Table des matières

1 Partie jouée sur le tablier européenen un minimun de coups

1. À titre préliminaire nommer le type de parcours de graphe le plus approprié pour déterminer un chemin entre deux sommet qui minimise le nombre d'arcs traversés. Quelle politique de mire en attente de sommets caractérise ce parcours ?

   solution : Le parcours le plus approprié est un parcours en largeur, obtenu en mettant les sommets en attente dans une file d'attente.

1.1 Numérotation du tablier

1. Écrire une fonction `numero_interieur (z:int): bool` qui teste si un entier naturel z est le numéro d'une encoche du tablier européen $\mathcal{E}$.
   solution :

```
let numero_interieur z =
  let x, y = z mod 8, z / 8 in
  0 <= x && x<= 6
  &&0 <= y && y<= 6
  && abs (x-3) + abs (y-3) <= 4
;;
```

2. En énumérant les entiers naturels inférieurs à 52 et à l'aide de la fonction `numero_interieur`, définir une constante `numeros_europeens` de type `int list` égale à la liste des numéros d'encoches du tablier européen.
   solution :

```
let rec intervalle a b =
  if a>b then [] else a::intervalle (a+1) b
;;
let numeros_europeens =
  List.filter numero_interieur (intervalle 0 53)
;;
```

3. En supposant qu'elles existent, donner par une expression mathématique le numéro des quatre encoches voisines de l'encoche de numéro z.
   solution : Ce sont les encoches de numéros $z+1$, $z-1$, $z+8$ et $z-8$ (dans l'ordre droite, gauche, haut, bas).

1.2 Motifs

1. Écrire une fonction `numero_vers_ponctuel (z:int): ponctuel` qui renvoie le motif ponctuel $\{z\}$ formé d'une seule encoche de numéro z.
   solution :

```
let numero_vers_ponctuel z : ponctuel =
  1 lsl z;;
```

2. Écrire une fonction `numeros_vers_motif (l: int list): modif` qui renvoie le motif formé des encoches de numéro appartenant à la liste l.
   solution :

```
let  numeros_vers_motifs l : motif =
  List.fold_left
    (lor)
    0
    (List.map numero_vers_ponctuel l)
;;
```

3. En s'appuyant sur la constante `numeros_europeens`, définir une constante globale `motif_europeen`, de type motif, égale au motif formé de toutes les encoches du tablier européen.
solution :

```
let motif_europeen = numeros_vers_motifs numeros_europeens;;
```

4. Démontrer que la fonction suivante :

```
let est_ponctuel (m:motif) : bool =
    (m>0) && ((m land (m-1)) = 0) ;;
```

renvoie `true` si et seulement si le motif m est ponctuel.

solution :

- Supposons que m est ponctuel. Soit alors $i \in [\![0, 62]\!]$ tel que $m = 2^i$. En base deux, on a $m = \overline{0...0 \underbrace{1}_{\text{position i}} 0...}$ et $m - 1 = \overline{0...0 \underbrace{0}_{\text{position i}} 1...1}$. Le « et » bit à bit de ces deux nombres vaut bien 0. En outre, $m > 0$.
- Traitons la réciproque : on suppose $m > 0$ et `(m land (m-1))= 0`. Procédons par l'absurde et supposons que m contient au moins deux encoches. Soient i et j les numéros des deux plus petites d'entre elles, en prenant $i < j$. Donc

$$\begin{cases} m = \overline{... \underbrace{1}_{\text{position j}} 0...0 \underbrace{1}_{\text{position i}} 0...0} \\ m - 1 = \overline{... \underbrace{1}_{\text{position j}} 0...0 \underbrace{0}_{\text{position i}} 1...1} \end{cases}$$

  Et on constate que `(m land (m-1))= 0` est non nul car le bit j (au moins) est non nul. Ce qui est absurde.

5. Écrire à l'aide des opérateurs logiques sur les entiers une fonction `inclus (m:motif)(p:ponctuel): bool` qui renvoie `true` si et seulement si le motif ponctuel p est inclus dans le motif m. Donner une preuve mathématique du résultat.
solution :

```
let inclus (m:motif) (p:ponctuel) : bool =
  m land p <> 0;;
```

Soit m un motif et p un motif ponctuel. Notons i le numéro de l'unique encoche de p, donc $p = 2^i = \overline{0...010...0}$ (le seul 1 étant en position i).
Alors `m land p` vaut $\overline{0...0[m]_i0...0}$. Il est nul si et seulement si $[m]_i$ est nul, si et seulement si le motif m contient l'encoche i.

6. Écrire, à l'aide des opérateurs logiques sur les entiers, une fonction `voisin_g (p:ponctuel): ponctuel` qui calcule la translation par une encoche vers la gauche du motif ponctuel p. Si le motif ponctuel p sort du tablier, la valeur de retour est le motif vide 0.
solution :

```
let valide (p:ponctuel) =
  (* Entrée : motif ponctuel p.
     Sortie : booléen « p est dans le tablier européen ». *)
  inclus motif_europeen p;;


let voisin_g (p:ponctuel) : ponctuel =
  let res = p lsr 1 in
  if  valide res then res
  else 0
;;
```

1.3 Coups simples et composés

1. Écrire une fonction `coup_simple ((m,p):motif*ponctuel): (motif*ponctuel)list` qui prend un motif m et un motif ponctuel p contenu dans m et qui renvoie la liste des couples (m', p') où m' est un motif obtenu à partir de m à la suite du déplacement par un coup simple du fichet initialement placé dans p, et p' est le motif ponctuel représentant le même fichet après son déplacement.
solution :

```
let coup_simple ((m,p) : motif*ponctuel) =

  let coup voisin  =
    (* Entrée : voisin, une des quatre fonctions voisin_g, voisin_d, voisin_h, voisin_b.
       Sortie : liste [ (m', p')] si le coup est possible, [] sinon *)
    let case_suivante = voisin p in
    let case_dapres = voisin case_suivante in
    if (inclus m case_suivante) && not (inclus m case_dapres) && case_dapres<>0  then [
        ↪ m-p-case_suivante+case_dapres, case_dapres]
    else []
  in
  List.flatten (List.map coup [voisin_g; voisin_d; voisin_h; voisin_b] )
;;
```

2. Écrire une fonction `coup_compose ((m,p): motif*ponctuel): (motif*ponctuel)list` qui prend un motif m et un motif ponctuel p et renvoie la liste des couples (m', p') où m' est un motif obtenu à partir de m à la suite d'un déplacement par un coup composé du fichet initialement placé en p, p' est le motif ponctuel représentant le même fichet à la suite de son déplacement.
solution :

```
let coup_compose (c:motif*ponctuel) =

  let rec aux l =
    (* Entrée : l, liste de couples (m',p') accessibles depuis (m,p).
       Sortie : liste des couples accessibles (en n'importe quel nombre de coups avec le
           ↪ même fichet) depuis un élément de l.
       Contruirement à la consigne de l'énoncée je ne mets pas le coup consistant à ne
           ↪ rien faire.
     *)
    if l=[] then []
    else
      let nv_couples = List.flatten (List.map coup_simple l) in
      nv_couples @ aux nv_couples

  in
  aux [c]
;;
```

3. Écrire une fonction `mouvements (m:motif): motif list` dont la valeur de retour est la liste de tous les motifs que l'on peut obtenir en jouant un coup composé depuis le motif m, n'importe quel fichet pouvant être déplacé.
solution :

```
let mouvements (m:motif) : motif list =

  let rec aux (p:ponctuel) =
    (* Entrée : p, prochain motif ponctuel à essayer
       Sourtie : liste des motifs accessibles en jouant un fichet en p ou plus. *)
    if p = 1 lsl 53 then []
    else (if inclus m p then coup_compose (m,p) else [] )
         @ aux (2*p)
  in
  List.map fst (aux 4)
;;
```

1.4 Partie de longueur minimale

1. Écrire une fonction `ajoute_et_mem` (d:dico) (s:int) (m:motif) : bool| qui ajoute l'association entre le motif m et l'entier s dans le dictionnaire d si le motif m est initialement absent. En outre, cette fonction renverra un booléen qui indique si le dictionnaire a été modifié.
solution : Cette fonction aurait du s'appeler `ajoute_et_pas_mem`.

```
let ajoute_et_pas_mem (d: dico) (s:int) (m:motif) =
  let dedans = Hashtbl.mem d m in
  if not dedans then Hashtbl.add d m s ;
  dedans
;;
```

2. Écrire une fonction `strate (d:dico)(s:int)(l:motif list): motif list` qui, pour tout motif que l'on peut obtenir après un coup composé à partir d'un motif de la liste l, ajoute l'association entre le motif m et l'entier s au dictionnaire d si le motif m n'est pas présent dans le dictionnaire d. La valeur de retour de la fonction est la liste des motifs que la fonction ajoute.
solution :

```
let strate (d:dico) s (l:motif list) =
  let nv_motifs = List.flatten (List.map mouvements l) in

  let rec aux  = function
      (* Fonction qui va parcourir la liste des motifs accessibles.
         Chaque motif qui n'était pas encore dans d va y être rajouté, associé à la
             ↪ valeur s, et figurer dans la list erenvoyée.*)

    | [] -> []
    | m::suite when Hashtbl.mem d m -> aux suite
    | m::suite -> Hashtbl.add d m s; m :: aux suite
  in
  aux nv_motifs
;;
```

3. Dans cette question, nous supposons qu'il est possible de passer d'un motif m_i à un motif m_f par une suite de coups. Écrire une fonction `partie_minimale (mi:motif)(mf:motif): int` qui renvoie le nombre minimal de coups composés à jouer pour passer du motif initial m_i au motif final m_f. On pourra utiliser un dictionnaire qui associe à des motifs le nombre minimal de coups pour les atteindre depuis m_i.
solution :

```
let partie_minimale mi mf =
  let d = Hashtbl.create 42 in
  Hashtbl.add d mi 0 ;
  (* d associe à chaque motif obtenu le nombre de coups minimal pour l'atteindre *)

  let rec aux sphere dist =
    (* sphere : ensemble des motifs à distance dist de mi. *)
    if Hashtbl.mem d mf then Hashtbl.find d mf
    else if sphere = [] then failwith "motif final non accessible"
    else
      let sphere_suivante = strate d (dist+1) sphere in
      aux sphere_suivante (dist+1)
  in
  aux [mi] 0
;;
```

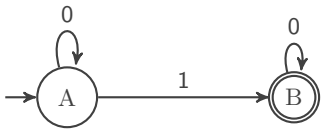
4. Peut-on considérer que la réponse donnée à la question 3 observe le parcours nommé à la question 1 ? Introduire un graphe puis argumenter.
solution : Soit V l'ensemble des motifs possibles sur le tablier choisi. Soit A l'ensemble des couples (m, m') où m' est un motif accessible en un coup composé depuis m. Soit $G = (S, A)$. Alors le but de la question précédente était de chercher la distance dans G d'un sommet m_i à m_f.
La programmation ci-dessus n'était pas la version traditionnelle utilisant une file d'attente : j'avais une liste qui contenait à chaque étape une sphère centrée en m_i. Mais les sommets sont bien visités du plus proche de m_i au plus lointain.
La file d'attente utilisée en cours contenait à chaque instant la fin d'une sphère et le début de la sphère suivante.

2 Reconnaissance des motifs résolubles sur le tablier unidimensionnel

2.1 Mots de motifs ponctuels

Nous notons $\mathcal{P} \subset \Sigma^*$ le langage des mots de longueur quelconque de motifs ponctuels.

1. Donner sans justification une expression rationnelle qui décrit le langage $\mathcal{P}$.
   solution : $\mathcal{P} = 0^*10^*$
2. Dessiner sans justification un automate fini que reconnaît $\mathcal{P}$.
   solution :



3. Le langage $\mathcal{P}$ est-il local ?
   solution : L'expression régulière que nous avons donnée ci-dessus n'est pas linéaire, mais cela ne prouve pas que $\mathcal{P}$ n'est pas local.
   Supposons $\mathcal{P}$ local. Soit alors (P, S, F, α) l'ensemble des premières lettres, des dernières lettre, des facteurs de longueur 2, et le booléen $\varepsilon \in \mathcal{P}$ qui décrivent $\mathcal{P}$ en tant que langage local.
   Comme $010 \in \mathcal{P}$, on a $0 \in P$, $0 \in S$, $\{01, 10\} \subset F$. Mais alors $01010 \in \mathcal{P}$, ce qui est absurde.
   Donc $\mathcal{P}$ n'est pas local.

2.2 Bascules de l'état d'une encoche

Soit $w = (w_0, ..., w_m) \in (\Sigma^n)^{m+1}$ une partie en m coups simples joués sur le tablier de dimension $n \times 1$ et $k \in [\![1, n]\!]$. Dans cette sous-partie nous souhaitons démontrer qu'il existe une constante α indépendante des entiers n, m ou k telle que la suite des $(m+1)$ symboles $[w_0]_k$,...,$[w_n]_k$ ne change jamais de valeur à plus de α reprises.

Nous notons $\varphi = \frac{\sqrt{5}-1}{2}$ l'une des racines du polynome $P = X^2 + X - 1$. On pourra utiliser sans preuve l'inégalité $\sum_{j=1}^{\infty} \varphi^j \leqslant 2$. Pour tout mot $w \in \Sigma^n$ de longueur n, nous introduisons le variant $V_k(w)$ par la somme

$$V_k(w) = \sum_{j=1}^{n} \varphi^{|j-k|} [w]_j \in \mathbb{R}.$$

1. Montrer qu'il existe un réel S, indépendant des entiers n et k, tel que, pour tout $w \in \Sigma^n$ on ait $V_k(w) \leq S$.
   solution : Déjà, les seules lettres de Σ sont 0 et 1, d'où pour tout $w \in \Sigma^n$,

$$V_k(w) \leq \sum_{j=1}^{n} \varphi^{|j-k|}$$

   Je vais ensuite découper la somme :

$$\begin{aligned} V_k(w) \leq & 1 + \textstyle\sum_{j=1}^{k-1} \varphi^{k-j} + \sum_{j=k+1}^{n} \varphi^{j-k} && \big) \textit{ Changements d'indices} \\ \leq & 1 + \textstyle\sum_{j=1}^{k-1} \varphi^{j} + \sum_{j=1}^{n-k} \varphi^{j} && \big) \textit{ Série à termes positifs} \\ \leq & 5 \end{aligned}$$

   La constante $S = 5$ convient, elle est bien indépendante de n, k, et w.
2. Montrer que la suite des $(m+1)$ variants $V_k(w_0)$,..., $V_k(w_m)$ est une suite de réels décroissante.
   solution :

   ⧙ Il s'agit de vérifier qu'à chaque coup joué, le variant diminue.

   Soit $i \in [\![0, m[\![$.

   - **Cas d'un coup vers la droite :** Supposons que le $i$ème coup joué est vers la droit. Soit a le numéro du fichet joué. Alors les lettres qui changent entre w_i et w_{i+1} sont celles d'indices a, $a+1$ et $a+2$ qui passent de 110 à 001.
     Ainsi :

$$\begin{aligned} V_k(w_{i+1}) - V_k(w_i) &= \textstyle\sum_{j=a}^{a+2} \varphi^{|j-k|} \left([w_{i+1}]_j - [w_i]_j \right) \\ &= -\varphi^{|a-k|} - \varphi^{|a+1-k|} + \varphi^{|a+2-k|} \end{aligned}$$

     ◇ *Si $a \geqslant k$:* $V_k(w_{i+1}) - V_k(w_i) = \varphi^{a-k}\left(-1 - \varphi + \varphi^2\right)$. Ce n'est pas exactement $P(\varphi)$, mais c'est plus petit ! On a $-1 - \varphi + \varphi^2 \leq -1 + \varphi + \varphi^2 = P(\varphi) = 0$. Ainsi, $V_k(w_{i+1}) - V_k(w_i) \leq 0$.

◇ *Si* $a \leq k-2$ *:* $V_k(w_{i+1}) - V_k(w_i) = -\varphi^{k-a} - \varphi^{k-a-1} + \varphi^{k-a-2} = \varphi^{k-a-2}\left(-\varphi^2 - \varphi + 1\right) = 0$.

◇ *Dernier cas : si* $a = k-1$. Alors $V_k(w_{i+1}) - V_k(w_i) = -\varphi - 1 + \varphi = -1$.

- **Cas d'un coup vers la gauche :** En notant encore a le numéro du fichet joué, ce sont cette fois les lettre d'indices $a-2$, $a-1$ et a qui passent de 011 à 100 d'où

$$V_k(w_{i+1}) - V_k(w_i) = \varphi^{|a-2-k|} - \varphi^{|a-1-k|} - \varphi^{|a-k|}.$$

On traite alors les trois cas $a \geqslant k+2$, $a \leq k$ et $a = k+1$ comme ci-dessus. En particulier dans le cas où $a = k+1$, on trouve encore $V_k(w_{i+1}) - V_k(w_i) = -1$.

3. Montrer que si pour un indice $i \in [\![0, m[\![$, $[w_i]_k = 1$ et $[w_{i+1}]_k = 0$, alors $V_k(w_{i+1}) \leq V_k(w_i) - 1$.
solution : Notons comme précédemment a l'emplacement du fichet joué au coup i. Plaçons-nous dans le cas suggéré par l'énoncé.

- **Cas d'un coup vers la droite :** On a alors $k = a$ ou $k = a+1$.
  ◇ Si $k = a$, on avait vu que $V_k(w_{i+1}) - V_k(w_i) = \varphi^{a-k}\left(-1 - \varphi + \varphi^2\right) = -1 - \varphi + \varphi^2 = P(\varphi) - 2\varphi = -2\varphi = 1 - \sqrt{5}$. Or

$$\begin{aligned} 1 - \sqrt{5} \leq -1 &\Leftrightarrow \sqrt{5} \geqslant 2 \\ &\Leftrightarrow 5 \geqslant 4 \\ &\Leftrightarrow \top \end{aligned}$$

Donc on a bien $V_k(w_{i+1}) - V_k(w_i) \leq -1$.

◇ Si $k = a+1$, on avait vu que $V_k(w_{i+1}) - V_k(w_i) = -1$.

- On traite le cas d'un coup vers la gauche de manière analogue.

4. En déduire qu'il existe une constante α, indépendante des entiers n, m et k telle que la suite $[w_0]_k$,...,$[w_m]_k$ ne change jamais de valeur à plus de α reprises.
solution : On sait que $V_k[w_0] \leq 5$. Ensuite la suite $\left(V_k[w_i]\right)_{i \in [\![0,m]\!]}$ est décroissante. Et chaque fois que la lettre k passe de 1 0, le variant diminue d'au moins 1. Cela ne peut se produire qu'au plus 5 fois. Le seul autre changement de valeur possible est le passage de 0 à 1. Celui-ci peut se produire au plus une fois de plus que le changement inverse (si la partie débute avec un 0 en position k et finit avec un 1). Ce qui nous fait un totale de 11 changements de valeur a maximum.

2.3 Trace d'une partie

1. Soit T la trace d'une partie. Décrire une construction qui produit une partie de trace T à partir de la trace T. Il est suggéré de raisonner par récurrence.
solution : J'appellerai « motif élémentaire » un triplet (c, d, e) de trois cases portant les marques I, II, et III, sur trois colonnes successives, et telles que la ligne de d est indiquée dans c, et la ligne de e est indiquée dans d.

Je définis une fonction auxiliaire récursive, donc je note T' l'argument. T' sera un sous tableau de T, sans case vide recouverte par une case non vide, obtenue en supprimant des motifs élémentaires de T. Ma fonction auxiliaire est définie ainsi :

- Si T' n'a qu'une seule ligne, prendre une partie sans aucun coup joué.
- Sinon, soit (c, d, e) un motif élémentaire tel que chacune des trois cases c, d, et e est au sommet de sa colonne. Il en existe au moins un : celui qui provient du dernier coup joué dans la partie de laquelle on a déduit T, et qui figure encore dans T.

  On supprime alors (c, d, e), on obtient un nouveau tableau T'' qui vérifie les conditions imposées aux arguments de ma fonction auxiliaire. On obtient une partie qui donne lieu à T'' par un appel récursif. On fait suivre cette partie d'un coup joué dans la case de la colonne de c, et dans la direction donnée par les flèches contenues dans c, d, et e.

N.B. Pour terminer ce sujet de façon satisfaisante, il faudrait une définition de trace « valide » et prouver la justesse de cet algorithme pour ces traces. Ce qui prouverait que toute trace « valide » provient d'une partie. Cette définition sera en fait vue question 4.

2. Montrer que l'on peut majorer le nombre de lignes non vides dans la trace d'une partie par une constante indépendante de la dimension n du tablier et de la longueur m de la partie. En déduire qu'une colonne ne peut prendre qu'un nombre fini de valeurs distinctes.
solution : Par construction, il ne reste dans une colonne k que des cases correspondant à un changement dans la lettre k. Or nous avons vu à la partie précédente que le nombre de changement de lettre dans une position donnée est majoré (par 11). D'où le résultat.

Ensuite, chaque case n'a qu'un nombre fini de valeurs possible : un élément de Σ (il y en a deux), assorti d'une flèche vers la gauche ou la droite (deux choix), d'une marque en exposant (trois choix), et éventuellement d'un numéro de ligne, or le nombre de numéros de lignes possible est borné.

Il y a donc un nombre fini de colonnes différentes.

Nous appelons Δ l'ensemble, fini, des valeurs des colonnes que peuvent prendre les colonnes des traces de partie. Nous appelons $\mathcal{T} \subset \Delta^*$ le langage des mots sur Δ qui représentent la trace d'une partie.

3. Donner une caractérisation de l'ensemble des facteurs de longueur 2 de mots du langage $\mathcal{T}$.
solution : Je noterai pour toute colonne C, $|C|$ sa longueur.
L'ensemble cherché est l'ensemble des couples $(C, D) \in \Delta^2$ tels que pour tout $i \in [\![0, |C|[\![$, si C_i porte un numéro de ligne j, alors D_j contient la suite du motif élémentaire présent en C_i.

   En toute rigueur il faudrait prouver que tout couple (C, D) vérifiant cette propriété figure effectivement dans un mot de $\mathcal{T}$.

4. Montrer qu'il existe un automate local qui reconnaît exactement le langage $\mathcal{T} \in \Delta^*$.
solution : Notons F l'ensemble décrit question précédente.
Soit $P \subset \Delta$ l'ensemble des colonnes dont toutes les cases, sauf celle du bas (qui n'a aucune marque), ont la marque I, et D l'ensemble des colonnes dont toutes les cases, sauf celle du bas, ont la marque III.
Montrons alors que $\mathcal{T}$ est l'ensemble des mots non vides sur Δ dont la première lettre est dans P, la dernière dans D, et les facteurs de longueur 2 dans F. Ceci prouvera que $\mathcal{T}$ est local et donc reconnaissable par un automate local.
   - Soit $T \in \Delta^*$ non vide, dont la première lettre est dans P, la dernière dans D, et les facteurs de longueur 2 dans F. Par l'algorithme de la question 1 il existe une partie dont T est la trace.
   - Réciproquement, soit T la trace d'une partie. Elle vérifie alors bien les trois conditions : sa première colonne n'a que des marques I, sa dernière n'a que des III, et les facteurs de longueur deux sont dans F.

5. Montrer qu'il existe un automate fini qui reconnaît exactement le langage $\mathcal{R} \subset \Sigma^*$ des mots de longueur quelconque de motifs résolubles.
solution : Prenons pour ensemble d'états $Q = \Delta \times \{0, 1, p\} \cup \{q_0\}$ (p est considéré ici comme un simple symbole, q_0 aussi.). L'idée est que le deuxième élément d'un couple de Q indique le nombre de 1 au sommet d'une colonne rencontré jusqu'ici. Et « p » signifie « strictement plus que 1 ».
Pour tout $\big((C, n), (D, m)\big) \in Q^2$ et $x \in \Sigma$, nous mettons une transition $(C, n) \underset{x}{\curvearrowright} (D, m)$ si et seulement si (C, D) est dans l'ensemble déterminé question 3, et la case du bas de D contient x, et si le sommet de D est 0 ou si $n = p$, alors $n = m$, dans le contraire, $m = n + 1$ (étant convenu que $1 + 1 = p$).
Comme état initial prenons q_0. Nous mettons pour tout $(C, n) \in Q$ et $x \in \Sigma$ une transition de q_0 vers (C, n) ssi $C \in P$ (défini dans la réponse à la question 4), la case du bas de C contient x, et $n = 1$ si la case du haut de C contient 1, 0 sinon.
Comme ensemble d'états finals, prenons tous les états dont la seconde composante est 1 et dont la première composante est dans D (défini dans la réponse à la question 4).

   - Soit m un motif reconnu par cet automate et γ le chemin acceptant correspondant. La suite des colonnes dans γ fournit un $T \in \Delta^*$ qui vérifie les conditions décrites question **??** . Par conséquent il existe une partie de trace T. Comme l'état final est acceptant, cela signifie que dans T une seule colonne a un 1 à son sommet, autrement dit que la partie est gagnante.
   - Réciproquement, si m est résoluble, soit T la trace associée à une partie gagnante partant de m. De cette trace on déduit un chemin acceptant dans notre automate, donc m est reconnu par celui-ci.

Dans la question suivante, le terme *complexité* désigne un ordre de grandeur asymptotique de la complexité en temps et dans le pire des cas.

6. On s'intéresse au problème de déterminer si un motif sur le tablier unidimensionnel est résoluble ou non. Montrer que l'on peut déduire de l'automate construit à la question 5 un algorithme de complexité optimale pour résoudre ce problème.
solution : Soit A l'automate de la question 5, déterminisé. Donc la lecture de cette automate fourni un algorithme en $O(n)$ pour déterminer si un motif est résoluble. Toute la question est de prouver que ce $O(n)$ est optimal.
Il suffit par exemple de trouver pour tout $n \in \mathbb{N}$ et tout $k \in [\![0, n[\![$ un motif de longueur n résoluble tel que le changement de son élément d'indice k le transforme en motif non résoluble. Je propose le motif $m = 0...01$. Il est résoluble, et pour tout $k \in [\![0, n-3]\!]$ si on change le 0 en position k en un 1, on obtient un motif non résoluble. Ceci prouve qu'il est indispensable de lire ces $n-2$ lettres, donc un algorithme capable de déterminer si m est résoluble aura une complexité au moins de l'ordre de n.

Les résultats démontrés dans cette partie ont été établis par B. Ravikumar pour un tablier rectangulaire de dimension $n \times n_0$ où n_0 est un entier fixé et n un entier pouvant varier.