Mines 2017 - Option informatique

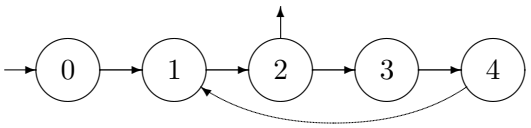
1 Langages et automates

On s'intéresse aux langages sur l'alphabet $\Sigma = \{a\}$; un tel langage est dit *unaire*. Un automate reconnaissant un langage unaire sera dit *unaire*. Lorsqu'on dessinera un automate unaire, il ne sera pas utile de faire figurer les étiquettes des transitions, toutes ces étiquettes étant l'étiquette a. C'est ce qui est fait dans cet énoncé.
Dans un automate unaire, on appelle *chemin* une suite $q_1, \ldots, q_p$ d'états telle que, pour i compris entre 1 et p, il existe une transition de q_{i-1} vers q_i ; on dit qu'il s'agit d'un chemin de q_1 à q_p. On appelle *circuit* un chemin $q_1, \ldots, q_p$ tel qu'il existe une transition de q_p vers q_1.
Dans cet exercice, tous les automates considérés seront finis et auront un et un seul état initial. On dit qu'un automate est *émondé* si, pour tout état q, il existe d'une part un chemin de l'état initial à q et d'autre part un chemin de q à un état final.
On rappelle qu'un langage non vide est rationnel si et seulement s'il est reconnu par un automate ou encore si et seulement s'il est reconnu par un automate déterministe émondé.

Soient α et β deux entiers positifs ou nuls. On note $L(\alpha, \beta)$ le langage unaire défini par :

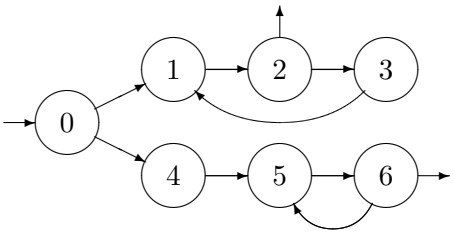
$$L(\alpha, \beta) = \{a^{\alpha k+\beta} / \ k \in \mathbb{N}\}$$

1. Donner sans justification une condition nécessaire et suffisante pour que $L(\alpha, \beta)$ soit fini. Dans le cas où cette condition est satisfaite, donner sans justification le cardinal de $L(\alpha, \beta)$.
2. On considère l'automate A_1 ci-dessous. Indiquer sans justification deux entiers α_1, β_1 tels que A_1 reconnaisse le langage $L(\alpha_1, \beta_1)$.



Automate A_1

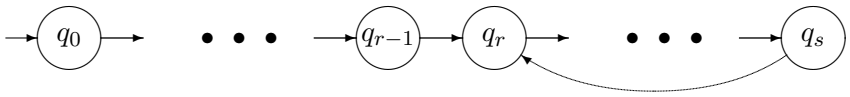
3. On considère l'automate A_2 ci-dessous :



Automate A_2

   On note L_2 le langage reconnu par A_2. Indiquer sans justification quatre entiers $\alpha_2, \beta_2, \alpha_3, \beta_3$ tels que A_2 reconnaisse le langage $L_2 = L(\alpha_2, \beta_2) \cup L(\alpha_3, \beta_3)$.
4. Construire un automate *déterministe émondé* A_3 en appliquant la procédure de déterminisation à l'automate A_2.
5. En s'appuyant sur l'automate A_3, indiquer sans justification cinq entiers $\alpha_4, \beta_4, \beta_5, \beta_6, \beta_7$ tel que A_3 reconnaisse le langage $L_3 = L(\alpha_4, \beta_4) \cup L(\alpha_4, \beta_5) \cup L(\alpha_4, \beta_6) \cup L(\alpha_4, \beta_7)$ (remarque : le langage L_3 est égal par ailleurs au langage L_2).

On dit ci-dessous qu'un automate est *de la forme F* si, en omettant les états finals, il peut se tracer selon le schéma ci-dessous :



Le chemin $q_0, \ldots, q_{r-1}$ peut être vide, auquel cas on a $r = 0$. Le circuit $q_r, \ldots, q_s$ ne doit pas être vide mais on peut avoir $r = s$ avec une transition de l'état q_r vers lui même (un tel circuit s'appelle aussi une *boucle*). On constate que les automates A_1 et A_3 sont de la forme F, mais non A_2.

6. Dessiner sans justification un automate de la forme F qui reconnaît le langage $L(1, 2)$. On fera figurer le ou les états finals.
   ATTENTION : on ne demande aucune justification mais uniquement de tracer un automate de la forme F en choisissant correctement les longueurs du chemin et du circuit et en ajoutant le ou les état(s) final(s).
7. Dessiner un automate de la forme F qui reconnaît le langage $L(2, 3) \cup L(5, 2)$. On fera figurer le ou les état(s) final(s). Comme à la question précédente, on ne demande aucune justification.
8. En s'inspirant de la réponse à la question précédente, décrire sans justification un automate de la forme F qui reconnaît le langage $L(2, 3) \cap L(5, 2)$. Indiquer deux entiers α et β tels que $L(2, 3) \cap L(5, 2) = L(\alpha, \beta)$.
9. Montrer qu'un automate déterministe émondé qui reconnaît un langage unaire rationnel infini est de la forme F. Donner une condition nécessaire et suffisante portant sur les états finals pour qu'un automate de la forme F reconnaisse un langage infini.
10. Soit L un langage rationnel unaire infini. En s'appuyant sur la question précédente, montrer qu'il existe deux entiers $\alpha \geq 1$ et $\beta \geq 0$ tels que L contient $L(\alpha, \beta)$.
11. On considère une suite $(u_n)_{n\geq 0}$ de nombres entiers positifs ou nuls. On suppose que la suite $(u_{n+1} - u_n)_{n\geq 0}$ est positive et strictement croissante. Soit L le langage défini par

    $$L = \{a^{u_n} / \ n \geq 0\}$$

    En utilisant la question précédente, montrer que L n'est pas rationnel.
12. Montrer que le langage L défini par $L = \{a^{n^2} / \ n \geq 0\}$ n'est pas rationnel.

2 Algorithmique et programmation

Préliminaires concernant la programmation

Il faudra coder des fonctions à l'aide du langage de programmation Caml, tout autre langage étant exclu. Lorsque le candidat écrira une fonction, il pourra faire appel à d'autres fonctions définies dans les questions précédentes ; il pourra aussi définir des fonctions auxiliaires. Quand l'énoncé demande de coder une fonction, il n'est pas nécessaire de justifier que celle-ci est correcte, sauf si l'énoncé le demande explicitement. Enfin, si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien vérifiées.
Dans les énoncés de l'exercice, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain (par exemple `n`).
On ne se préoccupera pas d'un éventuel dépassement du plus grand entier représentable.
On suppose disposer de deux fonctions

```
longueur : 'a list -> int
inverse : 'a list -> 'a list
```

donnant pour la première le nombre d'éléments d'une liste et pour la seconde une liste "miroir" de la première. Ainsi, `inverse [1;2;3]` est égal à la liste `[3;2;1]`.

REMARQUE : le sujet donnait aussi deux fonctions `empiler` *et* `dépiler` *agissant sur des références de listes. Ces fonctions me semblent inutiles voire troublantes pour la suite et je ne les ai pas reprises.*

On considère un ensemble U muni d'une loi de composition interne associative appelée *multiplication* et possédant un neutre pour cette loi noté e. Cette multiplication est notée avec le signe $\times$.
Par exemple, U peut être l'ensemble des entiers ou des réels munis de la multiplication usuelle, l'élément neutre étant 1. L'ensemble U peut aussi être l'ensemble des matrices carrées booléennes (ou d'entiers ou de réels) d'une même dimension d avec le produit usuel comme multiplication, l'élément neutre étant la matrice identité booléenne (ou entière ou réelle) de dimension d.
Soit $a \in U$ et soit $n \in \mathbb{N}$. On définit a^n de la façon suivante :

- $a^0 = e$
- si $n \geq 1$, $a^n = a^{n-1} \times a$.

La multiplication étant associative, si i et j sont deux entiers positifs ou nuls de somme n alors $a^n = a^i \times a^j$.
Un élément $a \in U$ et un entier $n \in \mathbb{N}^*$ étant donnés, on cherche à calculer a^n en s'intéressant au nombre de multiplications effectuées.
Dans toute la suite, a et n désignent respectivement un élément quelconque de U et un élément de $\mathbb{N}^*$.

Exemple 1 : si $n = 14$, on peut calculer a^{14} en multipliant 13 fois l'élément a par lui même. On effectue alors 13 multiplications.
Exemple 2 : si $n = 14$, on peut calculer a^{14} en calculant a^2 par $a^2 = a \times a$, puis a^3 par $a^3 = a^2 \times a$, puis a^6 par $a^6 = a^3 \times a^3$, puis a^7 par $a^7 = a^6 \times a$, puis enfin a^{14} par $a^{14} = a^7 \times a^7$. On aura ainsi obtenu le résultat en effectuant 5 multiplications.
Exemple 3 : si $n = 14$, on peut aussi calculer a^{14} en calculant a^2 par $a^2 = a \times a$, puis a^4 par $a^4 = a^2 \times a^2$, puis a^6 par $a^6 = a^4 \times a^2$ puis a^8 par $a^8 = a^4 \times a^4$, puis a^{14} par $a^{14} = a^8 \times a^6$. On aura ainsi obtenu le résultat en effectuant 5 multiplications.

L'objectif est de déterminer des algorithmes qui effectuent peu de multiplications. Soit x un nombre réel positif ; on note $\lfloor x \rfloor$ la partie entière par défaut de x et $\lceil x \rceil$ sa partie entière par excès.

On appelle *suite pour l'obtention de la puissance n* toute suite non vide croissante d'entiers distincts $(n_0, \ldots, n_r)$ telle que

- $n_0 = 1$
- $n_r = n$
- pour tout indice k vérifiant $1 \leq k \leq r$, il existe deux entiers i et j distincts ou non vérifiant $0 \leq i \leq k-1$, $0 \leq j \leq k-1$ et $n_k = n_i + n_j$ (la paire $\{i, j\}$ n'est pas forcément unique).

A une suite pour l'obtention de la puissance n correspond une suite de multiplications conduisant au calcul de a^n. Par exemple, la suite $(1, 2, 4, 6, 7, 12, 19)$ correspond au calcul de a^{19} en faisant les 6 multiplications suivantes : $a^2 = a \times a$, $a^4 = a^2 \times a^2$, $a^6 = a^4 \times a^2$, $a^7 = a^6 \times a$, $a^{12} = a^6 \times a^6$, $a^{19} = a^{12} \times a^7$.
Réciproquement, considérons un calcul de a^n dans lequel on fait en sorte d'ordonner les multiplications pour que les puissances calculées soient d'exposants croissants ; on peut associer à ce calcul une suite pour l'obtention de la puissance n.
A l'exemple 1 est associé la suite $(1, 2, 3, 4, 5, \ldots, 14)$ de longueur 14.
A l'exemple 2 est associé la suite $(1, 2, 3, 6, 7, 14)$ de longueur 6.
A l'exemple 3 est associé la suite $(1, 2, 4, 6, 8, 14)$ de longueur 6.

Le nombre de multiplications correspondant à une suite pour l'obtention de la puissance n est égal à la longueur de la suite diminuée de 1.

13. Montrer que tout calcul de a^n qui n'utilise que des multiplications nécessite un nombre de multiplications au moins égal à $\lceil \log_2(n) \rceil$. Donner une famille infinie de valeurs de n qui peuvent être calculées en effectuant exactement ce nombre de multiplications. Justifier la réponse.

On considère un algorithme appelé *par_division* ayant pour objectif le calcul de a^n. Cet algorithme s'appuie sur le principe récursif suivant :

si n vaut 1 alors a^n vaut a

sinon

- on calcule la partie entière par défaut, notée q, de $n/2$
- on calcule par l'algorithme *par_division* la valeur de $b = a^q$
- si n est pair, alors $a^n = b \times b$ et sinon $a^n = (b \times b) \times a$.

Ainsi, pour obtenir a^{14} l'algorithme *par_division* fait appel au calcul de a^7 qui fait appel au calcul de a^3 (pour obtenir a^6 en multipliant $a^"$ par a^3 puis a^7 en multipliant a^6 par a) qui fait appel au calcul de a^1 (pour obtenir a^2 puis a^3). Les différentes puissances calculées sont les puissances $1, 2, 3, 6, 7$ et 14. On constate que la suite pour l'obtention de la puissance 14 correspondant à l'algorithme *par_division* est la suite $(1, 2, 3, 6, 7, 14)$, de longueur 6. De même, la suite pour obtention de la puissance 19 correspondant à l'algorithme *par_division* est $(1, 2, 4, 8, 9, 18, 19)$ de longueur 7.

14. Calculer (sans justification) la suite correspondant à l'algorithme *par_division* successivement :
    - pour l'obtention de la puissance 15 ;
    - pour l'obtention de la puissance 16 ;
    - pour l'obtention de la puissance 27 ;
    - pour l'obtention de la puissance 125.

    Dans chaque cas, indiquer la longueur de la suite obtenue.
15. Ecrire en Caml la fonction `par_division : int → int list` qui calcule la suite de la puissance n correspondant à l'algorithme *par_division*.
16. Montrer que l'algorithme *par_division* appliqué à n effectue au plus $2 \times \lfloor \log_2(n) \rfloor$. Montrer que ce nombre est atteint pour un nombre infini de valeurs de n.

On considère un algorithme *par_décomposition_binaire* dont l'objectif est aussi le calcul de a^n. Cet algorithme utilise la décomposition d'un entier suivant les puissances de 2. L'algorithme est expliqué ci-dessous à l'aide d'exemples.

- Soit $n = 14$. On décompose 14 selon les puissances de 2 : $14 = 2 + 4 + 8$. On a donc : $a^{14} = (a^2 \times a^4) \times a^8$, ce qui conduit à calculer les puissances de a d'exposants $2, 4, 8$ mais aussi 6 et 14 ; la suite pour l'obtention de la puissance 14 correspondant à cet algorithme est la suite $(1, 2, 4, 6, 8, 14)$.
- Soit $n = 18$. On a $18 = 2+16$ et donc $a^{18} = a^2 a^{16}$. L'algorithme calcule les puissances d'exposant $2, 4, 8, 16$ puis 18 ; la suite pour l'obtention de la puissance 18 correspondant à cet algorithme est la suite $(1, 2, 4, 8, 16, 18)$.
- Soit $n = 101$. On a $101 = 1 + 4 + 32 + 64$. L'algorithme calcule a^{101} en utilisant les multiplications impliquées par la formule $a^{101} = ((a \times a^4) \times a^{32}) \times a^{64}$; on calcule les puissances $2, 4, 5$ (pour $a \times a^4 = a^5$), $8, 16, 32, 37$ (pour $a^5 \times a^{32} = a^{37}$), 64 et 101 (pour $a^{37} \times a^{64} = a^{101}$) ; la suite pour l'obtention de la puissance 101 correspondant à cet algorithme est la suite $(1, 2, 4, 5, 8, 16, 32, 37, 64, 101)$.

De manière générale, l'algorithme procède en écrivant la décomposition unique de n comme une somme de puissances croissantes du nombre 2, et calcule la valeur cible de a^n en effectuant les produits correspondant aux sommes partielles de cette somme.

17. Calculer (sans justification) la suite correspondant à l'algorithme *par_décomposition_binaire* successivement :
    - pour l'obtention de la puissance 15 ;
    - pour l'obtention de la puissance 16 ;
    - pour l'obtention de la puissance 27 ;
    - pour l'obtention de la puissance 125.

    Dans chaque cas, indiquer la longueur de la suite obtenue.

On considère la décomposition de n suivant les puissances croissantes du nombre 2 :

$$n = c_0 + c_1 \times 2 + \cdots + c_i \times 2^i + \cdots + c_k \times 2^k$$

où pour i vérifiant $0 \leq i < k$, le coefficient c_i vaut 0 ou 1 et c_k vaut 1.
On appelle *écriture binaire inverse* de n la suite $(c_0, c_1, \ldots, c_k)$.
Par exemple, l'écriture binaire inverse de l'entier 14 est $(0, 1, 1, 1)$, celle de l'entier 18 est $(0, 1, 0, 0, 1)$ et celle de l'entier 101 est $(1, 0, 1, 0, 0, 1, 1)$.

18. Ecrire en Caml une fonction `binaire_inverse : int → int list` qui à un entier $n \geq 1$ donné associe une liste correspondant à son écriture binaire inverse.
19. Ecrire en Caml la fonction `par_decomposition_binaire : int → int list` qui à un entier $n \geq 1$ donné associe une liste correspondant à l'algorithme *par_decomposition_binaire*.
20. On suppose que $n = 3^k$ où $k \in \mathbb{N}$. En utilisant la formule $3^k = 3^{k-1} + 2 \times 3^{k-1}$, montrer qu'il existe une suite pour l'obtention de la puissance n de longueur $2k + 1$. Indiquer la longueur de cette suite pour $n = 27$ et comparer avec le résultat obtenu par l'algorithme *par_division*.
21. Soit $k \in \mathbb{N}$. Ecrire en Caml une fonction `suite_3` qui à un entier n supposé être une puissance de 3 associe la liste de longueur $2k + 1$ évoquée en question précédente.
22. On suppose que $n = 5^k$ avec $k \in \mathbb{N}$. Montrer qu'il existe une suite pour l'obtention de la puissance n de longueur $3k + 1$. Indiquer la suite dans le cas $n = 125$ et comparer avec le résultat dans le cas de l'algorithme *par_division*.
23. Donner une suite de longueur 6 pour l'obtention de la puissance 15. Qu'en déduire quant aux algorithmes *par_division* et *par_decomposition_binaire* étudiés précédemment ?
24. On considère un tableau (ou vecteur) T, indicé à partir de 0, contenant une suite pour l'obtention d'une certaine puissance positive n. Soit k un entier compris entre 1 et la longueur de T diminuée de 1. Soit *val* la valeur contenue dans T à l'indicde k. On sait que *val* est est la somme de deux valeurs du tableau T situées à des indices (éventuellement confondus, éventuellement non uniques) strrictement inférieurs à k. Programmer une fonction `chercher_indice : int vect → int → int*int` qui étant donnés T et k calcule un couple (i, j) convenable (n'importe lequel). On supposera que k vérifie les contraintes exposées ci-dessus.
    Par exemple, si $T = [|1; 2; 3; 4; 7; 14; 17; 31|]$, la longueur du tableau est 8. Si $k = 6$, *val* vaut 17 et la fonction renvoie $(2, 5)$ (correspondant aux valeurs 3 et 14 du tableau). Si $k = 1$, la fonction renvoie $(0, 0)$.
25. Soit x un réel représenté par un élément de type `float`. Soit T un tableau contenant une suite pour l'obtention d'une certaine puissance positive n. Ecrire en Caml une fonction `puissance : float → int vect → float` qui prend en argument x et T et renvoie x^n en utilisant la tactique associée à la suite associée à T. Indiquer (sans justification) la complexité $C(k)$ de cette fonction. *Indications* :
    - on utilisera `chercher_indice` et en appelant h la longueur de T, la complexité de la fonction `puissance` devra être en $O(h \times C(h))$ (ce que l'on ne demande pas de justifier) ;
    - si `t` est un vecteur, `vect_length t` donne la longueur de `t` ;

- l'opérateur `*.` permet de faire le produit de deux valeurs de type `float`.

26. Décrire le principe d'un algorithme *suite_optimale* permettant d'exhiber une suite de longueur minimale pour l'obtention de la puissance n en effectuant une énumération exhaustive des suites possibles. Cette fonction fera appel à un algorithme récursif *suite_optimale_rec* dont on donnera aussi le principe.
27. Ecrire les fonctions `suite_optimale_rec` et `suite_optimale` correspondant aux algorithmes décrits.