DS 1 d'option informatique - MPSI - Correction

Question 1 – Il s'agit d'une question de cours.

```
let rev li0 =
  let rec aux li acc = match li with
    | [] -> acc
    | e :: q -> aux q (e::acc)
  in
  aux li0 [];;
```

Question 2 –

```
let verif (li: int list): bool =
  let pred e =
    e >= 0 && e <= 9
  in
  List.for_all pred li;;
```

Question 3.a –

```
(* La fonction aux renvoie une liste contenant les chiffres de n en
   commençant par le chiffre de poids faible. *)
let intToList (n0: int): int list =
  if n0 < 0 then failwith "intToList: n < 0";
  let rec aux = function
    | n when n < 10 -> [n]
    | n -> (n mod 10) :: aux (n/10)
  in
  rev (aux n0);;
```

Question 3.b –

```
(* La fonction aux prend en entrée une liste contenant les chiffres de
   n en commençant par le chiffre de poids faible. *)
let listToInt (li0: int list) =
  let rec aux = function
    | [] -> 0
    | e :: q -> e + 10 * aux q
  in
  match li0 with
  | [] -> failwith "listToInt: liste vide"
  | 0 :: _ :: _ -> failwith "listToInt: premier élément nul"
  | li0 when not (verif li0) ->
     failwith "listToInt: l'un des éléments n'est pas un chiffre"
  | li0 -> aux (rev li0);;
```

Question 4 – Soit $n > 0$ et ℓ la taille de `L(n)`, alors ℓ est le nombre de chiffres dans l'écriture en base 10 de n. Donc :

$$10^{\ell-1} \leqslant n < 10^{\ell}$$

C'est à dire :

$$\ell - 1 \leqslant \log_{10}(n) < \ell$$

Par définition de la partie entière :

$$\ell - 1 = \lfloor \log_{10}(n) \rfloor$$

En conclusion :

La taille de `L(n)` est 1 si $n = 0$ et $\lfloor \log_{10}(n) \rfloor + 1$ si $n > 0$.

Question 5.a –

```
let rec somme (li: int list): int = match li with
  | [] -> 0
  | e :: q -> e + somme q;;
```

Question 5.b –

```
let rec persAdd (n: int): int =
  if n < 0 then failwith "persAdd: n < 0";
  if n < 10 then 0 else 1 + persAdd (somme (intToList n));;
```

Question 6.a – On le montre par récurrence sur $k \in \mathbb{N}$.

⋆ Pour $k = 0$, l'entier $n = 0$ convient

⋆ Soit $k \geqslant 0$, on suppose qu'il existe un entier $n \in \mathbb{N}$ dont la persistance additive est k et on montre la propriété au rang $k + 1$. Pour cela, il suffit de définir $n' \in \mathbb{N}$ un entier dont la somme des chiffres vaut n. On peut par exemple choisir pour n' l'entier avec n chiffres tous égaux à 1 (par exemple pour $n = 5$, on aurait $n' = 11111$).

⋆ Finalement, d'après le principe de récurrence, pour tout $k \in \mathbb{N}$, il existe un entier $n \in \mathbb{N}$ dont la persistance additive est k.

Question 6.b – Pour construire la liste demandée, on part de $n_0 = 9\,897\,787$ dont la persistance additive est 3 (voir l'énoncé). L'entier n sera un entier tel que $S(n) = n_0$. Plutôt que de construire une liste ne contenant que des uns (comme dans la question précédente), on construit une liste ne contenant que des neufs, sauf éventuellement pour le premier élément.

```
(* La fonction aux prend en entrée un entier m et renvoie une liste
   dont la somme des éléments vaut m.
   La fonction aux est récursive terminale pour éviter les stack
   overflow (une fonction non terminale aurait été acceptée le jour du
   DS).
 *)
let exPersAdd4 (): int list =
  let rec aux (acc: int list): int -> int list = function
    | n when n < 0 -> failwith "exPersAdd4: n < 0"
    | n when n < 10 -> n :: acc
    | n -> aux (9::acc) (n-9)
  in
  aux [] 9_897_787;;
```

Question 7 – Voici les termes de la suite $(v_i)_{i\in\mathbb{N}}$ pour $n = 678$:

$$v_0 = 678 \qquad v_1 = 336 \qquad v_2 = 54 \qquad v_3 = 20 \qquad v_4 = 0 \qquad \dots$$

Ainsi, la persistance multiplicative de 678 est 4.

Question 8 – Il s'agit de montrer que pour tout $n \in \mathbb{N}$ avec $n \geqslant 10$, le produit des chiffres de n est strictement inférieur à n. Décomposons n en base 10 :

$$n = \sum_{k=0}^{\ell} a_k 10^k$$

où $a_k \in [\![0; 9]\!]$ pour tout $k \in [\![0; \ell]\!]$ et $a_\ell > 0$. Le produit des chiffres de n est égal à :

$$\prod_{k=0}^{\ell} a_k .$$

On a donc :

$$n = \sum_{k=0}^{\ell} a_k 10^k \geqslant a_\ell 10^\ell = a_\ell \prod_{k=0}^{\ell-1} 10 > a_\ell \prod_{k=0}^{\ell-1} 9 \geqslant a_\ell \prod_{k=0}^{\ell-1} a_k = \prod_{k=0}^{\ell} a_k .$$

Notons en particulier que l'inégalité stricte vient de $\ell \geqslant 1$ et $a_\ell > 0$ qui sont des conséquences de l'hypothèse $n \geqslant 10$.

Question 9 –

```
let prodChiffres (n: int): int =
  let res = ref 1 in
  let f e =
    res := !res * e
  in
  List.iter f (intToList n);
  !res;;
```

Question 10.a – Pour cela il suffit de remplir le tableau dans l'ordre croissant des indices.
Soit $n \geqslant 10$. Si on suppose que `tabPM.(i)` a été caculé pour tout `i < n` alors il suffit de calculer `m` le produit des chiffres de `n` (avec un appel à la fonction `prodChiffres`) et d'ajouter un à la persistance mutiplicative de `m` récupérée dans `tabPM`. On a donc un appel à `prodChiffres` pour chaque $n \in [\![10; n_0]\!]$ ce qui est bien en $\mathcal{O}(n_0)$.

Question 10.b –

```
let makeTabPM (n0: int): int array =
  let tabPM = Array.make (n0+1) 0 in
  for n = 10 to n0 do
    let pm = 1 + tabPM.(prodChiffres n) in
    tabPM.(n) <- pm;
  done;
  tabPM;;
```

Question 10.c – On a exactement un appel à `prodChiffres` pour chaque $n \in [\![10; n_0]\!]$. Ainsi le nombre d'appels à `prodChiffres` vaut :

- 0 si $n_0 < 10$.
- $n_0 - 9$ si $n_0 \geqslant 10$.

Question 11 – Pour $n \geqslant 10$, calculons les termes de la suite $(v_i)_{i\in\mathbb{N}}$ (jusqu'à obtenir un entier avec un seul chiffre) :

$10 \to 0$	$11 \to 1$	$12 \to 2$	$13 \to 3$	$14 \to 4$
$15 \to 5$	$16 \to 6$	$17 \to 7$	$18 \to 8$	$19 \to 9$
$20 \to 0$	$21 \to 2$	$22 \to 4$	$23 \to 6$	$24 \to 8$
$25 \to 10 \to 0$	$26 \to 12 \to 2$	$27 \to 14 \to 4$	$28 \to 16 \to 6$	$29 \to 18 \to 8$
$30 \to 0$	$31 \to 3$	$32 \to 6$	$33 \to 9$	$34 \to 12 \to 2$
$35 \to 15 \to 5$	$36 \to 18 \to 8$	$37 \to 21 \to 2$	$38 \to 24 \to 8$	$39 \to 27 \to 14 \to 4$

Ainsi, $\boxed{N_2 = 25 \text{ et } N_3 = 39}$.

Question 12 –

```
(* Renvoie le plus petit indice i tel que tabPM.(i) = x.
 * Renvoie Array.length tabPM si x n'appartient pas à tabPM. *)
let index (tabPM: int array) (x: int): int =
  let i = ref 0 in
  while !i < Array.length tabPM && tabPM.(!i) <> x do
    incr i;
  done;
  !i;;
```

```
let getNk (k: int): int =
  let n0 = ref 1 in
  let res = ref 2 in
  while !res = !n0 + 1 do
    n0 := !n0 * 10;
    let tabPM = makeTabPM !n0 in
    res := index tabPM k;
  done;
  !res;;
```

Question 13.a –

⋆ Supposons que $L(N_k)$ contienne un 0. Alors le produit des chiffres de N_k est égal à 0. Donc la persistance mutiplicative de N_k est au plus 1 ce qui contredit l'hypothèse $k \geqslant 3$.

⋆ Supposons que $L(N_k)$ contienne un 1 (et aucun 0). Comme la persistance multiplicative de N_k vaut $k \geqslant 3 \geqslant 2$, la liste $L(N_k)$ contient au moins un autre chiffre supérieur ou égal à 2. On note L' la liste $L(N_k)$ dans laquelle l'un des 1 a été supprimé. Alors $L' = L(n')$ pour un certain $n' \in \mathbb{N}$ et donc $P(n') = P(N_k)$. On a deux cas :

- Si $L(n')$ est de taille 1, alors $P(N_k) = P(n') = n'$. Donc la persistance multiplicative de N_k est 1, ce qui contredit l'hypothèse $k \geqslant 3$.
- Sinon, $L(n')$ est de taille au moins 2 et donc la persistance mutiplicative de n' est égale à la persistance multiplicative de N_k (qui vaut k). Comme n' possède un chiffre de moins que N_k, on a $n < N_k$. L'inégalité précédente est une contradiction avec la minimalité de N_k parmi les entiers dont la persistance multiplicative est k.

Question 13.b – Supposons que $L(N_k)$ ne soit pas triée par ordre croissant (et ne contienne pas de 0). On note L' la liste triée contenant les mêmes éléments que $L(N_k)$ et $n' \in \mathbb{N}$ l'entier tel que $L' = L(n')$. On a $P(N_k) = P(n')$ et donc la persistance mutiplicative de n' est égale à la persistance mutiplicative de N_k (qui vaut k). De plus, si on note i le premier indice tel que $L(n')[i] \neq L(N_k)[i]$ (on utilise ici la notation $li[i]$ de Python), alors $L(n')[i] < L(N_k)[i]$ (car $L(n')$ est triée) et donc $n' < N_k$. L'inégalité précédente est une contradiction avec la minimalité de N_k parmi les entiers dont la persistance multiplicative est k.

Question 13.c –

$\star$ Supposons que $L(N_k)$ contienne deux fois le chiffre 2 (et aucun 0). On note L' la liste dans laquelle deux des occurrences de 2 ont été supprimées et dans laquelle un 4 a été ajouté. Soit $n' \in \mathbb{N}$ l'entier tel que $L' = L(n')$, alors $n' < N_k$ ce qui constitue une contradiction (même raisonnement que dans les questions précédentes).

$\star$ Pour 3, 4 et 6 :

- Si $L(N_k)$ contient deux fois 3, on remplace ces deux occurences par un 9.
- Si $L(N_k)$ contient deux fois 4, on remplace la première occurence par un 2 et la deuxième par 8.
- Si $L(N_k)$ contient deux fois 6, on remplace la première occurence par un 4 et la deuxième par 9.

Dans tous les cas, on obtient un entier $n' < N_k$ avec la même persistance multiplicative que N_k ce qui est impossible.

Question 13.d – Supposons que $L(N_k)$ contienne un chiffre pair et 5. Alors $P(N_k) \equiv 0 \mod 10$, c'est à dire que le dernier chiffre de $P(N_k)$ est 0. Ainsi, $P(P(N_k)) = 0$ et donc la persistance multiplicative de N_k est au plus 2, ce qui contredit l'hypothèse $k \geqslant 3$.

Question 14 – Supposons que le chiffre de poids fort de N_k soit un 2. Par la question précédente, on sait que $L(N_k)$ est triée par ordre croissant, ne peut pas contenir d'autre 2 et ne peut pas contenir de 5.

Si le 2ème chiffre de poids fort de N_k était un 3 (resp. 4), alors on pourrait remplacer les deux chiffres de poids fort par 6 (resp. 8) pour obtenir un nombre strictement plus petit avec la même persistance multiplicative ; ce qui constitue une contradiction.

Ainsi, le 2ème chiffre de poids fort de N_k est 6, 7, 8 ou 9. Comme $L(N_k)$ est triée et que 6 y apparaît au plus une fois, N_k est de la forme $267^\star8^\star9^\star$ ou $27^\star8^\star9^\star$.

Question 15.a –

```
let rec ajout (c: int) (li: int list list) = match li with
  | [] -> []
  | e :: q -> (c :: e) :: (ajout c q);;
```

Question 15.b –

```
let rec generer (n: int) (li: int list): int list list = match n, li with
  | n, _ when n < 0 -> failwith "generer: n < 0"
  | 0, _ -> [[]]
  | n, [] -> []
  | n, c :: q ->
     let res1 = ajout c (generer (n-1) li) in
     let res2 = generer n q in
     res1 @ res2;;
```

Question 16 –

```
(* Calcule la persistance multiplicative de son argument *)
let rec persMult (n: int): int =
  if n < 0 then failwith "persMult: n < 0";
  if n < 10 then 0 else 1 + persMult (prodChiffres n);;
```

```
(* Renvoie le plus petit élément d'une liste *)
let rec mini = function
  | [] -> failwith "Liste vide"
  | [e] -> e
  | e :: q -> let m = mini q in
              if m < e then m else e;;
```

```
(* On génère toutes les listes ayant la bonne forme de taille n = 2,
   puis n = 3, puis n = 4 jusqu'à trouver un nombre dont la persistance
   multiplicative est k *)
let getNkBis (k: int): int =
  if k < 0 then failwith "getNkBis: k < 0";
  if k >= 12 then failwith "getNkBis: k >= 12";
  if k = 0 then 0 else if k = 1 then 10 else if k = 2 then 25 else begin
      let n = ref 2 in
      let li = ref [] in
      while !li = [] do
        li := (ajout 2 (ajout 6 (generer (!n-2) [7;8;9]))) @ !li;
        li := (ajout 2 (generer (!n-1) [7;8;9])) @ !li;
        li := (ajout 3 (ajout 4 (generer (!n-2) [7;8;9]))) @ !li;
        li := (ajout 3 (ajout 5 (generer (!n-2) [7;9]))) @ !li;
        li := (ajout 3 (generer (!n-1) [7;8;9])) @ !li;
        li := (ajout 4 (generer (!n-1) [7;8;9])) @ !li;
        li := (ajout 5 (generer (!n-1) [7;9])) @ !li;
        li := (ajout 6 (generer (!n-1) [7;8;9])) @ !li;
        li := (generer !n [7;8;9]) @ !li;
        (***)
        li := List.filter (fun l -> persMult (listToInt l) = k) !li;
        incr n;
      done;
      mini (List.map (fun l -> listToInt l) !li);
    end;;
```