*Concours Centrale - Supélec 2010*

# Épreuve : INFORMATIQUE — Filière MP

***Calculatrices autorisées***

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 $\varepsilon$. Chaque mot entrant dans la concaténation est appelé facteur. En particulier si $m$ est un mot, $m^k = \underbrace{m \dots m}_{k \text{ fois}}$.

Ainsi $L^* = \bigcup_{n \in \mathbb{N}} L^n$, avec $L^0 = \{\varepsilon\}$ et $L^{n+1} = \{w_1 w_2 | (w_1, w_2) \in L^n \times L\}$, pour tout $n \in \mathbb{N}$.

Les trois parties de ce problème tournent autour d'une 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 (Caml, ou Pascal...) 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. On pourra admettre que si $P$ est un polynôme réel tendant vers $+\infty$ en $+\infty$, alors il existe un autre polynôme réel $Q$ tel que $Q$ est croissant sur $\mathbb{R}^+$, $P(t) \leqslant Q(t)$ pour tout $t \geqslant 0$, avec de plus $P(t) \sim Q(t)$ lorsque $t$ tend vers $+\infty$.

Un langage est déclaré *reconnaissable* lorsqu'il est le langage accepté par un automate fini.

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.

- version du 9 mars 2010 15h12

## *Partie I - Quelques exemples*

### I.A - Cas des langages reconnaissables

I.A.1) Soit $L$ un langage reconnaissable. Montrer qu'il est polynomial. On demande d'écrire un programme (pas forcément en Pascal ou Caml ; du pseudo-code en langage naturel sera accepté) prenant en entrée un mot et évaluant son appartenance à $L$.

On pourra supposer qu'on dispose d'un automate fini déterministe complet, d'état initial $q_i$, d'états finaux $F$, avec des transitions données par une fonction de transition $\delta$ calculant chaque $\delta(q, \alpha)$ en temps $O(1)$.

Le caractère polynomial du temps d'exécution devra être justifié

I.A.2) Soit $L$ un langage reconnaissable. Montrer que $L^*$ est également reconnaissable. On construira explicitement un automate **sans** transition instantanée (appelée aussi $\varepsilon$-transition) reconnaissant $L^*$.
*Les deux questions précédentes prouvent le théorème dans le cas particulier d'un langage reconnaissable.*

I.A.3) Évaluer le coût en temps et espace de la construction choisie d'un automate reconnaissant $L^*$ à partir d'un automate reconnaissant $L$.

### I.B - Le cas de $L_0 = \{a^n b^n \mid n \in \mathbb{N}\}$

I.B.1) Montrer que ni $L_0$ ni $L_0^*$ n'est reconnaissable.

I.B.2) Montrer que $L_0$ et $L_0^*$ sont polynomiaux.

*On écrira explicitement un programme Pascal ou Caml reconnaissant $L_0$ (resp. $L_0^*$).*

### I.C - Un autre exemple

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

I.C.1) $L_1$ est-il reconnaissable ? Donner un automate reconnaissant $L_1$ ou une preuve de non-reconnaissabilité de $L_1$.

I.C.2) Prouver que $L_1$ est polynomial.

I.C.3) Donner en la justifiant la valeur de $L_1^*$.

I.C.4) $L_1^*$ est-il reconnaissable ? polynomial ?

**I.D - Un dernier exemple**

Dans cette section **I.D**, l'alphabet est $A = \{0, 1, \#\}$ et on suppose qu'on dispose d'une fonction $\varphi$ codant les entiers en des mots sur l'alphabet $\{0, 1\}$ (leur décomposition en base 2, sans 0 en début de mot, sauf pour 0, de codage `0`). Par exemple, $\varphi(10) =$`1010`. On note cette fois

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

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

I.D.1) Montrer que $L_2$ est polynomial.

I.D.2) $L_2$ est-il reconnaissable ?

**I.E - Une petite variation**

Si $L$ est un langage, on lui associe un nouveau langage :

$$\psi(L) = \{w^n \mid w \in L, \text{ et } n \in \mathbb{N}\}.$$

I.E.1) Montrer l'inclusion : $\psi(L) \subset L^*$. Donner un exemple de langage $L$ pour lequel cette inclusion est stricte.

Dans les trois questions suivantes, on demande une preuve ou un contre-exemple rapidement justifié :

I.E.2) « $L$ est reconnaissable » implique-t-il « $\psi(L)$ est reconnaissable » ?

I.E.3) « $L$ est polynomial » implique-t-il « $\psi(L)$ est polynomial » ?

I.E.4) « $L$ est reconnaissable » implique-t-il « $\psi(L)$ est polynomial » ?

## *Partie II - Trois algorithmes*

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

II.A.1) Écrire une fonction prenant en entrée deux entiers $k$ et $n$, avec $k$ compris entre 0 et $2^n - 1$ et calculant un vecteur (ou un tableau) contenant la décomposition en base 2 de $k$ (l'ordre de placement des bits est laissé au candidat).

II.A.2) Expliquer comment utiliser la fonction précédente pour décrire toutes les sous-parties de $[\![0, n-1]\!]$.

II.A.3) Expliquer comment associer à une partie non vide de $[\![0, p]\!]$ ($p$ étant à préciser), une décomposition d'un mot en concaténation de mots non vides.

On traitera, pour illustrer l'explication, la décomposition :

```
centralesupelec=cent.rale.supelec
```

II.A.4) Écrire une fonction `Dans_L_etoile` prenant en entrée un mot $m$ et testant l'appartenance de $m$ à $L^*$ :

• **en Caml**, la fonction (dont on donnera le type) prendra en argument une fonction `Dans_L` testant l'appartenance d'un mot à $L$.

• **en Pascal**, on supposera qu'on dispose d'une fonction globale `Dans_L` prenant en argument un mot (de type `string`) et retournant un booléen.

II.A.5) Quelle est la complexité temporelle de la fonction `Dans_L_etoile` ? On distinguera les appels à `Dans_L`, et les opérations arithmétiques standards (telles que les divisions euclidiennes, supposées de complexité constante).

II.A.6) Si la longueur des mots est supérieure à quelques dizaines, la méthode précédente fait intervenir de trop gros entiers. Comment l'adapter pour énumérer toutes les parties de $[\![0, n-1]\!]$ sans manipuler des entiers de l'ordre de $2^n$ ? La complexité globale est-elle alors détériorée ?

**II.B - Un algorithme récursif**

II.B.1) Prouver la proposition suivante : « Si $w_0, w_1, \ldots, w_{n-1}$ sont des lettres de l'alphabet $A$, alors le mot $w = w_0 ... w_{n-1}$ est dans $L^*$ si et seulement si $w \in L$ ou bien il existe $k \in [\![0, n-2]\!]$ tel que $w_0 ... w_k \in L$ et $w_{k+1} ... w_{n-1} \in L^*$ ».

II.B.2) En utilisant la propriété précédente, écrire un programme `Dans_L_Etoile2` testant l'appartenance d'un mot à $L^*$, avec les conventions de la question II.A.4.

II.B.3) Évaluer le nombre d'appels à `Dans_L`, dans le pire cas. On donnera un majorant, atteint dans un cas qu'on explicitera.

**II.C - Une programmation dynamique**

Soit $L$ un langage sur l'alphabet $A$. On considère un mot $m$ de $L$ formé de $n$ lettres de $A$, $m = m_0 ... m_{n-1}$, avec $m_i \in A$ pour tout $i \in [\![0, n-1]\!]$.

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

II.C.1) Soit $i \in [\![0, n-1]\!]$. Que vaut $T_{i,i}$ ?

II.C.2) Montrer que si $0 \leqslant i < j \leqslant n-1$, alors :

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

($\vee$ désigne le *ou* logique, $\wedge$ le *et*).

II.C.3) En déduire un algorithme pour calculer tous les $T_{i,j}$.

II.C.4) Programmer l'algorithme précédent, en écrivant un programme Pascal ou Caml déterminant si $m \in L^*$, en calculant tous les $T_{i,j}$.

II.C.5) Évaluer la complexité temporelle de cet algorithme. On s'intéressera d'une part au nombre d'accès à la fonction testant l'appartenance à $L$, et d'autre part au nombre d'opérations élémentaires telles qu'un ou/et logique.

II.C.6) Évaluer la complexité spatiale de cette solution. Comparer avec les deux autres solutions proposées dans cette partie.

# *Partie III - Utilisation d'un graphe*

### III.A - Structure de file

Dans cette partie, nous aurons besoin d'une structure de données appelée file, ou encore « structure FIFO (pour *first in first out*) » : on entre les éléments les uns après les autres dans la file, et lorsqu'on les sort, le premier élément enlevé est le premier qui avait été entré. On fait le choix de la structure de données suivante :

- **en Caml**

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

- **en Pascal**

```
const nmax=1000;
type
   fifo = record
             contenu : array[1..nmax] of integer;
             debut, fin : integer ;
          end;
```

Les files ont une taille maximale imposée (à la création en Caml, globalement via `nmax` en Pascal). Les éléments de la file sont tous les éléments du vecteur/tableau dont les indices sont entre `debut` et `fin` (au sens large).

Quand on entre un élément dans la file, on incrémente la valeur de `fin` et on place le nouvel élément dans la case indexée par `fin` dans le tableau :

- **en Caml** fonction `put` de type `int -> fifo -> unit`,
- **en Pascal** procédure de signature : `procedure put(v:integer; var f:fifo))`.

Pour enlever un élément de la file, on lit la valeur de la case indexée par `debut`, puis on incrémente cet index :

- **en Caml** fonction `get` de type `fifo -> int`,
- **en Pascal** fonction de signature : `function get(var f:fifo):integer`.

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.

- **en Caml**, on suppose qu'on dispose d'une fonction de création de file `creer_file : int -> fifo`, prenant en entrée la taille du vecteur ;

et d'une fonction testant si la file est vide (cet état est caractérisé par le fait que l'indice de début a dépassé celui de fin) `est_vide : fifo -> bool`.

- **en Pascal**, on suppose qu'on dispose d'une procédure de création de file : `procedure creer_file(var f:fifo));`

et d'une fonction testant si la file est vide (cet état est caractérisé par le fait que l'indice de début a dépassé celui de fin) `function est_vide(f:fifo):boolean`.

III.A.1) Écrire des fonctions ou procédures `put` et `get` répondant aux spécifications données plus haut.

III.A.2) 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 ?

### III.B - Introduction d'un graphe orienté

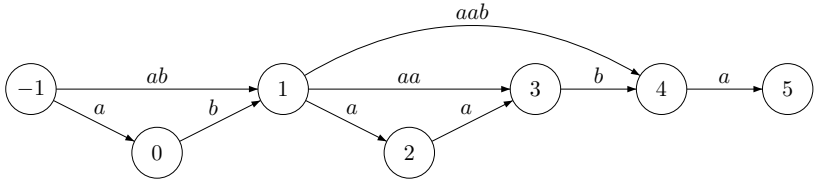
> On appelle *graphe orienté* le couple $(S, T)$ d'un ensemble fini $S$ d'éléments appelés *sommets* et d'un ensemble $T$ d'arêtes qui sont des éléments de $S \times S$. Dans un tel graphe, un *chemin* d'un sommet $d$ vers un sommet $f$ est une suite finie de sommets $s_0, s_1, \ldots, s_n$ telle que $s_0 = d$, $s_n = f$ et, pour tout $k \in \{0, 1, \ldots, n-1\}$, $(s_k, s_{k+1}) \in T$.

Pour savoir si un mot $w = w_0 w_1 \ldots w_{n-1}$ de $A^*$ appartient à $L^*$, on va considérer le graphe orienté $\mathcal{G}_L(w) = (S, T)$ défini par :

- $S = [\![-1, n-1]\!]$ comme ensemble de sommets ;
- $T = \{(i, j) \mid i < j \text{ et } w_{i+1} \ldots w_j \in L\}$ comme ensemble d'arêtes.

Les arêtes de ce graphe permettent de localiser les facteurs de $w$ qui appartiennent à $L$. L'appartenance de $w$ à $L^*$ est alors équivalente à l'existence d'un chemin de $-1$ vers $n-1$ dans ce graphe (ce qui revient au problème de l'accessibilité d'un état dans un automate fini, par exemple). On ne demande pas de justifier ce fait.

III.B.1) **Exemple :** Soit $A = \{a, b\}$ et $L = \{a\}^*\{b\}^* = \{a^i b^j \mid (i, j) \in \mathbb{N}^2\}$. Dans ces conditions le graphe orienté $\mathcal{G}_L(abaaba)$ est représenté par




Chaque arête $\alpha = (i, j)$ est indexée par le facteur $w_{i+1} \dots w_j \in L$ justifiant la présence de $\alpha$ dans le graphe.

Représenter le graphe $\mathcal{G}_{L_0}(aabbaba)$.

**On revient au cas général**.

Pour déterminer l'accessibilité de $n-1$ depuis $-1$, on peut effectuer un parcours en largeur du graphe : une file contient les sommets déjà détectés comme accessibles mais non encore traités ; dans un vecteur/tableau global, on tient à jour les états que l'on sait accessibles. Le pseudo-code suivant décrit l'algorithme :

```
entrer -1 dans la file
TANT QUE la file n'est pas vide et que n-1 n'a pas été vu :
   enlever un sommet s de la file
   POUR i allant de s+1 jusqu'à n-1 :
      SI (i n'a pas déjà été vu) et (w[s+1..i] appartient à L)
      ALORS
        vus[i]<-true
        entrer i dans la file
      FIN de si
   FIN de pour
FIN de tant que
```

On admet qu'à la fin de cette boucle, `vus[n-1]` vaut `true` si et seulement si $n-1$ est effectivement accessible depuis $-1$ dans le graphe.

III.B.2) Écrire une nouvelle fonction testant l'appartenance d'un mot à l'étoile d'un langage, en appliquant l'algorithme précédent.

III.B.3) Évaluer la complexité de ce programme dans le pire des cas (accès à la fonction d'appartenance, et opérations élémentaires).

III.B.4) Faire le bilan comparé des quatre algorithmes présentés dans ce problème. *On ira si possible au delà de « celui-ci est plus rapide que celui-là »...*

**• • • FIN • • •**