On considère dans tout le problème un alphabet A. Un langage L est un ensemble de mots de A. On désigne par L^* l'ensemble des mots de A obtenus par concaténation de suites finies de mots de L, y compris le mot vide ε. Chaque mot entrant dans la concaténation est appelé facteur.
Ainsi, $L^* = \bigcup_{n\in\mathbb{N}} L^n$, avec $L^0 = \{\varepsilon\}$ et $L^{n+1} = \{w_1 w_2 \mid (w_1, w_2) \in L^n \times L\}$, pour tout $n \in \mathbb{N}$.
Les trois parties de ce problème tournent autour de la même question, tout en restant largement indépendantes. Le résultat abordé dans cette épreuve est :

Si un langage L est polynomial, alors L^* l'est aussi.

- Un *langage polynomial* est, par définition, un langage L tel qu'il existe un programme prenant en entrée un mot m de A^* et retournant un booléen indiquant l'appartenance ou non de m à L, en un temps majoré par $P(|m|)$, où P est un polynôme dépendant du programme, mais pas de l'entrée m. L'entier $|m|$ désigne la longueur de m, c'est-à-dire son nombre de lettres.
- Un *automate déterministe* est un quadruplet $\mathscr{A} = (Q, q_0, \delta, F)$ avec
  — Q est l'ensemble des états. Il est fini et non vide ;
  — q_0 est l'état initial ;
  — $\delta : Q \times A \to Q$ est la fonction de transition (éventuellement partielle)
  — $F \subset Q$ est l'ensemble des états terminaux.
  Lorsque δ est définie sur l'ensemble $Q \times A$ tout entier, on parle d'automate complet. Que l'automate soit complet, ou non, on peut choisir de représenter les transitions non pas par une fonction de transition δ, mais par un ensemble de transitions $T \subset Q \times A \times Q$ où $(p, x, q) \in T \iff \delta(p, x) = q$. L'ensemble T devant alors vérifier la condition de déterminisme
  $$\forall (p, q_1, q_2) \in Q^3, \forall x \in A, ((p, x, q_1) \in T \text{ et } (p, x, q_2) \in T) \Rightarrow q_1 = q_2. \quad \text{(D)}$$
- Un *automate non-déterministe* est un quadruplet $\mathscr{A} = (Q, I, \delta, F)$ avec cette fois $I \subset Q$ l'ensemble des états initiaux et $\delta : Q \times A \to \mathscr{P}(Q)$ la fonction de transition : si $p \in Q$ et $x \in A$, $\delta(p, x)$ désigne l'ensemble (éventuellement vide) des $q \in Q$ tels qu'il existe une transition étiquetée par x de p vers q. On peut encore représenter les transitions par une partie de $Q \times A \times Q$. La condition (D) n'a plus à être vérifiée.
- Un langage est déclaré *reconnaissable* lorsqu'il est le langage accepté par un automate fini (déterministe ou non)

Pour les programmes, les mots seront codées par des variables de type `string`.
— Si `m` est de type `string`, `m.[i]` désigne son i-ème caractère. Il est de type `char`.
— La fonction `String.length : string -> int` renvoie la longueur d'une chaine de caractères.
— Les chaines de caractères sont désignées par des guillemets `"toto"` et les caractères par des apostrophes `` `t` ``
— La fonction `String.sub : string -> int -> int -> string` permet d'extraire une sous chaine de caractère. Précisément, `String.sub ch d l` renvoie la sous-chaine de `ch` qui commence à l'indice `d` et de longueur `l`. Par exemple `String.sub "azertyuiop" 3 2` renvoie `"rt"`.
— La fonction `string_of_char : char -> string` transforme un caractère en la chaine de caractères correspondante.

Lorsque le candidat devra écrire un programme testant l'appartenance d'un mot à un langage donné, il pourra supposer que les mots donnés en entrée ont leurs lettres dans le bon alphabet : il est inutile de le vérifier.

L'énoncé contient 4 pages.

I - Quelques exemples

1. Cas des langages reconnaissables

   (a) Soit L un langage reconnaissable. Montrer qu'il est polynomial. On demande d'écrire un programme (par forcément en Caml ; du pseudo-code en langage naturel sera accepté) prenant en entrée un mot et évaluant son appartenance à L.

   On pourra supposer que l'on dispose d'un automate fini déterministe complet $A = (Q, q_0, \delta, F)$, d'état initial q_0, d'états terminaux F, avec des transitions données par une fonction de transition δ calculant chaque $\delta(q, x)$ en temps $O(1)$.

   Le caractère polynomial devra être justifié.

   (b) Soit L un langage reconnaissable et $\mathscr{A} = (Q, q_0, T, F)$ un automate fini déterministe reconnaissant L où Q est l'ensemble des états, q_0 l'état initial, F l'ensemble des états terminaux et $T \subset Q \times A \times Q$ les transitions. On considère l'automate $\mathcal{S} = (Q \cup \{i\}, i, T', F')$ où
   — i est un nouvel état,
   — F' vaut F si $q_0 \notin F$ et F' vaut $F' \cup \{i\}$ si $q_0 \in F$
   — T' est obtenu en ajoutant à T toutes les transitions (i, x, q) dès qu'il existe une transition (q_0, x, q) dans T.
   L'automate $\mathcal{S}$ est un automate déterministe *standard*, c'est-à-dire qu'il n'existe pas de transitions arrivant à l'état initial.

   Montrer que le mot vide est reconnu par $\mathscr{A}$ si et seulement s'il est reconnu par $\mathcal{S}$ puis que le langage reconnu par $\mathcal{S}$ est L.

   (c) Soit L un langage reconnaissable et, $\mathscr{A} = (Q, q_0, T, F)$ un automate fini déterministe standard reconnaissant L (qui existe d'après la question précédente).

   On considère l'automate $\mathscr{A}' = (Q, q_0, T', F')$ où $F' = F \cup \{q_0\}$ et T' est obtenu à partir de T en ajoutant des transitions (p, x, q) où $p \in F$ et (x, q) tels que $(q_0, x, q) \in T$.

   Justifier *proprement* que le langage accepté par l'automate $\mathscr{A}'$ est L^*.

   On vient donc de montrer, dans le cas des langages reconnaissables, que si L est polynomial alors L^* aussi.

2. Le cas de $L_0 = \{a^n b^n \mid n \in \mathbb{N}\}$

   (a) Montrer que ni L_0 ni L_0^* n'est reconnaissable.

   (b) Montrer que L_0 est polynomial en écrivant une fonction `recL0 : string -> bool` telle que `recL0 m` renvoie `true` si et seulement si le mot `m` appartient à L_0.

   (c) Montrer de même que L_0^* est polynomial.

3. Un autre exemple

   Dans cette question, l'alphabet est réduit à $A = \{a\}$ et on note $L_1 = \{a^{2^n} \mid n \in \mathbb{N}\}$.

   (a) Le langage L_1 est-il reconnaissable ? Donner un automate reconnaissant L_1 ou une preuve de sa non-reconnaissabilité.

   (b) Montrer que L_1 est polynomial.

   (c) Donner, en la justifiant, la valeur de L_1^*.

   (d) Le langage L_1^* est-il reconnaissable ? Polynomial ?

4. Un dernier exemple

   Dans cette question l'alphabet est $A = \{0, 1, \#\}$ et on suppose que l'on dispose d'une fonction φ codant les entiers en des mots de l'alphabet $\{0, 1\}$ (leur décomposition en base 2 sans 0 au début de mot sauf pour 0 de codage 0). Par exemple, $\varphi(10) = 1010$.

   On note alors $L_2 = \{\varphi(n_1)\#\varphi(n_2)\#\varphi(n_1 n_2)\# \mid n_1, n_2 \in \mathbb{N}\}$.

   Par exemple `11#100#1100#`$\in L_2$ car $3 \times 4 = 12$.

   (a) Montrer que L_2 est polynomial.

   (b) Le langage L_2 est-il reconnaissable ?

II - Trois algorithmes

5. Une énumération des parties de $[\![0, n-1]\!]$

   (a) Écrire une fonction `dec2 : int -> int -> int array` prenant en entrée deux entiers k et n, avec k compris entre 0 et $2^n - 1$ et calculant un tableau de n cases contenant la décomposition en base 2 de k (l'ordre de placement des bits est laissé au candidat mais devra être spécifié).

   (b) Expliquer comment utiliser la fonction précédente pour décrire toutes les sous-parties de $[\![0, n-1]\!]$.

   (c) Soit m un mot non vide. Expliquer comment associer à une partie non vide de $[\![0, p]\!]$ (p étant à préciser en fonction de $|m|$), une décomposition d'e m en concaténation de mots non vides. On traitera, pour illustrer l'explication, la décomposition :

   `centralesupelec = cent.rale.supelec`

   (d) Écrire une fonction `dansLetoile` prenant en entrée un mot `m` et testant son appartenance à L^*. La fonction prendra en argument une fonction `dansL` qui teste l'appartenance à L.

   On précisera les signatures des fonctions.

   (e) Quelle est la complexité temporelle de la fonction `dansLetoile` ? On distinguera les appels à `dansL` et les opérations arithmétiques standards (telles que les divisions euclidiennes, supposées de complexité constante).

6. Un algorithme récursif

   (a) Soit $n \geq 1$. Montrer que si $m_0, \ldots, m_{n-1}$ sont des lettres de A, alors le mot $m = m_0 \ldots m_{n-1}$ est dans L^* si et seulement si $m \in L$ ou bien il existe $k \in [\![0, n-2]\!]$ tel que $m_0 \ldots m_k \in L$ et $m_{k+1} \ldots m_{n-1} \in L^*$.

   (b) En utilisant la propriété précédente, écrire une fonction récursive `dansLetoile2` testant l'appartenance d'un mot à L^*. On garde les conventions de la question 5. d

   (c) Évaluer le nombre d'appels à `dansL` dans le pire des cas. On donnera un majorant, atteint dans un cas que l'on explicitera.

7. Une programmation dynamique

   Soit L un langage sur l'alphabet A. On considère un mot m de L formé de n lettres de A : $m = m_0 \ldots m_{n-1}$.

   On définit pour $0 \leq i \leq j \leq n-1$, le booléen $T_{i,j}$ valant `true` si le facteur $m_i \ldots m_j$ appartient à L^* et `false` sinon. Ainsi, $m \in L^*$ si et seulement si $T_{0,n-1}$ vaut `true`.

   (a) Soit $i \in [\![0, n-1]\!]$ comment connaitre la valeur de $T_{i,i}$?

   (b) Montrer que si $0 \leq i < j \leq n-1$ alors

   $$T_{i,j} = (m_i \ldots m_j \in L) \bigvee \left(\bigvee_{k=i}^{j-1} T_{i,k} \wedge T_{k+1,j} \right).$$

   (c) En déduire un algorithme pour calculer tous les $T_{i,j}$.

   (d) Programmer l'algorithme précédent en Caml.

   (e) Évaluer la complexité temporelle de cet algorithme. On s'intéressera d'une part au nombre d'accès à la fonction `dansL`, et d'autre part au nombre d'opérations élémentaires telles qu'un ou/et logique.

III - Utilisation d'un graphe

8. Structure de file

Dans cette partie nous aurons besoin d'une structure de file (FIFO).

`type fifo = {contenu : int vect ; mutable debut : int ; mutable fin : int};;`

les files ont une taille maximale imposée lors de la création. Les éléments de la file sont les éléments du tableau dont les indices sont, au sens large, entre `debut` et `fin`. Dans un premier temps, on pourra faire l'hypothèse que le nombre total d'entrées dans la file reste inférieur à la taille du tableau.

— Quand on entre un élément dans la file, on incrémente la valeur de `fin` et on place l'élément dans la case indexée par `fin`.
— Pour enlever un élément de la file, on lit la valeur de ma case indexée par `debut`, puis on incrémente cet index.

(a) Ecrire les fonctions `creerFile : int -> fifo` qui prend en entrée la taille du tableau qui contient la file et renvoie une file vide, `estVide : fifo -> bool` qui teste si une file est vide, `put : int -> fifo -> unit` qui ajoute un élément à la file et `get : fifo -> int` qui enlève un élément de la file et le renvoie.

(b) Comment modifier les fonctions précédentes si on suppose que le nombre d'éléments entrés dans la file peut dépasser la taille du tableau, mais qu'à chaque instant, le nombre d'éléments restant dans la file (ceux qui sont entrés dans la file et n'en sont pas sortis) reste inférieur à la taille du tableau ?

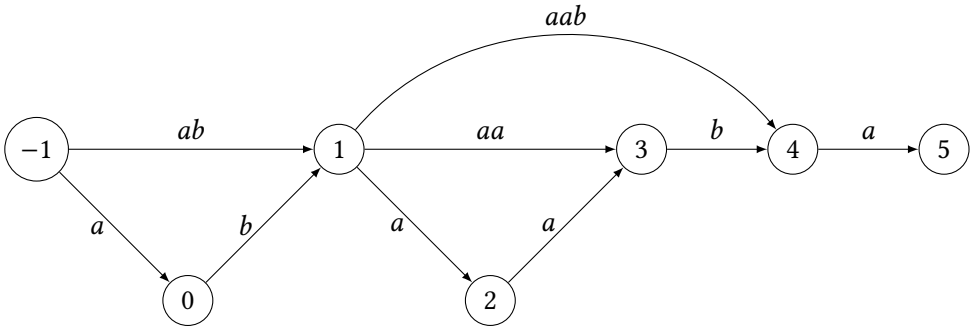
9. Introduction d'un graphe orienté

On appelle *graphe orienté* un couple (S, T) où S est un ensemble fini dont les éléments s'appellent les *sommets* et T est une partie de $S \times S$. Les éléments de T s'appellent les *arêtes*. Un *chemin* d'un sommet d vers un sommet f est une suite de sommets $s_0, s_1, \dots s_n$ telle que $s_0 = d$, $s_n = f$ et, pour tout $k \in \{0, 1, \dots, n-1\}$, $(s_k, s_{k+1}) \in T$.

Pour savoir si un mot $m = m_0, \dots, m_{n-1} \in A^*$ appartient à L^*, on va considérer le graphe $G_L(m) = (S, T)$ défini par
— $S = [\![-1, n-1]\!]$ comme ensemble de sommets ;
— $T = \{(i, j) \mid i < j \text{ et } m_{i+1} \dots m_j \in L\}$ comme ensemble d'arêtes.

Les arêtes de ce graphe permettent de localiser les facteurs de m qui appartiennent à L. L'appartenance de m à L^* est alors équivalente à l'existence d'un chemin de -1 vers $n-1$ dans ce graphe. On ne demande pas de justifier ce fait.

Exemple : Soit $A = \{a, b\}$ et $L = \{a^i b^j \mid (i, j) \in \mathbb{N}^2\}$. Dans ces conditions le graphe $G_L(abaaba)$ est



(a) Représenter $G_{L_0}(aabbaba)$ où L_0 est le langage de la question 2.

(b) Écrire une fonction `contruitGraphe : string -> (string -> bool) -> int list vect` telle que `construitGraphe mot dansL` renvoie le graphe ci-dessus où `mot` est le mot considéré et `dansL` est une fonction testant l'appartenance à L comme dans la partie II. Le graphe sera codé par des listes d'adjacence.

(c) En déduire une fonction `dansLetoile4 : string -> (string -> bool) -> bool` testant l'appartenance d'un mot à L^*. On réalisera, en utilisant une file, un parcours en largeur du graphe obtenu à la question précédente.

(d) Déterminer la complexité temporelle de la fonction `dansLetoile4`.