir en discuter avec le candidat ou la candidate, qui peut faire les choix qui l'arrangent. Il faut être indulgent toutefois car les multi-arêtes et les boucles ne sont explicitement pas au programme.

Pour commencer, il ne faut pas oublier d'éliminer les états inutiles. On peut le faire naïvement en temps quadratique, voire cubique (pour chaque état, pour chaque état final, faire un test d'accessibilité ; et de même avec l'état initial), ou plus intelligemment en temps linéaire : exploration BFS à partir de l'état initial pour marquer les accessibles, et une seule exploration BFS à partir de l'ensemble des états finaux (et en inversant le sens des arêtes pour marquer les co-accessibles). *[Pour cette question, ne pas exiger d'algorithme efficace pour cela : cf la question suivante.]*

Ensuite, on teste le motif à rechercher : pour chaque choix de sommet q ayant deux transitions a et b allant respectivement vers q' et q'' (noter que bien sûr q, q', q'' ne sont pas nécessairement distincts), on teste s'il y a un chemin de q' à q et de q'' à q. C'est en $O(nm)$ pour n le nombre d'états et m le nombre de transitions.

Par exemple :

```
Input: graphe G de n sommets représenté par listes d'adjacences

Fonction Accessible(u, v):
  Q := file()
  Q.insère(u)
  Visité = tableau de n cases initialisé à False
  Tant que Q n'est pas vide:
    x := Q.top()
    Q.pop()
    Si x == v:
      Renvoyer Vrai
    Fin Si
    Si Visité[x]:
      Continuer
    Fin Si
    Visité[x] := Vrai
    Pour chaque (x, y):
      Q.push(y)
    Fin Pour
  Fin Tant que
  Renvoyer Faux
Fin Fonction

Utile = tableau de n cases initialisé à Vrai
```

```
Pour chaque état q:
  Si non Accessible(qinit, q):
    Utile[q] := Faux
  Fin Si
Fin Pour

Pour chaque état q:
  OK := Faux
  Pour chaque état final qf:
    Si accessible(q, qf):
      OK := Vrai
      Break
    Fin Si
  Fin Pour
  Si non OK:
    Utile[q] := Faux
  Fin Si
Fin Pour

Pour chaque q ayant deux transitions (q, qa) et (q, qb):
  Si Utile[q] et accessible(qa, q) et accessible(qb, q):
    Renvoyer Vrai
  Fin Si
Fin Pour

Renvoyer Faux
```

Question 6. *[On rappelle que la notion de composante fortement connexe (CFC) d'un graphe orienté est exigible.]*

Il faut comme précédemment éliminer les états non-accessibles et non-coaccessibles, mais cette fois il faut impérativement le faire en temps linéaire.

On note que le motif que l'on recherche se situe forcément dans une CFC, donc il suffit de considérer chaque CFC.

À présent, notons qu'une occurrence du motif interdit dans une CFC témoigne de l'existence d'un état q avec des transitions a et b menant à des états respectifs q' et q'' qui sont dans la même CFC. Mais réciproquement, s'il y a un tel état q, alors comme q' et q'' sont dans la même CFC que q, il y a un chemin retour, et donc un motif interdit.

Pour le reformuler autrement, la condition revient à affirmer que chaque CFC est soit un cycle, soit un cycle vide (un seul état dont toutes les transitions sortantes sortent de la CFC).

Pour résumer, après élimination des états inutiles, si on suppose chaque état annoté par un identifiant de sa CFC, alors il suffit de vérifier chaque état q pour tester si on a deux transitions sortantes qui sont vers des états de la même CFC que q (potentiellement identiques, potentiellement q lui-même). C'est manifestement faisable en temps linéaire.

Question 7. Montrons que si l'automate n'a pas le motif interdit, ou autrement dit si chacune de ses CFC est un cycle (après suppression des états inutiles), alors le langage qu'il reconnaît est épars. L'automate étant déterministe, chaque mot de son langage a un unique chemin acceptant (allant d'un état initial à un état final). Notons K le nombre de CFC ; c'est une constante.

On considère l'application ϕ associant à chaque mot les informations suivantes sur son chemin

acceptant : pour chaque CFC, l'information de la première et dernière position du chemin acceptant où on est dans cette CFC (c'est clairement un segment du chemin, noter en particulier que par définition on ne peut pas revenir dans une CFC une fois qu'on en est parti), et les états à cette première et dernière position. L'application ϕ associe donc à un mot de longueur n un K-tuple d'entiers entre 0 et n et un $2K$-tuple d'états de Q. Ainsi, il y a au plus $(n+1)^K \times |Q|^2 K$ images possibles ; c'est un polynôme P de degré K.

Or, l'application ϕ est injective. En effet, sachant qu'un chemin acceptant se trouve à l'état q en position i et r en position j, le chemin entre i et j doit rester dans la CFC commune de q et r, et comme cette CFC est un cycle il y a un seul chemin possible.

Pour L le langage accepté par l'automate, on a donc $|L \cap \Sigma^n| \leq P$ par ce qu'on vient de dire, ce qui conclut que le langage est épars.

Question 8. Un langage de la forme indiquée est manifestement régulier, et épars pour la même raison qu'en question précédente. Ainsi, des unions finies de tels langages sont des langages réguliers épars.

Pour la réciproque, on raisonne comme à la question précédente : un langage régulier épars est accepté par un automate qui satisfait donc le critère de la question 4, or le langage accepté par un tel automate s'écrit comme une union finie de langages de la forme requise suivant leur image par la fonction ϕ. Intuitivement, les u_i correspondent à des tours complets de boucle dans une CFC, et les v_i correspondent à des déplacements entre CFC ou à des tours incomplets de boucle dans une CFC (qu'on peut mettre par convention à la fin du segment où l'on reste dans une CFC).

Question 9. Il y a les langages non-épars dont la fonction de densité est $\Theta(2^{\Theta(n)})$, que nous avons caractérisés comme ceux n'ayant pas la forme de la question 8, ou acceptés par un automate comportant le motif interdit de la question 4. Ensuite, on peut montrer que pour un langage épars L, les conditions suivantes sont équivalentes :

1. L est de densité $O(n^k)$;
2. L est accepté par un automate où il n'y a pas de chemin acceptant qui passe par $k+1$ CFC cycliques (CFC qui ne sont pas un singleton sans boucle) ;
3. L est une union finie de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_t^* u_t$ avec $t \leq k$.

En effet, (3) implique manifestement (1). Ensuite, (2) implique (3) exactement comme à la question 7. Par ailleurs, la négation de (2) implique la négation de (1) dans le même esprit qu'en questions 3 et 4 : si l'on prend un chemin témoin passant par $k+1$ CFC cycliques avec au moins un tour de boucle de chaque, que l'on ajuste les parties intermédiaires pour identifier des tours entiers de boucles dans chaque CFC, et qu'on prend le PPCM des longueurs de boucle, alors on a la possibilité de déplacer les $k+1$ points intermédiaires entre CFCs cycliques en contrôlant le nombre de tours de chaque boucle, donc $\Omega(n^{k+1})$ possibilités.

D'où la caractérisation :

1. L est de densité $\Theta(n^k)$;
2. L est accepté par un automate où il n'y a pas de chemin acceptant qui passe par $k+1$ CFC cycliques (CFC qui ne sont pas un singleton sans boucle) mais il y en a un qui passe par k telles CFC ;
3. L est une union finie de langages de la forme $u_0 v_1^* u_1 v_2^* u_2 \cdots v_t^* u_t$ avec $t \leq k$, avec $t = k$ pour au moins un terme de l'union.

Noter le cas particulier $k = 0$ où la densité est donc bornée, il n'y a aucune CFC cyclique (donc le langage est bien fini), et L est une union finie de langages singletons.

Ceci implique en particulier que, parmi les langages épars, la densité doit être équivalente à un polynôme dont le degré est déterminable par la caractérisation ci-dessus, et notamment qu'aucun régime "intermédiaire" n'est possible.

Question 10. On a vu en question 6 comment tester en temps linéaire si un automate accepte ou non un langage épars (par la caractérisation des questions 4 et 7).

Si le langage est épars, une fois calculées les CFC, et étiquetées les CFC cycliques, le plus grand nombre de CFC cycliques traversées par un chemin acceptant se calcule de bas en haut en temps linéaire, ce qui permet de savoir en temps également linéaire où on se place dans les régimes de la question 9.

[Les résultats de ce sujet ont été originellement montrés dans [SYZS92], et peuvent également être trouvés dans [Pin19], Chapitre XII, Section 4.2]

Références

[Pin19] Jean-Éric Pin. Mathematical foundations of automata theory. `https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`, 2019.

[SYZS92] Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing regular languages with polynomial densities. In *MFCS*, 1992. Non disponible en libre accès.

A5 – Ensembles inévitables

On fixe un alphabet fini $\Sigma = \{a, b\}$. Pour $w \in \Sigma^*$, on écrit $w = w_1 \cdots w_n$ où $n := |w|$ est la *longueur* de w. On dit qu'un mot $w \in \Sigma^*$ *évite* un mot $s \in \Sigma^*$ si s n'apparaît *pas* comme facteur de w. On dit que w *évite* un ensemble de mots $S \subseteq \Sigma^*$ s'il évite chaque mot de S.

Question 0. Donner un mot de longueur au moins 12 qui évite l'ensemble $S = \{aaaa, aaab, aba, baaa, bab, bbbb\}$.

Question 1. Donner le pseudocode d'un algorithme simple qui détermine, étant donné un mot $w \in \Sigma^*$ et un ensemble fini $S \subseteq \Sigma^*$, si w évite S. Analyser sa complexité en temps et en espace quand tous les mots de S font la même longueur.

On dit qu'un ensemble $S \subseteq \Sigma^*$ est *inévitable* s'il n'existe qu'un nombre fini de mots $w \in \Sigma^*$ qui évitent S. Sinon, il est dit *évitable*.

Question 2. L'ensemble de la question 0 est-il inévitable ? L'ensemble $\{aaa, abb, baa, abab\}$ est-il inévitable ?

Question 3. Montrer que l'ensemble $\{aaa, bab, baab, bbb\}$ est inévitable.

Question 4. Montrer le lemme suivant : pour tout alphabet fini Σ fixé et $k \in \mathbb{N}$, il existe $n \in \mathbb{N}$ tel que pour tout mot $w \in \Sigma^*$ tel que $|w| > n$, il existe deux entiers $p < q$ tels que pour tout $0 \leq l < k$, on ait $w_{p+l} = w_{q+l}$.

Question 5. Utiliser cette observation pour proposer un algorithme (pas nécessairement efficace) qui, étant donné un ensemble fini $S \subseteq \Sigma^*$, détermine si S est inévitable. Décrire sa complexité en temps et en espace.

Question 6. On dit qu'une expression rationnelle e est *inévitable* si l'ensemble (généralement infini) des mots du langage de e est inévitable. Donner un exemple d'expression rationnelle inévitable.

Question 7. L'expression rationnelle e est *bornée* s'il existe $n \in \mathbb{N}$ tel que tout mot satisfaisant e soit de longueur $\leq n$. Si e est bornée, comment peut-on déterminer si elle est inévitable ?

Question 8. Expliquer comment déterminer si une expression rationnelle e quelconque est inévitable.

Question 9. Pour e une expression rationnelle, on dit qu'un ensemble $S \subseteq \Sigma^*$ est *inévitable pour* e s'il existe un ensemble infini de mots du langage de e qui évitent S. Étant donné une expression rationnelle e et un ensemble fini S, expliquer comment déterminer si S est inévitable pour e.

Question 10. Montrer que, pour tout ensemble inévitable $S \subseteq \Sigma^*$, il existe un sous-ensemble $S' \subseteq S$ fini tel que S' soit inévitable.

Question 11. Étant donné une expression rationnelle e inévitable, expliquer comment calculer un sous-ensemble inévitable fini du langage de e.

Corrigé

Question 0. On peut prendre par exemple *aabbaabbaabb*.

Question 1. La principale difficulté est d'être rigoureux avec les indices.

```
Entrée:
  - mot w représenté comme un tableau de longueur n,
  - ensemble S représenté comme un tableau de longueur m de tableaux de
  longueurs nS[i] pour chaque i

Pour j de 0 à m exclu faire:
  Pour i de 0 à n-nS[j] exclu faire:
    ok := True
    Pour d de 0 à nS[j] exclu faire:
      Si w[i+d] != S[i][d]:
        ok := False
        Sortir de Pour
      Fin si
    Fin pour
    Si ok:
      // on a trouvé une occurrence de S[j] dans w
      Renvoyer faux
    Fin si
  Fin pour
Fin pour
Renvoyer vrai
```

Si les m mots de S font la même longueur l, la complexité en temps est en $O(n \times m \times l)$ et la complexité en espace est constante.

[On peut faire mieux en temps en utilisant l'algorithme de Knuth-Morris-Pratt (prétraitement en $O(m \times l)$ puis vérification en $O(n \times m)$) ou l'algorithme d'Aho-Corasick (prétraitement en $O(m \times l)$ puis vérification en $O(n)$), mais cela n'est pas demandé.]

Question 2. L'ensemble de la question 0 est évitable puisque, pour tout $i \in \mathbb{N}$, le mot $(aabb)^i$ l'évite.
L'ensemble $\{aaa, abb, baa, abab\}$ est évitable aussi : prendre b^i pour tout $i \in \mathbb{N}$.

Question 3. Procédons par l'absurde. Comme l'ensemble contient *aaa* et *bbb*, un mot suffisamment long évitant l'ensemble doit nécessairement comporter un b suivi d'un a, c'est-à-dire un facteur ba ; et si le mot est suffisamment long ce facteur est suivi d'autres caractères. Mais alors ce facteur ne peut être suivi de b (sinon on a un facteur *bab*), donc il est suivi de a, c'est-à-dire un facteur *baa*. Mais ce facteur ne peut être suivi de a (ce qui donnerait *aaa*), ni de b (ce qui donnerait *baab*), contradiction. Ainsi l'ensemble est-il bien inévitable.

Question 4. C'est une application immédiate du principe des tiroirs. On prend $n = |\Sigma|^k + k$. Ce mot a strictement plus de $|\Sigma|^k$ positions où commence un facteur de taille k, donc par application du principe des tiroirs il y a deux positions différentes où commence un facteur identique, ce qui est le résultat voulu.

Question 5. Soit k la longueur maximale d'un mot de S. Montrons tout d'abord que S n'est pas inévitable si et seulement s'il existe un mot évitant S avec deux occurrences distinctes d'un même facteur de longueur k. *(C'est une indication possible.)*

Dans le sens direct, si S est évitable alors il existe une infinité de mots évitant S, donc il existe des mots arbitrairement longs évitant S, en particulier des mots de longueur strictement supérieure au n de la question précédente. Ainsi, si on prend un tel mot, il a deux occurrences distinctes du même facteur de taille k.

Réciproquement, s'il existe un mot évitant S avec deux occurrences du même facteur de longueur k, alors on enlève un suffixe de ce mot pour que le mot se termine à la fin de la deuxième occurrence du facteur : le mot w_0 résultant évite toujours S. Soit $d > 0$ le décalage entre les deux occurrences. On peut ensuite construire des mots de plus en plus longs évitant S en ajoutant itérativement une copie de la lettre à d positions en partant de la fin : les mots résultants évitent tous S car ils ont un ensemble de facteurs de longueur k qui est le même que celui de w_0, qui évitait S, or la propriété d'éviter S est entièrement déterminée par les facteurs de longueur k par définition de k. Ainsi, S est évitable.

La caractérisation que nous avons démontrée implique que S est inévitable si et seulement si tous les mots de longueur $> n$ ont un facteur dans S, pour n la valeur de la question précédente. En effet, le sens réciproque est trivial (car il y a alors un nombre fini de mots évitant S), et pour le sens direct on vient de montrer que si S est inévitable alors tout mot ayant deux occurrences distinctes d'un même facteur de longueur k n'évite pas S, donc en particulier tous les mots de longueur $> n$.

Ceci suggère l'algorithme suivant : on énumère tous les mots de longueur $\leq n$, et on teste s'ils ont un facteur dans S. La complexité de cet algorithme est en $\left(\sum_{w \in S} |w|\right) \times |\Sigma|^k + k$ où k est la longueur maximale d'un mot de S, c'est-à-dire exponentiel en S. La complexité en espace demande de stocker le mot en cours, c'est-à-dire $|\Sigma|^k + k$.

En pratique il vaudrait mieux construire incrémentalement un mot par backtracking, en abandonnant si on contient un facteur dans S, et en réussissant si on a répété un facteur de longueur k déjà atteint.

Un bien meilleur algorithme (en $O(|\Sigma|^{k+1})$ pour k la longueur maximale d'un mot de S) est de considérer chaque mot de Σ^k et de construire un graphe orienté sur cet ensemble (avec $|\Sigma|^{k+1}$ arêtes) : on relie w à w' si et seulement s'il existe $a \in \Sigma$ tel que, si on écrit $w = a'w''$, on a $w' = w''a$. On élimine ensuite tous les sommets qui contiennent un mot de S. Il est clair que si ce graphe a un cycle alors on construit (comme dans la caractérisation précédente) un mot infini évitant S ; à l'inverse si on peut construire un tel mot alors comme à la caractérisation précédente un mot suffisamment long témoigne de l'existence d'un cycle dans ce graphe.

Question 6. Tout simplement $(aaa|bab|baab|bbb)^*$ d'après la question 3.

Question 7. Si une expression rationnelle est bornée, alors son langage est fini, et on peut simplement appliquer la question 5.

Question 8. On construit à partir de e l'expression rationnelle $\Sigma^* e \Sigma^*$ des mots qui ont un facteur dans e, i.e., ceux qui n'évitent pas e. On souhaite savoir si le complémentaire de ce langage est fini. Pour ce faire, on peut transformer e en un automate fini qui reconnaît le même langage avec les outils du cours, et complémenter cet automate (la clôture par complémentaire est également au programme). Il s'agit alors de déterminer si cet automate reconnaît un langage fini : après suppression des états inutiles, c'est le cas si et seulement s'il n'a pas de cycle.

On peut tester algorithmiquement tout cela, mais l'étape coûteuse est la complémentation : l'automate obtenu pour $\Sigma^* e \Sigma^*$ n'est généralement pas déterministe, et la clôture d'un automate implique de le complémenter d'abord, ce qui prend un temps exponentiel en général.

(Noter que cette construction généralise celle de la question 5.)

Question 9. On construit aisément l'expression rationnelle e_S des mots qui ont un facteur dans S, i.e., n'évitent pas S. On souhaite savoir si l'intersection de e avec le complémentaire de ce langage est finie. Pour ce faire, comme à la question 8, on construit à partit de e_S un automate A_S qu'on complémente en un automate A'_S (en le déterminisant au préalable), et on construit à partir de e un automate A. On teste alors si les langages de A et A'_S ont une intersection infinie en construisant un automate pour l'intersection (clôture par intersection au programme) et en testant comme à la question précédente.

Question 10. Soit $S \subseteq \Sigma^*$ inévitable. Soit n la longueur maximale d'un mot qui évite S. Ainsi, tout mot de longueur $n+1$ a un facteur dans S : soit S' l'ensemble de ces facteurs. C'est un ensemble fini, et il est inévitable puisque par définition tout mot de longueur $\geq n+1$ a un préfixe de longueur $n+1$ qui a un facteur dans S.

Question 11. En suivant la question précédente : à partir de l'automate construit à la question 8, on construit l'ensemble fini des mots dans le complémentaire L' du langage $\Sigma^* e \Sigma^*$ directement à partir de l'automate, on regarde leur longueur maximale n, puis on prend tous les mots de longueur $n+1$ et on leur trouve un facteur qui satisfait e. (En pratique il suffit de se limiter à un ensemble clos par préfixe des mots de longueur $n+1$, donc on peut notamment s'arrêter à tous les mots qui n'ont pas de continuation dans L', qu'on peut voir sur l'automate.)

A3 – Maintenance incrémentale de langages réguliers

On fixe un alphabet fini Σ. Un *mot* $w \in \Sigma^*$ est une suite finie d'éléments de Σ, et un langage $L \subseteq \Sigma^*$ est un ensemble de mots. Un langage est *régulier* s'il est reconnu par un automate fini, ou dénoté par une expression rationnelle (on admet que ces caractérisations sont équivalentes).

Question 0. Si l'on fixe un langage régulier $L \subseteq \Sigma^*$, le problème d'*appartenance à* L est de déterminer, étant donné en entrée un mot $w \in \Sigma^*$, si $w \in L$. Quelle est la complexité du problème d'appartenance à L en fonction de la longueur du mot d'entrée ?

Ce sujet s'intéresse à la *complexité incrémentale* du problème d'appartenance à un langage régulier L. Dans ce problème, on reçoit en entrée un mot $w \in \Sigma^*$ de longueur n. On effectue d'abord un *pré-traitement* pour déterminer si $w \in L$ et pour construire si on le souhaite une structure de données auxiliaire : cette phase de pré-traitement doit s'exécuter en $O(n)$. Ensuite, on reçoit des *mises à jour*, c'est-à-dire des paires (i, a) pour $1 \leq i \leq n$ et $a \in \Sigma$, données l'une après l'autre. À chaque mise à jour, on modifie le mot w pour que sa i-ème lettre devienne a, et on doit déterminer si $w \in L$ après cette modification. La longueur n du mot ne change jamais. La *complexité incrémentale* d'un langage est la complexité dans le pire cas pour prendre en compte une mise à jour, exprimée en fonction de n.

Question 1. Montrer que tout langage régulier a une complexité incrémentale en $O(n)$.

Question 2. Montrer que le langage régulier a^* sur l'alphabet $\Sigma = \{a, b\}$ a une complexité incrémentale en $O(1)$.

Question 3. Soit L_3 le langage des mots sur l'alphabet $\Sigma = \{a, b\}$ comportant au moins deux a, un nombre pair de a, et un nombre de b qui n'est pas divisible par 3. Ce langage est-il régulier ? Quelle est sa complexité incrémentale ?

Question 4. On dénote par $w_1, \ldots, w_n$ les lettres d'un mot $w \in \Sigma^*$ de longueur n. Pour toute permutation $\sigma \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, on écrit par abus de notation $\sigma(w)$ pour désigner le mot $w_{\sigma(1)} \cdots w_{\sigma(n)}$. Un langage L est *commutatif* si pour tout $w \in \Sigma^*$, pour n la longueur de w, pour toute permutation $\sigma \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$, on a $w \in L$ si et seulement si $\sigma(w) \in L$.

Montrer que tout langage régulier commutatif a une complexité incrémentale en $O(1)$.

Question 5. Proposer une structure de données pour stocker un ensemble d'entiers S qui supporte les opérations suivantes :

- Ajouter un entier dans S, en $O(1)$;
- Retirer un entier de S, en $O(1)$;
- Parcourir les entiers actuellement stockés dans S, en $O(|S|)$.

Écrire le pseudocode pour ces opérations.

Question 6. On considère le langage L_6 sur l'alphabet $\Sigma = \{a, b, c\}$ dénoté par l'expression rationnelle $c^*ac^*bc^*$. Montrer que sa complexité incrémentale est $O(1)$ en utilisant la structure de données de la question précédente.

Suite des questions

Question 7. À quelle classe de langages peut-on généraliser cette technique ?

Question 8. Soient deux langages L_1 et L_2 de complexité incrémentale en $O(1)$. Quelle est la complexité incrémentale des langages $L_1 \cap L_2$ et $L_1 \cup L_2$?

Question 9. On s'intéresse aux langages réguliers admettant au moins une "lettre neutre" (notion que le candidat ou la candidate aura dû identifier à la question 7). Proposer une classe aussi générale que possible de langages de complexité incrémentale en $O(1)$.

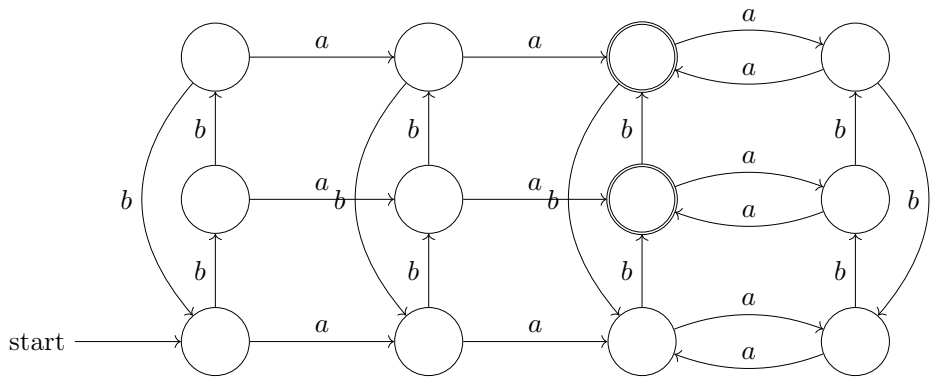
Corrigé

Question 0. On se donne un automate déterministe pour le langage L (on n'a pas besoin de se demander de la complexité de le calculer). Étant donné le mot w, on simule son exécution dans l'automate. La complexité est donc en $O(n)$. *[C'est surtout une question de cours, on attend juste $O(n)$ avec une phrase d'explication. On ne demande pas la complexité en fonction de l'automate ou autre représentation du langage.]*

Question 1. On stocke le mot w dans un tableau (ce qu'on fera toujours par la suite). Il suffit, à chaque mise à jour, de refléter la modification sur le tableau, puis de réévaluer si le mot courant appartient ou non au langage avec la question précédente.

Question 2. On maintient un compteur du nombre courant de a et du nombre courant de b, qu'on initialise au prétraitement. On maintient à jour ces compteurs à chaque mise à jour (ceci nécessite de connaître le nouveau caractère, et l'ancien caractère qui vient d'être changé, qu'on retrouve avec le tableau). Le mot courant est dans le langage si et seulement si le nombre de b est de 0.

Question 3. Le langage est régulier. On peut exhiber un automate :



On peut aussi remarquer plus intelligemment que c'est l'intersection de trois langages dont on montre facilement qu'ils sont réguliers (en exhibant un automate), or on sait d'après le cours que les langages réguliers sont clos par intersection.

Sa complexité incrémentale est en $O(1)$: on maintient un compteur du nombre de a et de b et on teste la condition.

Question 4. *Indication : considérer un automate déterministe et précalculer des chemins.*

On considère un automate déterministe pour le langage concerné. Dans le cadre du précalcul en $O(n)$, on calcule pour chaque état q de l'automate, pour chaque lettre $a \in \Sigma^*$, pour chaque entier

$0 \leq i \leq n$, la fonction $\delta^i(q, a)$ qui indique l'état auquel on aboutit après lecture de a^i depuis q. Ce calcul se fait de proche en proche : $\delta^0(q, a) := q$ et $\delta^{i+1}(q, a) := \delta(\delta^i(q, a), a)$ pour $\delta = \delta^i$ la fonction de transition de l'automate. Noter que le nombre d'états de l'automate et de lettres est une constante indépendante de la longueur n du mot, donc ce précalcul est bien en $O(n)$.

On maintient avec complexité incrémentale $O(1)$ un compteur du nombre d'occurrences de chaque lettre dans le mot, comme à la question précédente.

Pour savoir si le mot courant appartient à L, on utilise le fait que L est commutatif pour savoir que c'est le cas si et seulement si le mot $a_1^{n(a_1)} a_2^{n(a_2)} \cdots a_k^{n(a_k)}$ y appartient, où $a_1, \ldots, a_k$ désigne les lettres de Σ et $n(a)$ désigne le nombre d'occurrences dans le mot courant de la lettre a, qui est l'information maintenue par les compteurs. En effet, par définition des compteurs, le mot en question est une permutation du mot courant.

Or l'acceptation de ce mot par l'automate se détermine en $O(1)$ à l'aide des fonctions précalculées : on calcule $q_1 = \delta^{n(a_1)}(q_0, a_1)$, puis $q_2 = \delta^{n(a_2)}(q_1, a_2)$, et ainsi de suite, en k étapes où k est la taille de l'alphabet qui est une constante indépendante de n. On teste enfin si le dernier état obtenu est final ou non.

Question 5. *Indication : utiliser un tableau, mais s'inspirer de la structure de liste.*

Le problème est qu'une simple liste ne permet pas de retirer un élément identifié par sa valeur (il faut retrouver le maillon de liste qui le stocke), et qu'un tableau ne permet pas de parcourir efficacement les cases occupées. On pourrait naturellement concilier les deux, c'est-à-dire une structure de liste avec un tableau stockant des pointeurs vers les maillons. Pour que la suppression soit plus simple, on utiliserait spontanément des listes doublement chaînées.

Ceci dit, les pointeurs et les listes chaînées ne sont pas au programme, donc on attend plutôt une solution élémentaire à base de tableaux uniquement.

On stocke un tableau "suivant", un tableau "précédent", et une variable "début", avec l'invariant que les éléments actuellement stockés peuvent être parcourus comme début, suivant[début], suivant[suivant[début]], etc., jusqu'à atteindre la valeur -1.

```
début := -1
suivant := tableau d'entiers de taille n initialisé à -1
précédent := tableau d'entiers de taille n initialisé à -1

Fonction Énumérer():
  courant := début
  Tant que courant != -1:
    Produire courant
    courant := suivant[courant]
  Fin Tant que
Fin Fonction

Fonction Ajouter(x):
  // On ajoute au début
  Assertion(suivant[x] == précédent[x] == -1)
  suivant[x] := début
  // précédent[x] reste à -1
  Si début != -1:
    précédent[début] := x
  Fin Si
  début := x
Fin Fonction
```

```
Fonction Supprimer(x):
  // Si on supprime le premier élément, on change début
  Si début == x:
    début := suivant[x]
  Fin Si
  // On saute par dessus l'élément supprimé
  Si suivant[x] != -1:
    précédent[suivant[x]] := précédent[x]
  Fin Si
  Si précédent[x] != -1:
    suivant[précédent[x]] := suivant[x]
  Fin Si
  // Maintenant, on supprime
  suivant[x] := -1
  précédent[x] := -1
Fin Fonction
```

Noter que bien sûr les éléments ne sont pas parcourus dans l'ordre.

Question 6. On maintient une structure de données de la question 5 pour l'ensemble de positions contenant la lettre a, une autre pour l'ensemble des positions contenant la lettre b, en $O(1)$ par la question précédente. On maintient le nombre de a et de b comme aux questions 2–4, en $O(1)$.

Pour savoir si le mot courant est dans L, on vérifie d'abord s'il y a exactement un a et un b. Si non, alors la réponse est non. Si oui, alors on énumère le contenu des deux structures, ce qui est en $O(1)$ car elles contiennent chacune un élément. On trouve aussi la position de l'unique a et de l'unique b, et on les compare, pour savoir si on est ou non dans L.

Question 7. Intuitivement, si on peut partitionner l'alphabet Σ entre des lettres qui n'ont pas d'effet et des lettres sur lesquelles on doit réaliser un mot fini (ou un langage fini), alors la même technique s'appliquera.

Pour être précis (on attend une telle formalisation du candidat ou de la candidate) : on définit une lettre a qui est *neutre* pour L comme à la question 13. On définit le *réduit* de L pour une lettre neutre a comme le langage $L^{-a} := \{w^{-a} \mid w \in L\}$. On étend cette définition à un ensemble non-vide de lettres neutres : $L^{-\Sigma'}$ pour $\Sigma' \subseteq \Sigma$ non-vide est l'ensemble des $w^{-\Sigma'}$ pour $w \in L$ où $w^{-\Sigma'}$ s'obtient en retirant de w toutes les lettres de Σ'. Pour un langage L avec un ensemble non-vide Σ' de lettres neutres, si $L^{-\Sigma'}$ est fini, alors la complexité incrémentale est en $O(1)$ comme à la question précédente.

Pour le montrer, on crée une structure de données de la question 5 pour chaque lettre de $\Sigma \setminus \Sigma'$. On maintient ces structures, ainsi que le nombre d'occurrences de chaque lettre, en $O(1)$. Soit N la longueur maximale d'un mot de $L^{-\Sigma'}$; c'est une constante indépendante de n. Pour savoir si le mot courant appartient ou non à L, tant que le nombre d'occurrences total des lettres de $\Sigma \setminus \Sigma'$ est $> N$, on peut répondre non directement, en $O(1)$. Sinon, on utilise les structures de données de la question 5 pour identifier le sous-ensemble (de taille au plus N) de positions contenant des lettres de Σ'. On le trie en temps $O(N \log N)$, ce qui est toujours constant. On lit le mot formé et on teste (en temps constant car il est de taille $\leq N$) s'il appartient à $L^{-\Sigma'}$, ce qui conclut.

Question 8. Si on peut maintenir une structure avec complexité incrémentale en $O(1)$ pour l'appartenance à L_1, et pour l'appartenance à L_2, alors ces structures nous permettent de savoir si on appartient à L_1 et à L_2, ou à L_1 ou à L_2, toujours en $O(1)$.

Question 9. La réponse naïve (mais nécessitant un peu de recul) est : la clôture par intersections et par unions de la classe de la question 7 et des langages commutatifs couverts en question 4. Mais on s'attend à ce que le candidat ou la candidate sache se représenter la puissance d'expression de cette classe :

— L'intersection d'un langage de la question 7 avec un langage commutatif permet de poser des conditions commutatives sur les lettres neutres (qui ne sont du coup plus neutres), mais elle n'est pas intéressante sur les autres lettres (puisque le langage est fini)

— L'union de tels langages est intéressante parce qu'elle permet de changer la partition entre lettres neutres et non-neutres. En d'autres termes, on a une complexité incrémentale de $O(1)$ pour le langage "j'ai un nombre pair de a, une lettre neutre b, et exactement un c avant exactement un d, OU j'ai un nombre pair de c, une lettre neutre d, et exactement un a avant exactement un b".

— L'union permet aussi, pour la même partition, de poser des conditions commutatives différentes suivant ce qui est attendu pour les lettres non-neutres, par exemple couvrir des langages comme "j'ai une lettre neutre d, et j'ai exactement un a avant exactement un b et un nombre pair de c OU exactement un b avant exactement un a et un nombre impair de c.

Pour résumer, une forme normale serait : une union finie de langages qui sont l'entrelacement (shuffle) d'un langage commutatif sur un alphabet Σ', et d'un singleton (un seul mot) sur l'alphabet $\Sigma \setminus \Sigma'$. *[Cette caractérisation est celle des langages ZG, voir [AJP21, AP21]. On peut conjecturer qu'il s'agit des seuls langages réguliers de complexité incrémentale $O(1)$, ou en tout cas c'est le cas sous une certaine hypothèse de complexité.]*

Références

[AJP21] Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic Membership for Regular Languages. In *ICALP*, 2021.

[AP21] Antoine Amarilli and Charles Paperman. Locality and Centrality: The Variety ZG. Preprint : `http://arxiv.org/abs/2102.07724`, 2021.