# Corrigé DS 3

# Partie 1 : Mots de Lyndon et De Bruijn

### Partie 1 : Préliminaires

1. Cette question est assez simple si on sait que Ocaml permet de comparer les chaines de caractères, et qu'on remarque que l'ordre présenté est l'ordre lexicographique.

```
let inferieur u v =
  if u=v then 0
  else if u<v then -1
  else 1;;
```

C'est également une bonne chose de savoir le programmer depuis "from scratch" :

```
let inferieur u v =
  let n = String.length u in
  let m = String.length v in
  let i = ref 0 in
  let j = ref 0 in
  while !i<n && !j<m && u.[!i]=v.[!i] do (*On va a la premiere lettre différente*)
    i:=!i+1;
    j:=!j+1;
  done;
  if !i=n && !j=m then 0 (*On a atteint la fin des deux chaines. Il n'y a pas de lettre différente, u=v*)
  else if !i=n then -1 (*On a atteint la fin de u. u est un début de v*)
  else if !j=m then 1 (*On a atteint la fin de v. v est un début de u*)
  else if u.[!i]<v.[!j] then -1 (*On a une lettre différente, u<v*)
  else 1;; (*Pareil et u>v*)
```

2. Soit $u, v \in A^\star$ tel que $u \neq v$. On peut sans perte de généralité supposer que $|u| \geq |v|$. Considérons $u_1 ... u_{k-1}$ le plus long préfixe de $u$ qui est aussi préfixe de $v$ (égal à $\varepsilon$ si $k = 1$).
   Si ce préfixe est égal à $u$ alors $u$ est un début de $v$ donc $u \prec v$. Sinon, la lettre $u_k$ existe et on a $u_k \neq v_k$ par construction : si $u_k < v_k$ on a $u \prec v$ et si $u_k > v_k$, on a $v \prec u$.
   On vient de (re)démontrer que l'ordre lexicographique est total.
3. Il n'y a pas compatibilité par concaténation à droite : $0 \prec 00$ mais $01 \not\prec 001$.
4. Attention : les **`string`** en Ocaml ne sont pas mutables. Donc toutes méthode consistant à créer une chaine de caractère de la même taille que le mot puis changer ses lettres est vouée à l'échec.

```
let conjugue m i =
   let res = ref "" in
   for j = i to String.length m -1 do
     res:= !res ^ (String.make 1 m.[j])
   done;

   for j = 0 to i-1 do
     res:= !res ^ (String.make 1 m.[j])
   done;

   !res;;
```

5. Un mot de longueur $n$ a au moins un conjugué : lui-même. Il y a atteinte dès qu'on considère un mot dont toutes les lettres sont identiques (quel que soit l'alphabet). Un mot de longueur $n$ a au plus $n$ conjugués et on a atteinte par exemple dès que toutes les lettres du mot sont différentes (ce qui arrive dès que $A$ dispose de plus de $n$ lettres).
6. La relation $C$ est évidemment réflexive, symétrique et transitive.

### Partie 2 : Mots de Lyndon

7. Dans chacun des cas, on calcule la liste $L$ des conjugués du mot $u$ considéré et on vérifie que $u$ est le plus petit mot de $L$ selon l'ordre lexicographique.
   (a) 0010011 est un mot de Lyndon.
   (b) 010011 n'est pas un mot de Lyndon car 001101 en est un conjugué strictement plus petit.
   (c) 001001 n'est pas un mot de Lyndon car faire un décalage de 3 lettres vers la gauche de ce mot en donne un conjugué non trivial qui est égal à 001001 et pas strictement plus grand.

Gare à la définition de ce qu'est un conjugué trivial de $u$ : il n'y en a qu'un et c'est celui consistant à ne pas permuter circulairement les lettres de $u$. Tous les autres conjugués sont non triviaux, y compris quand — par chance — il forment le mot $u$.

8. Les mots de une lettre n'ont aucun conjugué non trivial : les merveilleuses propriétés de l'ensemble vide assurent donc qu'un tel mot est un mot de Lyndon.
9. Il suffit de comparer le mot en entrée à tous ses conjugués non triviaux. SI le mot est de taille 1, on peut directement répondre.

```
let lyndon m =
   let res = ref true in
   for i=0 to (String.length m -1) do
     res:=!res && (compare (conjugue m i) m = 0)
   done;
   !res;;
```

10. On utilise la stratégie suivante : on calcule la classe d'équivalence du premier (selon l'ordre lexicographique) mot de longueur 4 dont on ignore encore dans quelle classe il se trouve. On obtient 6 classes :

$$\{0000\}, \{0001, 0010, 0100, 1000\}, \{0011, 0110, 1100, 1001\}, \{0101, 1010\}, \{0111, 1110, 1101, 1011\}, \{1111\}$$

11. Pour trouver les mots de Lyndon de taille 4, il suffit de déterminer les colliers apériodiques parmi ceux de la question 10 puis de trouver le plus petit élément pour l'ordre lexicographique dans chaque.
    Les colliers apériodiques sont les deuxième, troisième et cinquième. Dans chacun, le plus petit élément est en fait le premier élément. On en déduit qu'il y a 3 mots de Lyndon de taille 4 : 0001, 0011 et 0111.
12. On peut utiliser une structure union-find.
    Si deux mots sont dans la même classe d'équivalence, ils sont seront dans la même classe de union-find.
    On peut utiliser le plus petit mot comme représentant de chaque classe, ce qui nous permet d'accéder directement aux mots de Lyndon.
13. Soit $n \in \mathbb{N}$ fixé.
    On initialise la structure union-find avec tous les mots de taille $n$.
    On parcourt ensuite tous les mots $m$ :
    On calcule chaque conjugé $c$ de $m$. Si $m$ et $c$ ne sont pas dans la même classe, alors on unit leurs classes.
    À la fin, la structure union-find représente les classes d'équivalence.
    (On pourrait gagner du temps en ne regardant pas tous les mots mais uniquement ceux qui sont seuls dans leur classe (et qui ne sont pas 0...0 ou 1...1))
14. On concatène 00111 à lui même pour obtenir un mot de taille 9 (il est nécessaire pour ce faire de tronquer la dernière lettre) ; on obtient le mot 001110011. On supprime tous les 1 en fin de ce mot ce qui donne 0011100. On remplace enfin le 0 final par un 1 et ainsi le mot ajouté à $E$ dans ces conditions sera 0011101.
15. On obtient dans cet ordre les mots 0, 0001, 001, 0011, 01, 011, 0111, 1. En observant les mots de taille 4 trouvés via cet algorithme, on constate que cette réponse est cohérente avec celle de la question 11.
16. Il s'agit d'expliquer pourquoi la boucle tant que extérieure (celle en ligne 4) termine. C'est le cas car $m$ croît strictement (pour l'ordre lexicographique) au fil des itérations de cette boucle et a toujours une taille bornée par $n$. Comme l'ensemble des mots de taille $n$ sur $\{0, 1\}$ est borné par $2^n$, il arrive un moment ou $m$ n'est constitué que de 1, et alors la boucle tant que intérieure supprime toutes les lettres de $m$ : à l'itération suivante, $m = \varepsilon$.
17. Les instructions de la boucle de la ligne 4 ont une complexité pire cas en $O(n)$ : on fait au pire (et en fait ce cas n'arrive jamais) $n$ ajouts de lettres à la ligne 5 puis $n$ suppressions de lettres à la ligne 7. Le raisonnement ci-dessus montre par ailleurs qu'on passe dans cette boucle au maximum $2^n$ fois. On obtient donc une complexité en $O(n2^n)$.

**Partie 3 : Factorisations de Lyndon**

18. On obtient la factorisation suivante : $(1)(01)(001011)(001)(0)(0)$.
19. La propriété suivante : "les mots $u^{(1)}, ..., u^{(k)}$ de la factorisation courante sont tous des mots de Lyndon" est un invariant de l'algorithme décrit par l'énoncé. C'est initialement vrai puisque les mots d'une lettre sont des mots de Lyndon (cf question 8).
    Si ça l'était avant une itération, cela le reste après puisque la nouvelle factorisation n'a fait que fusionner deux mots (par hypothèse) de Lyndon $u, v$ consécutifs et vérifiant $u \prec v$. L'énoncé assure que dans ces conditions $uv$ reste un mot de Lyndon et donc tous les mots de la nouvelle factorisation sont de Lyndon.
    En particulier, suite à la dernière itération tous les mots de la factorisation $(u^{(1)}, ..., u^{(k)})$ sont des mots de Lyndon et ces derniers vérifient $u^{(1)} \succeq ... \succeq u^{(k)}$ d'après la condition d'arrêt des itérations de l'algorithme $\mathcal{A}$. Cette factorisation est par définition une factorisation de Lyndon ce qui conclut.

20.
```
let extraire_chaine mot debut fin =
 let res = ref "" in
 for i=debut to fin do
   res:= !res^(String.make 1 mot.[i])
 done;

 !res;;
```

21. C'est la fonction non triviale de l'énoncé. On détaille un peu la conception de cet algorithme.
L'idée générale est de stocker en permanence deux entiers indiquant où se trouvent la fin du prochain mot à lire dans la factorisation et la fin du successeur de ce mot dans la factorisation. C'est le rôle des variables **fin_premier_mot** et **fin_deuxieme_mot**.
Connaître ces deux entiers permet d'extraire les deux prochains facteurs (**premier_mot** et **deuxieme_mot**) à étudier dans la factorisation. Une fois extraits à l'aide de **extrait_chaine**, on compare ces derniers à l'aide de **inferieur**. S'ils sont correctement triés, on passe au couple de facteurs suivants en mettant à jour **fin_premier_mot**, **fin_deuxieme_mot** et par conséquent **premier_mot** et **deuxieme_mot**. Sinon, on a trouvé un défaut de tri : c'est l'entier stocké dans **fin_premier_mot** !
Si après vérification de tous les couples de facteurs consécutifs, on constate qu'aucun ne viole la décroissance de la factorisation, on peut conclure que la factorisation donnée est correctement triée. Dans tous les cas, on n'oubliera pas de faire les libérations de mémoire nécessaires.

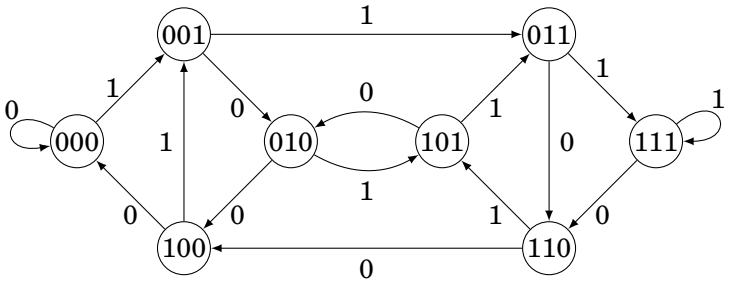
La première boucle tant que (qui termine car **facto** contient au moins un **true** puisqu'elle représente la factorisation d'un mot non vide d'après l'énoncé) et la première conditionnelle servent juste à récupérer la fin du tout premier mot et à s'interrompre au cas où la factorisation donnée ne contient en fait qu'un seul mot.

```
let est_trie mot facto =
 let res = ref true in
 let n = String.length mot in
 let fin_premier_mot = ref 0 in

 while (not facto.(!fin_premier_mot)) do
   incr fin_premier_mot
 done;

 if !fin_premier_mot = n-1 then (true,-1)
 else begin
   let premier_mot = ref (extraire_chaine mot 0 !fin_premier_mot) in
   let fin_deuxieme_mot = ref (!fin_premier_mot +1) in
   let i = ref (!fin_deuxieme_mot) in
   while !i<n && !res do
     if facto.(!i) then begin
       fin_deuxieme_mot := !i;
       let deuxieme_mot = extraire_chaine mot (!fin_premier_mot+1) (!fin_deuxieme_mot) in
       if inferieur !premier_mot deuxieme_mot <0 then res:=false
       else begin
         premier_mot := deuxieme_mot;
         fin_premier_mot:= !fin_deuxieme_mot;
         fin_deuxieme_mot:= !fin_deuxieme_mot+1
       end;
     end;
     i:=!i+1;
   done; (!res,!fin_premier_mot); end;;
```

22.
```
let factorisation mot =
 let n = String.length mot in
 let fin_de_mots = Array.make n true in
 let a,i = est_trie mot fin_de_mots in
 let b = ref a in
 let indice_pb = ref i in
 while not !b do
   fin_de_mots.(!indice_pb) <- false;
   let a,i = est_trie mot fin_de_mots in
   b:=a;
   indice_pb:=i
 done;

 fin_de_mots;;
```

**Partie 4 : Mots de de Bruijn**

23. Le mot $abba$ est un mot de de Bruijn d'ordre 2 sur $\{a, b\}$.
24. La longueur d'un mot de de Bruijn d'ordre $n$ sur un alphabet de taille $k$ est $k^n$. En effet, il y a bijection entre les positions des lettres d'un mot de de Bruijn et l'ensemble des mots de taille $n$ sur $A$ par construction.
25. On obtient le graphe suivant :

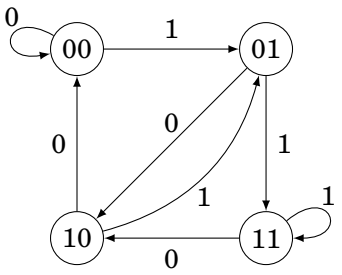



26. Soit $s$ un sommet de $B(k, n)$. Par construction d'un graphe de de Bruijn, il existe $a \in A$ et $m \in A^\star$ tel que (l'étiquette de) $s$ soit $am$ et la fonction :

$$\begin{cases} A \to \{\text{arêtes sortant de } am\} \\ b \mapsto (am, mb, b) \end{cases}$$

est une bijection claire, d'où on déduit qu'il y a autant d'arêtes sortant de $s$ que d'éléments de $A$, à savoir $k$. Le même raisonnement s'applique pour les arêtes entrant sur $s$ en remarquant que $s$ peut aussi s'écrire $mb$ avec $m$ un mot de $A^\star$ et $b$ une lettre de $A$. Ainsi, les degrés entrant et sortant de $s$ valent $k$.

27. Il suffit de compter le nombre d'arcs sortants de $B(n, k)$ pour conclure. Ce graphe a $k^n$ sommets, chacun de ses sommets est de degré sortant $k$ d'après 24 et on en déduit donc que $B(k, n)$ comporte $k^{n+1}$ arcs.
28. La méthode est la même que pour la question 23 :




Le chemin $(01, 10, 00, 00, 01, 11, 11, 10, 01)$ est un circuit eulérien dans $B(2, 2)$. L'étiquette de ce chemin est 00011101. Ce mot est un mot de de Bruijn d'ordre 3 (ce que la suite de l'énoncé confirme).

29. Soit $n \in \mathbb{N}^\star$ et $k \in \mathbb{N}$. Le graphe $B(k, n)$ est connexe : si $u = u_1 \dots u_m$ et $v = v_1 \dots v_l$ sont deux sommets, le chemin $(u_1 \dots u_m, u_1 \dots u_m v_1, u_2 \dots u_m v_1 v_2, \dots, u_m v_1 \dots v_{l-1}, v_1 \dots v_l)$ relie $u$ et $v$. De plus la question 24 montre que tout sommet de $B(k, n)$ voit son degré entrant et son degré sortant être égaux. Ainsi, $B(k, n)$ contient un circuit eulérien ce qui fournit un mot de de Bruijn d'ordre $n + 1$ d'après l'énoncé.
30. Pour répondre à cette question, il faut d'abord calculer les mots de Lyndon de taille 4, 2 et 1. Mais ceci a déjà été fait en question 13 : les mots convoités sont $0, 0001, 0011, 01, 0111, 1$ dans l'ordre lexicographique. En les concaténant, on en déduit que 0000100110101111 est le plus petit mot de de Bruijn d'ordre 4.
31. Il y a $k^n$ codes possibles, chacun de longueur $n$. A priori, il faut entrer $n \times k^n$ chiffres dans le digicode.
32. Plutôt que d'entrer successivement tous les codes, on entre un mot de de Brujin sur l'alphabet $[\![0, k-1]\!]$ d'ordre $n$ puis on réentre les $n-1$ premières lettres pour avoir les facteurs obtenus circulairement. Le mot de Bruijn contient toutes les séquences de $n$ chiffres de $[\![0, k-1]\!]$ ce qui garantit l'ouverture de la porte et ne nécessite de taper "que" $k^n + n$ chiffres d'après la question 24.