Mines 2016 - Option informatique
Un corrigé

1 Graphe du web

Fonctions utilitaires

1. Un consage transforme chaque couple en liste. Il suffit de concaténer ces listes.

```
let rec aplatir liste =
    match liste with
   |[] -> []
   |(x,l)::q -> (x::l)@(aplatir q);;
```

2. Un première fonction découpe une liste en deux morceaux de tailles comparables. Je choisis ici de répartir les éléments selon la parité de leur position dans la liste argument.

```
let rec decouper l =
    match l with
   |[] -> [],[]
   |[x] -> [x],[]
   |x::y::q -> let (l1,l2)=decouper q in
               x::l1,y::l2 ;;
```

La seconde fonction effectue la fusion de deux lists triées.

```
let rec fusion l1 l2 =
    match (l1,l2) with
   |[],_ -> l2
   |_,[] -> l1
   |(a,b)::q1,(c,d)::q2 -> if b>=d then (a,b)::(fusion q1 l2)
                           else (c,d)::(fusion l1 q2);;
```

La fonction principale combine le tout.

```
let rec tri_fusion l =
    match l with
   |[] -> []
   |[x] -> [x]
   |_  -> let (l1,l2)=decouper l in
          fusion (tri_fusion l1) (tri_fusion l2);;
```

3. On écrit une fonction auxiliaire
`uni_aux :  string liste` $\to$ `dictionnaire` $\to$ `int` $\to$ `string list * dictionnaire`.
Dans l'appel `uni_aux liste dico n`, on suppose que le dictionnaire *dico* possède exactement n éléments. On renvoie un couple formé des éléments distincts de *liste* et du dictionnaire obtenu en ajoutant ces éléments comme clefs à *dico*. L'argument n sert pour savoir quelle valeur associer à une clef et respecter la consigne de l'énoncé.

```
let unique liste =
    let rec uni_aux liste dico n=
       match liste with
```

```
    |[] -> [],dico
    |c::q -> if contient c dico then uni_aux q dico n
             else let (l,d)=uni_aux q (ajoute c n dico) (n+1)
                  in (c::l,d)
in uni_aux liste (dictionnaire_vide ()) 0;;
```

4. Pour chaque élément de *liste*, on teste son appartenance à *dico*. Le dictionnaire contenant au plus m clefs, ces tests ont un coût $O(n\log(m))$.
La construction du dictionnaire (appels à `ajoute`) a un coût $O(\sum_{i=1}^{m}\log(i)) = O(m\log(m))$.
La construction de la liste se fait par consages et a un coût $O(m)$.
Le coût total de l'appel est donc (puisque $n \geq m$) $O(n\log(m)+m)$.

Crawler simple

5. Dans une stratégie de parcours en largeur d'un graphe, on utilise usuellement :

   - un marquage des sommets pour éviter les trajets infinis (on ne redémarre pas une recherche à partir d'un sommet marqué, c'est à dire un sommet déjà visité) ;
   - une structure de file (FIFO) pour privilégier le traitement des sommets à courte distance.

   Ici, le sujet nous propose à l'évidence d'utiliser un dictionnaire comme structure de marquage : un adresse est marquée si elle est enregistrée dans le dictionnaire. Notons que la valeur (quantité entière) associée aux clefs n'a a priori pas d'importance. Utiliser le type `dictionnaire` est un peu délicat car il n'est pas mutable. Il n'est donc pas question de "faire évoluer" un dictionnaire (ou il faudrait une référence de dictionnaire). Ceci explique (voir plus bas) la présence d'un élément de type dictionnaire dans les arguments de la fonction écrite.
   Aucune structure n'est proposée pour modéliser les files. On pourrait utiliser la structure de file implémentée en Caml dans le module `queue`. Celui-ci n'est cependant a priori pas au programme. On pourrait aussi commencer par créer un type `file` dédié (par exemple, implémenter de manière persistante une file à l'aide d'un couple de listes). Ceci ne semble pas raisonnable en temps limité. Je choisis donc d'implémenter une file de manière persistante à l'aide d'une simple liste. La tête de la liste est la tête de la file et l'ajout d'élément doit se faire par la droite, c'est à dire par concaténation. Là encore, on aura des files persistantes (non mutables) et ceci explique la présence de la file dans les arguments (on aurait aussi pu utiliser une référence de liste).
   Le dernier souci est que l'on ne doit pas faire un parcours en largeur complet. On doit l'interrompre quand on a visité n sites, c'est à dire quand le dictionnaire contient n éléments. On pourrait gérer une référence d'entier donnant le nombre de sites visités. J'ai préféré ajouter un argument entier à ma fonction donnant le nombre de site qu'il reste à ajouter (n au départ).
   J'écris finalement une fonction
   `remplir : int → string list → (string * string list) list → dictionnaire → (string * string list) list`
   Dans l'appel `remplir k file res dico`, les différents arguments ont été expliqués plus haut. `res` correspond à la liste en construction (que l'on renvoie quand on a terminé). On pourrait éviter cet argument supplémentaire (voir la fonction de la question suivante pour varier les plaisirs).

```
let crawler_bfs n u =
    let rec remplir k file res dico =
        match file with
       |[] -> res
       |c::q -> if k=0 then res
                else if not(contient c dico) then begin
```

```
                        let l=recupere_liens c in
                        remplir (k-1) (q@l) ((c,l)::res) (ajoute c 0 dico)
                        end
              else remplir k q res dico
    in remplir n [u] [] (dictionnaire_vide());;
```

6. La situation est similaire mais il ne faut pas utiliser une file mais une pile. On utilise encore une liste mais cette fois on ajoute et retire des éléments par la gauche. Je donne ici une version on n'utilise pas d'argument supplémentaire `res` pour la fonction auxiliaire.

```
let crawler_dfs n u =
     let rec parcourt pile dico k=
         match pile with
        |[] -> []
        |c::q -> if k=0 then []
                 else if contient c dico then parcourt q dico k
                 else begin
                      let l=recupere_liens c in
                      (c,l)::(parcourt (l@pile) (ajoute c 0 dico) (k-1))
                      end
     in parcourt [u] (dictionnaire_vide()) n;;
```

7. Les fonctions `aplatir` et `unique` permettent d'obtenir la liste des sommets du graphe (naturellement numérotés par leurs positions dans la liste) et un dictionnaire permettant s'associer une URL au numéro du sommet du graphe (connaissant une URL, on n'a pas à parcourir la liste pour connaître le numéro du sommet associé : il est donné par le dictionnaire).
   Pour chaque couple (c, l) du crawl, il faut ajouter au graphe l'arc de c vers chaque élément de l (du numéro de c vers le numéro de chaque élément de l). On doit donc disposer d'une fonction `miseajour : int → (string list) → unit`
   qui prend en argument un entier (le numéro de c) et une liste (l) et qui ajoute au graphe ces arcs.
   La fonction `parcourt : (string * string list) list → unit` est la fonction récursive de parcours du crawl.

```
let construit_graphe crawl =
    let (s,dico)=unique (aplatir crawl) in
    let n=list_length s in
    let g=make_matrix n n 0 in
    let rec miseajour i l =
        match l with
        [] -> ()
       |ch::q -> let j=valeur ch dico in
                 g.(i).(j) <- g.(i).(j) + 1 ;
                 miseajour i q
    in
    let rec parcourt crawl =
       match crawl with
       [] -> ()
      |(c,l)::q -> let i=valeur c dico in
                   miseajour i l ;
                   parcourt q
    in parcourt crawl;
```

```
s,g;;
```

On pourrait utiliser la fonction `do_list` pour effectuer l'itération sur les éléments des listes.

Calcul de PageRank

Dans toute cette partie, on travaille avec des éléments de type `float`. On devra donc utiliser les opérateur arithmétiques sur les flottants et transformer tous les entiers en flottants. Cela alourdit considérablement le code.

8. Il suffit de suivre les consignes.

```
let surf_aleatoire d G =
    let n=vect_length G in
    let M=make_matrix n n 0. in
    for i=0 to n-1 do
       let k=ref 0 in
       for j=0 to n-1 do k:= !k+G.(i).(j) done ;
       if !k=0 then
            for j=0 to n-1 do M.(i).(j) <- 1./.(float_of_int n) done
       else
            for j=0 to n-1 do
                   M.(i).(j) <- (1.-.d)*.(float_of_int G.(i).(j))
                                    /.(float_of_int !k)+.d/.(float_of_int n)
            done ;
       done ;
    M;;
```

9.
```
let multiplie v M =
    let n=vect_length v in
    let w=make_vect n 0. in
    for j=0 to n-1 do
       for i=0 to n-1 do
           w.(j) <- w.(j)+.v.(i)*.M.(i).(j)
           done;
       done;
    w;;
```

10. On commence par écrire une fonction de calcul de la norme de la différence de deux vecteurs (distance entre ces vecteurs).

```
let norme v w =
   let k=ref 0. in
   for i=0 to (vect_length v - 1) do
       if v.(i)-.w.(i) > 0. then k:= !k +. v.(i) -. w.(i)
       else k:= !k +. w.(i) -. v.(i)
       done ;
   !k;;
```

On écrit une fonction auxiliaire `itere` qui prend en argument un vecteur `v` et réalise le processus à partir de ce vecteur. Il suffit de l'appeler avec le bon vecteur initial.

```
let pagerank theta M =
    let rec itere v =
       let w=multiplie v M in
       if (norme v w) <= theta then w
       else itere w
    in let n=vect_length M
    in itere (make_vect n (1./.(float_of_int n))) ;;
```

11. La fonction auxiliaire `construireliste` prend en argument la liste des sommets (`string list`) et renvoie la liste (non ordonnée) des couples sommet/score, le score étant contenu dans le vecteur v obtenu avec `pagerank`. Il faut pouvoir, quand on rencontre un sommet sous forme d'URL, connaître le numéro du sommet. C'est pourquoi la fonction auxiliaire a un second argument de type `int` donnant le numéro du prochain sommet à traiter.

```
let calcule_pagerank d theta crawl =
    let (s,G)=construit_graphe crawl in
    let M=surf_aleatoire d G in
    let v=pagerank theta M in
    let rec construireliste s i =
        match s with
       |[] -> []
       |c::q -> (c,v.(i))::(construireliste q (i+1))
    in tri_fusion (construireliste s 0);;
```

2 Automates probabilistes

12. Le mot vide n'est l'étiquette d'aucun chemin acceptant et donc

$$Pr(\varepsilon) = 0$$

$q_0 \xrightarrow{\mathbf{0}} q_1$ est le seul chemin acceptant d'étiquette $\mathbf{0}$ et sa probabilité vaut $1/4$. On a donc

$$Pr(\mathbf{0}) = \frac{1}{4}$$

Il y a deux chemins acceptants d'étiquette $\mathbf{010}$ qui sont

$$q_0 \xrightarrow{\mathbf{0}} q_0 \xrightarrow{\mathbf{1}} q_0 \xrightarrow{\mathbf{0}} q_1 \text{ et } \; q_0 \xrightarrow{\mathbf{0}} q_1 \xrightarrow{\mathbf{1}} q_0 \xrightarrow{\mathbf{0}} q_1$$

et on a ainsi

$$Pr(\mathbf{010}) = \frac{3}{4}.1.\frac{1}{4} + \frac{1}{4}.1.\frac{1}{4} = \frac{1}{4}$$

13. Je propose de prouver un résultat plus général. Si $q \in Q$, j'appelle *q-chemin pour u* un chemin d'étiquette u d'état initial q. On parle aussi de q-chemin acceptant (ou non-acceptant) pour u selon que l'état final du chemin est (ou n'est pas) dans F. Je pose ensuite

$$Pr_q(u) := \sum_{\rho \ q\text{-chemin acceptant pour } u} Pr(\rho)$$

Je montre alors par récurrence sur la taille de u que

$$\forall q \in Q, \ Pr_q(u) = 1 - \sum_{\rho \ q\text{-chemin non-acceptant pour } u} Pr(\rho)$$

- Initialisation : ε est l'unique mot vide. Soit $q \in Q$. Le seul q-chemin pour ε est le chemin de longueur nulle q.
  Si ce chemin est q-acceptant ($q \in F$) on a $Pr_q(u) = 1$ et la somme des probabilités des q-chemin non-acceptant pour ε est bien nulle (il n'y a pas de tel chemin).
  Sinon, $Pr_q(u) = 0$ et la somme des probabilités des q-chemin non-acceptant pour ε est vaut 1 (probabilité du chemin q).
  Dans les deux cas, on a la formule demandée.
- Hérédité : supposons le résultat vrai pour les mot de longueur $n \geq 0$ donné. Considérons un mot u de longueur $n+1$. On peut l'écrire $u = xv$ avec v mot de longueur n et $x \in \{\mathbf{0}, \mathbf{1}\}$.
  Soit $q \in Q$; les q-chemins pour u sont du type

$$q \xrightarrow{x} q' \xrightarrow{v} q''$$

  et on a donc

$$Pr_q(u) = \sum_{q' \in Q} Pr(q \xrightarrow{x} q') Pr_{q'}(v)$$

  Par hypothèse de récurrence, on a

$$\forall q' \in Q,\ Pr_{q'}(v) = 1 - \sum_{\rho\ q'\text{-chemin non-acceptant pour } v} Pr(\rho)$$

  On en déduit que

$$\begin{aligned} Pr_q(u) &= \sum_{q' \in Q} Pr(q \xrightarrow{x} q') - \sum_{q' \in Q} \left(Pr(q \xrightarrow{x} q') \sum_{\rho\ q'\text{-chemin non-acceptant pour } v} Pr(\rho) \right) \\ &= 1 - \sum_{\substack{q' \in Q \\ \rho\ q'\text{-chemin non-acceptant pour } v}} (Pr(q \xrightarrow{x} q') Pr(\rho)) \\ &= 1 - \sum_{\rho\ q\text{-chemin non-acceptant pour } u} Pr(\rho) \end{aligned}$$

  Ceci prouve le résultat au rang $n + 1$.

Il suffit d'appliquer le résultat avec $q = q_0$.

14. Soit u un mot se terminant par $\mathbf{1}$. Un chemin dans $\mathcal{A}_0$ de q_0 à q_1 et d'étiquette u a une dernière transition de probabilité nulle et donc une probabilité nulle. Ainsi, $Pr(u) = 0$.
  Soit u un mot se terminant par $\mathbf{0}$. Un chemin bouclant sur q_0 et passant en q_1 lors de la lecture de la dernière lettre est composé de transitions de probabilités toutes non nulles et a donc une probabilité non nulle. On a alors $Pr(u) > 0$.
  Le mot vide est de probabilité nulle dans $\mathcal{A}_0$.
  Finalement, on a montré que

$$\text{les mots } u \text{ tels que } Pr(u) = 0 \text{ pour } \mathcal{A}_0 \text{ sont ceux de } \{\varepsilon\} \cup \Sigma^* \mathbf{1}.$$

  Pour tout mot u, il existe un chemin non-acceptant pour u et de probabilité > 0 (celui qui boucle sur q_0). Ainsi, la somme des longueur de ces chemins est > 0. Avec la question précédente, $Pr(u) < 1$ et

$$\text{il n'existe pas de mot } u \text{ tels que } Pr(u) = 1 \text{ pour } \mathcal{A}_0.$$

15. La question précédente montre que le langage cherché est

$$\Sigma^* \mathbf{0}$$

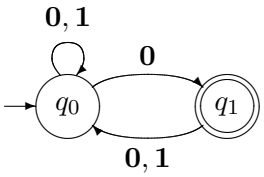
16. Notons $\mathcal{A}' = (Q, q_0, F, \gamma)$ où $\gamma \subset Q \times \Sigma \times Q$ est défini par

$$\gamma = \{(q, \alpha, q')/\ Pr(q, \alpha, q') > 0\}$$

On a un automate non déterministe dont on va montrer qu'il reconnaît exactement les mots u dont la probabilité $Pr(u)$ pour $\mathcal{A}$ est non nulle.

- Soit u un mot reconnu par $\mathcal{A}'$. Il existe alors un chemin dans $\mathcal{A}'$ d'origine q_0 et d'extrémité dans F d'étiquette u. Par définition de γ, le chemin similaire dans $\mathcal{A}$ est composé de transitions de probabilités > 0 et a donc une probabilité > 0. Ainsi $Pr(u) > 0$ (c'est la somme de quantités ≥ 0 dont une au moins est > 0).
- Réciproquement, soit u tel que $Pr(u) > 0$. Il y a alors au moins un chemin acceptant pour u dans $\mathcal{A}$ et de probabilité > 0 et donc composé de transitions de probabilités > 0. Par définition de γ, le même chemin existe dans $\mathcal{A}'$ et va de q_0 à un élément de F. Ainsi, $u \in \mathcal{A}'$.

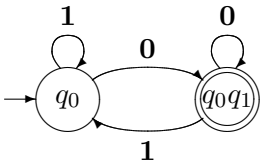
17. Dans le cas de $\mathcal{A}_0$, on obtient



La table du déterminisé de cet automate (par la méthode des sous-parties) est

	q_0	q_0, q_1
0	q_0, q_1	q_0, q_1
1	q_0	q_0

et sa représentation graphique est



18. Soit L un langage rationnel. Il existe un automate déterministe complet $\mathcal{A} = (Q, q_0, F, \delta)$ qui reconnaît ce langage. Considérons la fonction Pr définie par

$$\forall q \in Q,\ \forall \alpha \in \Sigma,\ Pr(q, \alpha, \delta(q, \alpha)) = 1$$

les autres valeurs prises par Pr étant nulles. Ceci revient à donner à chaque transition qui existe dans $\mathcal{A}$ la probabilité 1. Notons $\mathcal{A}' = (Q, q_0, F, Pr)$. Le fait que $\mathcal{A}$ est déterministe nous permet de voir que $\mathcal{A}'$ est déterministe (la somme des probabilités des transitions issues d'un état donné q et étiquetée par une lettre donnée α vaut 1).
Par construction, les chemins dans $\mathcal{A}'$ sont tous de probabilité 1. $Pr(u) > 0$ équivaut à l'existence d'un chemin acceptant pour u dans $\mathcal{A}'$ et donc au fait que le chemin dans $\mathcal{A}$ d'origine q_0 et d'étiquette u se termine dans un élément de F, c'est à dire au fait que u est reconnu par $\mathcal{A}$. On a ainsi

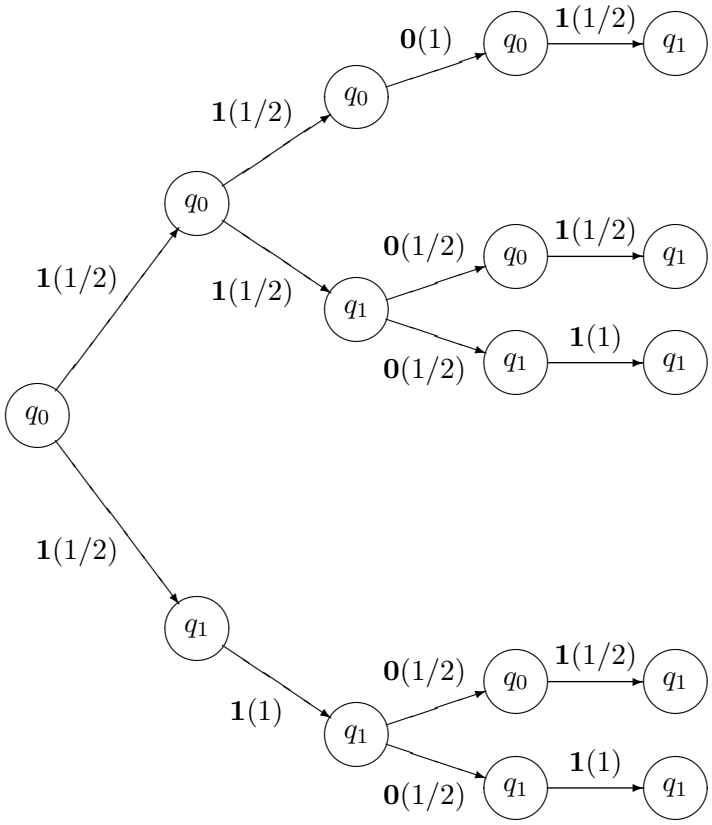
$$L = \mathcal{L}_0(\mathcal{A}')$$

et L est stochastique.

19.
$$Pr(q_0 \xrightarrow{\mathbf{1}} q_0 \xrightarrow{\mathbf{1}} q_1 \xrightarrow{\mathbf{1}} q_1 \xrightarrow{\mathbf{0}} q_0 \xrightarrow{\mathbf{1}} q_1) = \frac{1}{2} \times \frac{1}{2} \times 1 \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{16} = \underline{0,\mathbf{0001}}_2$$

20. $q_0 \xrightarrow{\mathbf{1}} q_1 \xrightarrow{\mathbf{0}} q_1$ est le seul chemin acceptant pour **10** et sa probabilité est $\frac{1}{4}$. Ainsi
$$Pr(\mathbf{10}) = \frac{1}{4} = \underline{0,\mathbf{01}}_2$$

21. On peut représenter de manière arborescente les différents chemins acceptants pour **1101** dans $\mathcal{A}_1$:



On a alors
$$Pr(\mathbf{1101}) = \frac{1}{8} + \frac{1}{16} + \frac{1}{8} + \frac{1}{8} + \frac{1}{4} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = \underline{0,\mathbf{1011}}_2$$

22. On a $Pr(\varepsilon) = 0$ dont une écriture finie en base 2 est 0. Montrons maintenant que
$$\forall u = \alpha_1 \dots \alpha_n \in \Sigma^n,\ Pr(u) = \underline{0,\alpha_n \dots \alpha_1}_2$$

On procède pour cela par récurrence sur n. Si $u = \alpha_1 \dots \alpha_n$, je note $N_u = \underline{0,\alpha_n \dots \alpha_1}_2$.

- On a $Pr(\mathbf{0}) = 0 = \underline{0,\mathbf{0}}_2$ et $Pr(\mathbf{1}) = \frac{1}{2} = \underline{0,\mathbf{1}}_2$ ce qui montre le résultat pour $n = 1$.
- Supposons le résultat vrai jusqu'à un rang $n \geq 1$. Soit $v = \alpha_1 \dots \alpha_n \alpha_{n+1}$; on pose $u = \alpha_1 \dots \alpha_n$. Par hypothèse de récurrence, la lecture de u depuis q_0 amène en q_1 avec probabilité N_u et donc en q_0 avec probabilité $1 - N_u$. On distingue deux cas.
  - Si $\alpha_{n+1} = 0$, la lecture de v mènera en q_1 si et seulement celle de u à amené en q_1 et qu'on emprunte la transition de q_1 vers lui même pour la dernière lettre. La probabilité est alors de $\frac{1}{2}N_u = N_v$ (multiplier par 1/2 décale toutes les "décimales").

- Si $\alpha_{n+1} = 1$, les bon chemins sont ceux qui mènent par v à q_0 puis passent à q_1 ou qui mènent par v à q_1 et y restent. Ceci advient avec probabilité $\frac{1}{2}(1 - N_u) + N_u = \frac{1}{2} + \frac{1}{2}N_u = N_v$ (on décale les "décimales" par le facteur $1/2$ et on ajoute une première "décimale" égale à 1 par l'ajout de $1/2$).

On obtient ainsi le résultat au rang $n + 1$.

23. Le mot vide est de probabilité nulle et donc n'est pas dans $\mathcal{L}_\eta(\mathcal{A}_1)$.
Soit $u = \alpha_1, \dots \alpha_n$ un mot non vide. Il est dans $\mathcal{L}_\eta(\mathcal{A}_1)$ si $Pr(u) > \eta$ et la question précédente montre que ceci équivaut à $\underline{0, \alpha_n \dots \alpha_1}_2 > \eta$.

24. Pour conclure, il reste à montrer qu'il existe $\eta \in [0, 1[$ tel que $\mathcal{L}_\eta(\mathcal{A}_1)$ n'est pas rationnel.
Or, le nombre de langages rationnels sur Σ est dénombrable (en effet, $Rat(\Sigma)$ est défini par récurrence à partir d'un nombre fini de cas de base et par application de trois règles ; l'ensemble R_n des langages rationnels obtenus par application d'au plus n fois une règle est fini ; $Rat(\Sigma)$ est la réunion, dénombrable, des R_n).
On pourra conclure si on montre qu'il existe un nombre infini non dénombrable de $\mathcal{L}_\eta(\mathcal{A}_1)$. Ceci est vrai car si $\eta < \eta'$ alors $\mathcal{L}_{\eta'}(\mathcal{A}_1) \subset \mathcal{L}_\eta(\mathcal{A}_1)$ et ceci est une inclusion stricte puisque l'ensemble des nombres ayant une écriture finie en base 2 est dense dans $[0, 1]$ (même preuve que dans le cs décimal) et qu'on peut donc trouver un nombre ayant une écriture finie en base 2 dans $]\eta, \eta'[$.