Lièvre, tortue et automates

Travail 03

Ceci est une proposition de correction. Il existe d'autres chemins pour arriver au même résultat.

Partie 1 : algorithme du lièvre et de la tortue

Question 1 : base mathématique

Soit E un ensemble fini et $f : E \to E$. Pour $x \in E$, on définit la suite $(u_n)_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ telle que $u_0 = x$ et $u_{n+1} = f(u_n)$.

1. Montrer que la suite $(u_n)_{n\in\mathbb{N}}$ est ultimement périodique, c.-à-d. qu'il existe $K \in N$ et $P \in \mathbb{N}^*$ tels que pour tout $n \geqslant K, u_n = u_{n+P}$.
2. On note $Per(u) = \{P \in \mathbb{N}^* \mid \exists K \in N, \forall n \geqslant K, u_n = u_{n+P}\}$. Montrer qu'il existe un unique $\tilde{P}$ tel que $Per(u) = \tilde{P}\mathbb{N}^*$.

   On appellera $\tilde{P}$ la période de u.
3. On note $Prep(u, P) = \{K \in \mathbb{N} \mid \forall n \geqslant K, u_n = u_{n+P}\}$.
   a) Justifier que $Prep(u, \tilde{P})$ admet un minimum $\tilde{K}$.
   b) Montrer que $\tilde{K} = \min\{\min(Prep(u, P)) \mid P \in Per(u)\}$.

   On appelle $\tilde{K}$ la prépériode de u. Quand l'énoncé évoquera **la** période et **la** prépériode d'une suite, ce sera en référence à $\tilde{P}$ et $\tilde{K}$.

Bonus. Justifier l'existence de la période et de la prépériode d'une suite par un simple dessin.

Solution.

1. On remarque que la suite $(u_n)_{n\in\mathbb{N}}$ est à valeurs dans un ensemble fini. Par le lemme des tiroirs, il existe $K \in \mathbb{N}$ et $P \in \mathbb{N}^*$ tel que $u_K = u_{K+P}$.
   Montrons par récurrence que pour tout $n \geqslant K$, $u_n = u_{n+P}$:
   Initialisation : pour $n = K$, on sait que $u_K = u_{K+P}$.
   Hérédité : supposons que $u_n = u_{n+P}$. Alors :
   $$u_{n+1} = f(u_n) = f(u_{n+P}) = u_{n+P+1}$$
   Donc la propriété est vérifiée au rang $n + 1$.
   Donc la suite est ultimement périodique.
2. Comme $Per(u)$ est non vide, est inclus dans $\mathbb{N}^*$ (donc minoré) et est fermé (pour la topologie sur $\mathbb{R}$), $Per(u)$ admet une borne inférieure qui est son minimum. Notons la $\tilde{P}$.
   Soit $Q \in Per(u)$. En notant $K_{\tilde{P}}$ et K_Q les K correspondant, on pose $K = \max(K_{\tilde{P}}, K_Q)$.
   D'après l'identité de Bézout, il existe $a, b \in \mathbb{Z}^2$ tels que $\text{PGCD}(\tilde{P}, Q) = a\tilde{P} + bQ$. L'un de a ou de b doit être positif ; on étudie le cas $a \geqslant 0$, l'autre étant analogue. Alors, pour tout $n \geqslant K$:
   $$u_n = u_{n+a\tilde{P}} = u_{n+a\tilde{P}+bQ} = u_{n+\text{PGCD}(\tilde{P},Q)}$$
   Donc $\text{PGCD}(\tilde{P}, Q) \in Per(u)$. Par minimalité de $\tilde{P}$, $\text{PGCD}(\tilde{P}, Q) \geqslant \tilde{P}$; donc $\text{PGCD}(\tilde{P}, Q) = \tilde{P}$. Donc $\tilde{P} \mid Q$. Donc $Per(u) \subseteq \tilde{P}\mathbb{N}^*$.
   Pour l'inclusion réciproque, si $k \in \mathbb{N}^*$, pour tout $n \geqslant K_{\tilde{P}}, u_{n+\tilde{P}} = u_{n+2\tilde{P}} = \cdots = u_{n+k\tilde{P}}$ par récurrence immédiate ; donc $\tilde{P}\mathbb{N}^* \subseteq Per(u)$.

3. a) L'ensemble $Prep(u, P)$ est un fermé (pour la topologie sur $\mathbb{R}$) non vide minoré ; il admet un minimum.

   b) *Repris d'Eric Desurmont.* Soit $K' = \min\{\min(Prep(u, P)) \mid P \in Per(u)\}$. Alors, par minimalité, $K' \leqslant \tilde{K}$.

   Puis soit $P \in Per(u)$ et $K \in Prep(u, P)$. Si $n \geqslant K$:

$$\begin{aligned} u_n &= u_{n+P\times k} && \text{pour tout } k \in \mathbb{N} \\ &= u_{n+P\times k_0} && \text{tel que } n + P \times k_0 \geqslant \tilde{K} \\ &= u_{n+P\times k_0+\tilde{P}} = u_{n+\tilde{P}} && \text{: on peut retirer } P \text{ car } n + \tilde{P} \geqslant n \geqslant K \end{aligned}$$

   Donc $K \in Prep(u, \tilde{P})$, donc $\tilde{K} \leqslant K$, donc $\tilde{K} \leqslant K'$.

L'objectif est maintenant de calculer la période et la prépériode d'une suite de manière efficace.

Question 2 : algorithme naïf

1. Écrire en OCaml une fonction `per_prep : 'a -> ('a -> 'a) -> (int * int)` qui prend en entrée u_0 et f et renvoie le couple $(\tilde{P}, \tilde{K})$ correspondant à la suite u générée.

   On suppose qu'un appel à f (peu importe l'entrée) nécessite F opérations : la fonction devra s'exécuter en $O((\tilde{N} + \tilde{K})F + (\tilde{N} + \tilde{K})^2)$. On justifiera cette complexité. *Si la suite n'est pas ultimement périodique, on autorise l'algorithme à ne pas terminer.*

2. Justifier que si le type `'a` est hashable, la complexité temporelle de votre algorithme peut descendre à $O((\tilde{N} + \tilde{K})F)$ en utilisant des dictionnaires.

Solution.

1.
```
let rec est_dedans li elem =
    match li with
    | [] -> false
    | t::q when t = elem -> true
    | t::q -> est_dedans q elem;;

let rec indice li elem =
    match elem with
    | [] -> failwith "oups !"
    | t::q when t = elem -> 0
    | t::q -> 1 + (indice li q);;

let per_prep u0 f =
    let rec boucler elem suite =
        if est_dedans elem suite
        then begin
            let k = indice suite elem in
            let p = (List.length suite) - k in
            k,p
            end
        else
            boucler (f elem) elem::suite
    in
    boucler u0 [];;
```

La fonction `est_dedans` opère en temps linéaire sur la liste à traiter; la fonction `indice` aussi. À chaque nouvel appel à la fonction `boucler`, on fait un test sur `suite`, qui est de longueur au plus $\tilde{K}+\tilde{P}$. De plus, à chaque appel, on appelle la fonciton F. Donc chaque appel nécessite $O(F+\tilde{K}+\tilde{P})$ opérations élémentaires. On procède à au plus $O(\tilde{K}+\tilde{P})$ appels; la fonction opère en temps $O((\tilde{K}+\tilde{P})\times(F+\tilde{K}+\tilde{P}))$.

2. Si le type `'a` est hashable, plutôt que de retenir la suite déjà calculée dans une liste, on peut stocker dans un dictionnaire qui, à un élément, associe l'indice où on trouve ledit élément. Plutôt que d'appeler `est_dedans` et `indice` de complexité linéaire, on fait alors des appels en $O(1)$; la complexité s'écrase en $O((\tilde{P}+\tilde{K})F)$.

Question 3 : algorithme du lièvre et de la tortue

On présente ici un autre algorithme en temps $O((\tilde{N}+\tilde{K})F)$ qui ne fait pas appel aux dictionnaires.

1. Montrer qu'il existe $m\in\mathbb{N}^*$ tel que $u_m=u_{2m}$.
2. Soit m minimal vérifiant la question précédente. Montrer que $\tilde{P}\mid m$ et que $\tilde{K}\leqslant m$.
3. Montrer que $m\leqslant\tilde{K}+\tilde{P}$.
4. Soit m' minimal tel que $m'>m$ et $u_{m'}=u_{2m'}$.
   a) Justifier l'existence de m'.
   b) Montrer que $m'\leqslant m+\tilde{P}$.
   c) Exprimer $\tilde{P}$ en fonction de m et m'.
5. Écrire une fonction `lievre_tortue : 'a -> ('a -> 'a) -> (int*int)` qui prend en entrée u_0 et f et renvoie le couple $(\tilde{P},\tilde{K})$ en temps $O((\tilde{N}+\tilde{K})F)$.

Solution.

1. Il suffit de prendre $m=\tilde{K}\times\tilde{P}$: $u_{\tilde{K}\times\tilde{P}}=u_{\tilde{K}\times\tilde{P}+\tilde{P}}=\cdots=u_{\tilde{K}\times\tilde{P}+\tilde{K}\times\tilde{P}}=u_{2\times\tilde{K}\times\tilde{P}}$.
2. On remarque que m définit une période de u : pour tout $n\geqslant 0, u_{m+n}=u_{m+n+m}$ par récurrence immédiate. Donc $m\in Per(u)=\tilde{P}\mathbb{N}$; donc $P\mid m$. On remarque par ailleurs que $m\in Prep(u,m)$; donc d'après la question 1.3.b), $\tilde{K}\leqslant m$.
3. Dans l'intervalle $[\![\tilde{K},\tilde{K}+\tilde{P}]\!]$, il y a un multiple strictement positif de $\tilde{P}$; notons-le $j\tilde{P}$. Alors $u_{2j\tilde{P}}=u_{j\tilde{P}}$; par minimalité de m, on en déduit que $m\leqslant\tilde{K}+\tilde{P}$.
4. a) On peut étendre le résultat de la question 3.1. : on remarque qu'en fait, pour tout $j\tilde{K}\tilde{P}$, on a aussi $u_{j\tilde{K}\tilde{P}}=u_{2j\tilde{K}\tilde{P}}$; donc il y a une infinité de m' tels que $u_{m'}=u_{2m'}$, et donc un minimal plus grand que m.
   b) C'est le même raisonnement qu'à la question 3.3 : dans l'intervalle $[\![m+1,m+\tilde{P}]\!]$, on est sûr de trouver un multiple de $\tilde{P}$ qui conviendra.
   c) De même que pour m, m' est aussi une période, donc $\tilde{P}\mid m'$. Donc $m<m'\leqslant m+\tilde{P}$, $P\mid m$ et $P\mid m'$: on a donc $m'=m+\tilde{P}$. Donc $\tilde{P}=m'-m$.

5.
```
let lievre_tortue u0 f =
    let a = ref (f u_0) and b = ref f (f u0) in
    let m = ref 1 in
    while !a <> !b do
        a := f !a ;
        b := f (f !b) ;
        incr m
    done;
    let m' = ref (!m+1) in
    a := f !a ; b := f (f !b);
    while !a <> !b do
        a := f!a ;
        b := f (f !b) ;
        incr m'
    done;
    let p = !m' - !m in
    a := u_0 ; b := u0 ;
    for k=0 to (p-1) do
        b := f !b
    done;
    let k = ref 0 in
    while !a <> !b do
        a := f !a ; b = f !b
    done;
    p,!k;;
```

Partie 2 : désubstitution d'automate

Question 4

Soit $\mathcal{A}$ un alphabet et soit $\sigma : \mathcal{A}^* \to \mathcal{A}^*$ un morphisme du monoïde libre, c.-à-d. $\forall (u,v) \in (\mathcal{A}^*)^2, \sigma(u \cdot v) = \sigma(u) \cdot \sigma(v)$.

1. Justifier que σ est entièrement défini par les images des lettres : si τ est un morphisme du monoïde libre tel que $\sigma(a) = \tau(a)$ pour tout $a \in \mathcal{A}$, alors $\sigma = \tau$.
2. Pour un langage L, on pose $\sigma^{-1}(L) = \{m \in \mathcal{A}^* \mid \sigma(m) \in L\}$. Montrer que si L est régulier, alors $\sigma^{-1}(L)$ est aussi régulier.
3. Pour un automate déterministe complet $\mathfrak{A} = (\mathcal{A}, Q, i, F, \delta)$, donner explicitement un automate déterministe complet $\sigma^{-1}(\mathfrak{A})$ tel que $\mathcal{L}(\sigma^{-1}(\mathfrak{A})) = \sigma^{-1}(\mathcal{L}(\mathfrak{A}))$.

Solution.

1. Soient σ et τ tels que pour tout $a \in \mathcal{A}, \sigma(a) = \tau(a)$. On montre que pour tout $m \in \mathcal{A}^*, \sigma(m) = \tau(m)$ par récurrence sur la longueur de m.
   Initialisation : pour $|m| = 0$, il suffit de remarquer que $\sigma(\varepsilon) = \varepsilon = \tau(\varepsilon)$.
   Hérédité : supposons la propriété vraie au rang n. Soit $m \in \mathcal{A}^{n+1}$; on pose $m = xa$ avec $x \in \mathcal{A}^n$ et $a \in \mathcal{A}$. Alors :

$$\begin{aligned} \sigma(m) &= \sigma(xa) = \sigma(x)\sigma(a) \\ &= \sigma(x)\tau(a) \qquad \text{par hypothèse sur les images des lettres} \end{aligned}$$

$$= \tau(x)\tau(a) \qquad \text{par hypothèse de récurrence}$$
$$= \tau(m)$$

Donc $\sigma = \tau$.

2 et 3. Soit L un langage régulier, alors par le théorème de Kleene, L est aussi reconnaissable, et même reconnaissable par un automate fini déterministe complet. Notons cet automate déterministe complet $\mathfrak{A} = (\mathcal{A}, Q, i, F, \delta)$.
Posons $\mathfrak{B} = (\mathcal{A}, Q, i, F, \Delta)$ tel que :

$$\forall q \in Q, \forall a \in \mathcal{A}, \Delta(q, a) = \delta^*(q, \sigma(a))$$

Le caractère déterministe complet de $\mathfrak{A}$ rend cette définition valide. Montrons que $\mathcal{L}(\mathfrak{B}) = \sigma^{-1}(L)$ par double inclusion.

Soit $m \in \mathcal{L}(\mathfrak{B})$; alors on peut décrire son calcul dans $\mathfrak{B}$ $i = q_0 \xrightarrow{m_0} q_1 \xrightarrow{m_1} q_2 \to \dots \xrightarrow{m_{|m|-1}} q_{|m|}$ avec $\Delta(q_j, m_j) = q_{j+1}$. On obtient alors que $\delta^*(q_j, \sigma(m_j)) = q_{j+1}$; donc en concaténant, on obtient par récurrence immédiate $\delta^*(i, \sigma(m)) = q_{|m|}$. Or, le premier calcul décrit étant acceptant dans $\mathfrak{B}$, $q_{|m|} \in F$; donc le dernier calcul est bien acceptant dans $\mathfrak{A}$, et $\sigma(m) \in \mathcal{L}(\mathfrak{A}) = L$. Donc $m \in \sigma^{-1}(L)$.
Soit $m \in \sigma^{-1}(L)$; alors $\sigma(m) \in L = \mathcal{L}(\mathfrak{A})$. On définit alors par récurrence les états $(q_j)_{0 \leqslant j \leqslant |m|}$ par :

— $q_0 = i$;
— $q_{j+1} = \delta^*(q_j, \sigma(m_j))$.

Ces états décrivent un calcul $i = q_0 \xrightarrow{\sigma(m_0)}{}^* q_1 \xrightarrow{\sigma(m_1)}{}^* q_2 \to^* \dots \xrightarrow{\sigma(m_{|m|-1})}{}^* q_{|m|}$ acceptant $\sigma(m)$ dans $\mathfrak{A}$. On en déduit immédiatement que $i = q_0 \xrightarrow{m_0} q_1 \xrightarrow{m_1} q_2 \to \dots \xrightarrow{m_{|m|-1}} q_{|m|}$ est un calcul acceptant m dans $\mathfrak{B}$. Donc $m \in \mathcal{L}(\mathfrak{B})$.

Question 5

On encode les automates déterministes complet sous le format suivant (similaire au sujet Mines-Ponts 2019) :

— on se place dans le cas d'un alphabet binaire $\mathcal{A} = \{a, b\}$;
— on suppose $Q = [\![0, n-1]\!]$ pour un certain entier n, et l'état initial i vaut toujours 0;
— un automate est encodé par un triplet `(n,delta,f)` où :
  — `n`, de type `int`, est le nombre d'états de l'automate;
  — `delta`, de type `(int*int) array`, est un tableau de longueur `n` qui stocke les couples $(\delta(q, a), \delta(q, b))$;
  — `f`, de type `bool array`, est un tableau de longueur `n` tel que `f.(q)` vaut `true` ssi q est un état final.
  — on pose alors : `type automate = int * (int*int) array * bool array`.

On implémente les mots comme étant des `char list` : le mot $abba$ est encodé comme `['a';'b';'b';'a']`. On rappelle que les `char` sont écrits avec des apostrophes, contrairement aux chaînes de caractères.

1. Écrire une fonction `delta_etoile : automate -> int -> char list -> int` qui prend en entrée un automate, un état q et un mot m et renvoie $\delta^*(q, m)$.
2. On encode un morphisme comme un couple de mots, donc de type `(char list) * (char list)` (par exemple `(['a';'b'],['a'])`). On pose alors : `type morphisme = (char list) * (char list)`. Le premier mot est l'image de a, le second l'image de b. Écrire une fonction `desubtitution : automate -> morphisme -> automate` d'entrées $\mathfrak{A}$ et σ et qui renvoie $\sigma^{-1}(\mathfrak{A})$.
3. La taille d'un morphisme est le maximum des tailles de ses images. Estimer la complexité de votre programme en fonction de n et de $|\sigma|$.

Solution.

1.
```
let rec delta_etoile aut q m =
    match m with
    | [] -> q
    | lettre::reste -> let (n,delta,f) = aut in
        let succ_a,succ_b = delta.(q) in
        let next = if lettre = 'a' then succ_a else succ_b in
        delta_etoile aut next reste;;
```

2.
```
let desubstitution aut sigma =
    let (n,delta,f) = aut in
    let mot1,mot2 = sigma in
    let delta2 = Array.make n (0,0) in
    for q=0 to (n-1) do
        delta2.(q) <- (delta_etoile q mot1, delta_etoile q mot2)
    done;
    (n,delta2,f);;
```

3. On rappelle que n est le nombre d'états de l'automate initial.
   D'abord, analysons la fonction `delta_etoile` : chaque appel est en $O(1)$, et elle procède à $|m|$ appels.
   Ensuite, analysons `desubstitution`. L'initialisation de `delta2` est en $O(n)$. Puis examinons la boucle pour : chaque passage de boucle fait 2 appels à `delta_etoile`, donc chaque passage est linéaire en la taille des mots du morphisme. Donc chaque passage est en $O(|\sigma|)$. Il y a n passages de boucle ; il y a donc $O(n \times |\sigma|)$ opérations élémentaires.

Question 6

Soit $\mathfrak{A} = (\mathcal{A}, Q, I, F, T)$ un automate fini de langage L, et σ un morphisme du monoïde libre.

1. Montrer que la suite $(\sigma^{-n}(L))_{n\in\mathbb{N}}$ est ultimement périodique.
2. Majorer la période et la prépériode de cette suite en fonction des données du problème.
3. Estimer la complexité du calcul des valeurs exactes de la période et de la prépériode en fonction de $\mathfrak{A}$ et de σ.

Solution.

1. On suppose $\mathfrak{A}$ déterministe complet. Plutôt que directement manipuler la suite des langages, intéressons-nous à la suite $(\sigma^{-n}(\mathfrak{A}))_{n\in\mathbb{N}}$. Par récurrence immédiate, on a $\mathcal{L}(\sigma^{-n}(\mathfrak{A})) = \sigma^{-n}(L)$.
   On remarque alors que l'opération de désubstitution construite ne change pas le nombre d'états : $\mathfrak{A}$ et $\sigma^{-1}(\mathfrak{A})$ ont le même nombre d'états. Donc la suite $(\sigma^{-n}(\mathfrak{A}))_{n\in\mathbb{N}}$ est à valeurs dans $E = \{\mathfrak{B} \mid \mathfrak{B}$ est un automate déterministe complet avec $|Q|$ états$\}$. Ce qui nous sauve est que E est un ensemble fini : $|E| \leqslant (|Q|^2)^{|Q|}$. Donc par la question 1.1), la suite $(\sigma^{-n}(\mathfrak{A}))_{n\in\mathbb{N}}$ est ultimement périodique.
   Donc $(\mathcal{L}(\sigma^{-n}(L)))_{n\in\mathbb{N}}$ est ultimement périodique.
2. Comme écrit précédemment, $|E| \leqslant (|Q|^2)^{|Q|} = |Q|^{2|Q|}$. En effet, il suffit de choisir pour chaque état ses deux états successeurs.
3. Il suffit d'appliquer la complexité du lièvre et de la tortue avec $\tilde{P}$ et $\tilde{K} \leqslant (|Q|^2)^{|Q|}$ et $F = O(|Q| \times |\sigma|)$. Finalement, on obtient une complexité en $O(|Q|^{2|Q|+1} \times |\sigma|)$. C'est beaucoup.