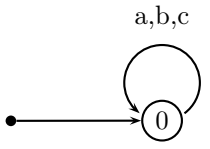
Corrigé feuille d'exercices d'informatique n°9

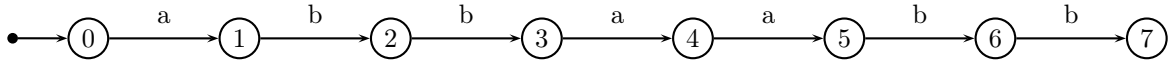
I.1

```
(* première solution en utilisant une exception *)
exception trouve of int;;

let chercher (M,T) =
try
let m = string_length M and n = string_length T and ok = ref true in
    for i = 0 to n-m do
      ok := true;
      for j = 0 to (m-1) do
          if T.[i+j] <> M.[j] then ok := false;
      done;
      if !ok then raise (trouve i)
    done;
-1
with trouve p -> p;;

chercher ( "abbaabb","cabcabbaabbcba");;

chercher ( "abbaabb","abbcaabbca");;

(* deuxième solution sans utiliser d'exception *)
let chercher (motif,texte) =
let m = string_length motif and n = string_length texte
     and trouvé = ref false and s = ref 0 in
while (!s <= n - m) && not !trouvé do
          let k = ref 0 in
          while (!k <= m-1) && (texte.[!s + !k] = motif.[!k]) do incr k done;
          if !k = m then trouvé := true else incr s
     done;
     if !trouvé then !s+1 else 0;;
```

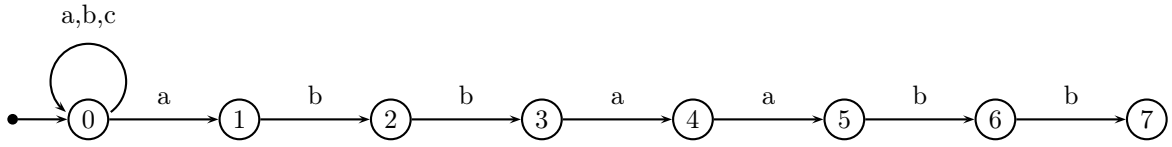
I.2 Dans le pire des cas, la boucle externe est exécutée $(n - m + 1)$ fois et la boucle interne $(m - 1)$ ce qui nous donne une complexité en $O(n.m)$.

II.1 Il suffit de concaténer 

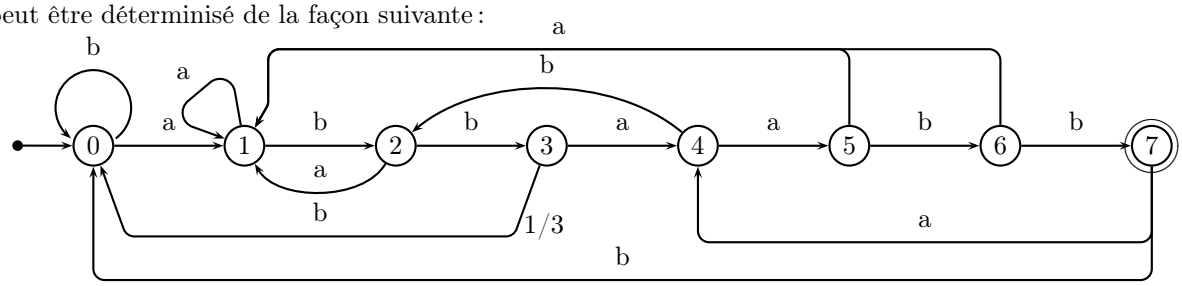
qui reconnaît X^* et



qui reconnaît M (les transitions de rebut ne sont représentées).



qui peut être déterminisé de la façon suivante :



II.2 Pour reconnaître toutes les positions de M dans T, il suffit de faire :

```
. j ← p_0
. POUR i allant de 1 à n FAIRE
.       j ← δ(j, T_i)
.       SI j = p_1 ecrire "motif reconnu en position" i − m + 1
. FINPOUR
```

En supposant que le calcul de $\delta(j, T_i)$ se fait à temps constant, la complexité est en $O(n)$.

III.1 fonction de saut pour le motif $M =$ "*abbaabb*".

k	$M_1 \dots M_k$	$f(k)$
1	a	0
2	ab	0
3	abb	0
4	abba	1
5	abbbaa	1
6	abbaab	2
7	abbaabb	3

III.2 Supposons que $M_1 M_2 \dots M_k = T_s T_{s+1} \dots T_{s+k-1}$ et posons $s' = s + k - 1$. Supposons que k est le plus grand possible, c'est à dire, $M_{k+1} \neq T_{s+k}$.
Pour $j = f(k)$, $M_1 \dots M_j$ est le plus grand préfixe de M qui soit aussi suffixe de $M_1 M_2 \dots M_k$. En mettant M_1 en correspondance avec T_{s+k-j}, on mettra en coïncidence le préfixe $M_1 M_2 \dots M_j$ de M et le mot $T_{s+k-j} \dots T_{s+k-1}$ de T.

En résumé : Si $M_1 \dots M_k$ est un préfixe de $T_s \dots T_n$ de $k < m$ et $T_{s+k} \neq M_{k+1}$, il faut essayer la coïncidence de M avec $T_{s+k-f(k)} \dots T_?$ par comparaison de M_{j+1} et T_{s+k} ($j = f(k)$) (cela revient à faire «glisser» le motif de $k - f(k)$ caractères vers la droite)

III.3.1 Validité de l'algorithme

Par récurrence sur j on a $g(j) \leqslant j$.
c'est vrai pour $j =$ et si on suppose la relation vraie pour $j = 1$:

- La boucle `TANT QUE` de la ligne 4 est finie puisque, au départ $i = g(j-1) \leqslant j - 1$ et il y aura au plus $g(j-1)$ passages.
Dans cette boucle, on a toujours $i \leqslant j - 1$.
- l'affectation d'une valeur à $g(j)$ en lignes 6 et 7 montre que $g(j) \leqslant i + 1 \leqslant j$.

Ce raisonnement établit également que l'algorithme est fini : il y a $m - 1$ itérations de la boucle `POUR` er, pour chacune d'elles, au plus $m - 1$ itérations (car $g(j_1) \leqslant j - 1 \leqslant m - 1$) de la boucle `TANT QUE`.

caractérisation de la fonction g
Soit $i_k = g^k(j-1)$ la $k^{ième}$ itération de $j - 1$ par g. Alors,

- s'il existe un entier p tel que $M_j = M_{1+i_p}$, k le plus petit de ces entiers, alors $g(j) = 1 + i_k$
- sinon, $g(j) = 0$

En effet, dans le 1^{ier} cas, la boucle `TANT QUE` (ligne 4) trouve le plus petit entier vérifiant la relation et la ligne 7 montre que $g(j) = 1 + i_k$.
sinon la boucle `TANT QUE` s'arrête avec $i = 0$ et grâce à 6 on a bien $g(j) = 0$
la fonction g est égale à la fonction f
...

```
(* fonction qui calcule la fonction de saut d'un motif *)
(* renvoie un tableau contenant les valeurs de la fonction de saut *)
(* remarque : saut.(0) n'est pas utilisé *)
(* fonction_de_saut : string -> int vect = <fun> *)
let fonction_de_saut motif =
let m =  string_length motif in
    let saut = make_vect (m+1) 0 and i = ref 0 in
saut.(1) <- 0;
for j = 2 to m do
    i := saut.(j-1);
    while (motif.[j-1] <> motif.[!i]) && (!i>0) do i := saut.(!i) done;
    if (motif.[j-1] <> motif.[!i]) && (!i=0)
         then saut.(j) <- 0
         else saut.(j) <- !i+1;
    done;
saut;;

fonction_de_saut "abbaabb";;
(*- : int vect = [|0; 0; 0; 0; 1; 1; 2; 3|] *)
```

III.3.2 complexité

Le raisonnement par récurrence de la question précédente a montré que pour chaque valeur de j, on fait au plus m comparaisons de lettres.
Il est donc simpliste de dire que la complexité est un $O(m^2)$
En fait il est facile de voir que plus la boucle `TANT QUE` est longue, plus petite sera la valeur attribuée à $g(j)$ de sorte qu'à l'itération suivante, la boucle `TANT QUE` sera fort raccourcie.
Plus précisément, soit i_j la valeur de i lors de la $j^{ième}$ exécution de `POUR`, et N_j le nombre de passages dans la boucle `TANT QUE`.
on a : $i_2 = 0$ $N_2 = 0$ et de manière évidente, $i_{j+1} \leqslant i_j - N_j + 1$ (il y a N_j affectations de i à une valeur strictement inférieure et au plus une incrémentation par l'intermédiaire de la ligne 7)
Puisque $0 \leqslant i_m - N_m$ il vient:

$$\sum_{2 \leqslant j \leqslant m} N_j \leqslant \sum_{2 \leqslant j \leqslant m-1} (i_j - i_{j+1} + 1) + i_m = i_2 + m - 2 = m - 2$$

On a finalement, au plus $m-2+(m-1)$ comparaisons (il y a $m-1$ exécution de la ligne 5) et l'algorithme de calcul de f est de complexité $O(m)$.

III.4.1 Pour $i = m$ à la ligne 3, le calcul de $X \setminus \{M_{i+1}\}$
On peut remplacer les lignes 3 et 4 par :

3' `POUR` i `allant de 1 jusque` $m-1$`,`
4' `POUR chaque lettre` x `de` $X \setminus \{M_{i+1}\}$ `FAIRE` $\delta(i,x) \leftarrow \delta(f(i),x)$ `FINPOUR`
5' `FINPOUR`
6' `POUR chaque lettre` x `de` X `FAIRE` $\delta(m,x) \leftarrow \delta(f(m),x)$ `FINPOUR`

III.4.2

État	f	a	b	c
0		1	0	0
1	0	1	2	0
2	0	1	3	0
3	0	4	0	0
4	1	5	2	0
5	1	1	6	0
6	2	1	7	0
7	3	4	0	0

Table des transitions remplie suivant l'algorithme :

. La ligne 1 permet d'obtenir les nombres soulignés
. la ligne 2 les 0 de la première ligne de la table
. les lignes 3 et 4 permettent d'obtenir les autres termes par recopie de la ligne "$f(i)$" sur la ligne "i".