Travaux pratiques 3

Polynômes

MPII : Ocaml

Dans ce T.P. nous allons manipuler les polynômes en choisissant une représentation **creuse** : on ne garde que les coefficients non nuls. On se restreint à des polynômes à coefficients **entiers**, on évite ainsi les opérations `+.`, `-.`, `*.`, `/.`

Définition 1 : convention typographique

Les variables dans le langage seront notées avec une police à espacement constant : `variable`
leur valeur dans les phrases sont notées en italique : variable.
Dans le cas de noms indicés a_i dans le texte sera noté `ai` dans les codes.

- Un polynôme est représenté par une liste de couples d'entiers (`(int * int)lis`)
  `[(ap, np); ...; (a1, n1); (a0, n0)]`
  avec les conditions $a_i \neq 0$ pour tout i et $n_p > n_{p-1} > \cdots > n_1 > n_0 \geq 0$.
  La liste représente le polynôme $a_p X^{n_p} + a_{p-1} X^{n_{p-1}} + \cdots + a_1 X^{n_1} + a_0 X^{n_0}$.
- Par exemple $P = 1 - 2X + 4X^3$ est représenté par `[(4, 3); (-2, 1); (1, 0)]`.
  Les parenthèses pour délimiter un couple ne sont pas obligatoires, on a alors une succession de ponctuations : `[4, 3; -2, 1; 1, 0]`.
- En particulier le polynôme nul est représenté par la liste vide `[]`.

À retenir : Pattern matching

Dans certaines fonctions, le motif terminal de la récursivité sera le cas d'une liste à un élément, la liste vide est simplement traitée à part. On utilisera alors un pattern-matching avec 3 motifs :

```
match liste with
|[] ->
|[x] ->
|x::y::q ->
```

Dans le cas d'un type composé comme ici, on peut rechercher un motif plus spécialisé que celui d'une liste :

```
match poly with
|[] -> ...
|(a, n)::q -> ...
```

On accède ainsi aux composantes du couples sans avoir besoin d'utiliser `fst` et `snd`.

I Premières fonctions

Exercice 1 - Vérification
Écrire une fonction `validation p` qui vérifie qu'une liste de couples d'entiers est une bonne représentation d'un polynôme : non nullité des coefficients et stricte décroissance des puissances.

Dans toute la suite on supposera que les listes de couples d'entiers sont valides sans avoir besoin de le vérifier.

Exercice 2 - Degré
Écrire une fonction `degre p` qui renvoie le degré du polynôme, on convient de renvoyer -1 pour le polynôme nul.

Exercice 3 - Coefficient
Écrire une fonction `coefficient k p` qui renvoie le coefficient de X^k du polynôme ; ce coefficient sera nul si le polynôme ne contient pas de terme de la forme (a, k).

Exercice 4 - Évaluation
Écrire une fonction `evaluation p a` qui renvoie la valeur en a du polynôme P représenté par `p`. Il sera utile d'écrire une fonction qui calcule a^n à partir de a et de n.

Exercice 5 - Composition par X^k
Écrire une fonction `composition k p` où `p` désigne un polynôme $P = \sum_{i=0}^{d} a_i X^i$ et k en entier strictement positif et qui renvoie la liste représentant le polynôme $P(X^k) = \sum_{i=0}^{d} a_i X^{ik}$

Exercice 6 - Changement d'échelle
Écrire une fonction `dilatation b p` où `p` désigne un polynôme et b en entier et qui renvoie la liste représentant le polynôme $P(bX) = \sum_{i=0}^{d} a_i b^i X^i$

II Opérations

Exercice 7 - Somme de polynômes
Écrire une fonction `plus p1 p2` qui renvoie la représentation de la somme des polynômes représentés respectivement par `p1` et `p2`

Exercice 8 - Produit de polynômes
Écrire une fonction `fois p1 p2` qui calcule une représentation du produit des polynômes représentés respectivement par `p1` et `p2`.
Il peut être utile d'écrire une fonction qui calcule le produit d'un monôme par un polynôme, le monôme étant représenté par un couple d'entiers.

Exercice 9 - Dérivée
Écrire une fonction `derivee p` qui renvoie une représentation de P' quand P est le polynôme représenté par `p`.

II.1 Application

Les polynômes de Tchebychev sont définis par $T_n\big(\cos(x)\big) = \cos(nx)$.
On peut prouver qu'on a $T_0 = 1$, $T_1 = X$ et $T_n = 2X.T_{n-1} - T_{n-2}$ pour $n \geq 2$.

Exercice 10 - Polynômes de Tchebychev
Écrire une fonction `tch n` qui renvoie une représentation de T_n.
Bien que la récursivité ne soit pas méthode la plus efficace ici, on écrira une fonction récursive.

III Division euclidienne

Dans cette partie nous étudions la division de polynômes. Nous supposerons que le polynôme diviseur est non nul et **unitaire**, le coefficient du terme de plus haut degré doit être 1.

La division euclidienne du polynôme A par le polynôme B est le couple de polynômes Q et R tels que $A = B.Q + R$ et $\deg(R) < \deg(B)$.
On admet que de tels polynômes existent et sont uniques.

Pour calculer les termes de la division euclidienne

1. on détermine les degrés de A, p, et de B, q,
2. si on a $p < q$ alors $Q = 0$ et $R = A$,
3. sinon
   (a) on détermine le coefficient de plus haut degré de A, a,
   (b) on calcule $A_1 = A - a.X^{p-q}.B$,
   (c) on calcule récursivement $A_1 = B.Q_1 + R$,
   (d) on a $A = (a.X^{p-q} + Q_1).B + R$.

La récursivité est possible car le degré de A_1 est strictement inférieur à celui de A.

Exercice 11 - Division euclidienne
Écrire une fonction `division a b` qui renvoie des représentants du quotient et du reste de la division euclidienne de A par B respectivement représentés par `a` et `b`.

Les polynômes cyclotomiques sont des polynômes définis par C.F. Gauss en 1801
Ils sont employés dans plusieurs domaines mathématiques : théorie de Galois, construction à la règle et au compas, ... Ils sont définis par $\Phi_n = \prod_{k=1,k\wedge n=1}^{n} \big(X - e^{2ik\pi/n}\big)$.
On prouve que ce sont des polynômes à coefficients entiers, irréductibles dans $\mathcal{Q}[X]$.
On peut les construire récursivement, à l'aide de la propriété ;

$$\Phi_1 = X - 1 \quad \Phi_n = \frac{X^n - 1}{\prod_{k|n,k<n} \Phi_k}$$

Exercice 12 - Calcul
Écrire une fonction `cycloto n` qui renvoie Φ_n.

IV Application d'une fonction à une liste

Dans l'usage des listes, il y a des motifs de fonctions qui se répètent. Il existe des meta-fonctions (ou fonctionnelles) qui en gardent la structure ; ce seront des fonctions dont un paramètre est lui même une fonction.
Nous allons ici montrer l'une de ces fonctions : `map` Les fonctions `composition` de l'exercice 5 et `dilatation` de l'exercice 6 reviennent toutes deux à appliquer une fonction f à tous les termes de la liste `l`.
Par exemple, dans la fonction `composition`, on appliquer la fonction `let f (a, n)-> (a, k*n);;` à tous les termes de la liste.
On peut fonc l'écrire sous la forme

```
let composition  k p =
   let f (a, n) ->  (a, k*n) in
   let rec aux l =
      match l with
      |[] -> []
      |t::q -> (f t) :: (aux f q) in
   aux p;;
```

On y voit un motif général, qui est reconnu sous la forme

List.map

```
let rec map f l =
   match l with
   |[] -> []
   [t::q -> (f t) :: (map f q);;
```

On peut alors écrire (on notera la disparition de `rec`)

```
let composition  k p =
   let f (a, n) ->  (a, k*n) in
   List.map f p;;
```

Souvent la fonction à utiliser n'aura pas d'existence en dehors de son usage : on peut dans ce cas créer une fonction **anonyme** avec `fun`

```
let composition  k p =
   List.map (fun (a, n) ->  (a, k*n))  p;;
```

Voire même `let composition k = List.map (fun (a, n)-> (a, k*n));;`.

Exercice 13 - Changement d'échelle bis
Récrire ainsi la fonction `dilatation` de l'exercice 6.

Exercice 14 - Pour aller plus loin
Proposer une récriture de `derivee` de l'exercice 9.
Elle n'est pas tout-à-fait ced que l'on attend ; pourquoi ?
Pour avoir une définition fonctionnelle on pourra chercher la documentation de `List.filter`

Solutions

Solution de l'exercice 1 - Vérification

```
let rec validation =
   match p with
   |[] -> true
   |[(a, n)] -> a <> 0 && n >= 0
   |(a, n) :: (b, m) :: q -> a <> 0 && n > m && validation ((b, m)
       :: q);;
```

Solution de l'exercice 2 - Degré

```
let degre p =
   match p with
   |[] -> -1
   |[(a, n)]  :: q -> n;;
```

Solution de l'exercice 3 - Coefficient

```
let rec coefficient k p  =
   match p with
   |[] -> 0
   |(a, n) :: q when n > k -> coefficient k q
   |(a, n) :: q when n = k -> a
   |(a, n) :: q -> 0;;
```

On peut simplifier les deux cas qui donnent 0

```
let rec coefficient k p  =
   match p with
   |(a, n) :: q when n > k -> coefficient k q
   |(a, n) :: q when n = k -> a
   |(_ -> 0;;
```

Solution de l'exercice 4 - Évaluation

```
let rec puissance a k =
  if k = 0 then 1
           else a * (puissance a (k-1));;

let rec eval p x =
  match p with
  |[] -> 0
  |(a, n)::q -> a*(puissance x n) + (eval q x);;
```

Solution de l'exercice 5 - Composition par X^k

```
let rec composition k p =
   match p with
   |[] -> []
   |(a, n)::q -> (a, k*n) :: (composition k q);;
```

Solution de l'exercice 6 - Changement d'échelle

```
let rec dilatation k p =
   match p with
   |[] -> []
   |(a, n)::q -> (a * puissance b n, n) :: (dilatation b q);;
```

Solution de l'exercice 7 - Somme de polynômes

On prend le terme de plus haut de degré des deux polynômes puis on continue récursivement. Si les deux termes sont de même degré on doit étudier le cas particulier d'une somme nulle pour les coefficients.

```
let rec plus p1 p2 =
  match (p1, p2) with
  |[], _ -> p2
  |_, [] -> p1
  |(a1, n1)::q1, (a2, n2)::q2 when n1 > n2
                              -> (a1, n1) :: (plus q1 p2)
  |(a1, n1)::q1, (a2, n2)::q2 when n1 < n2
                              -> (a2, n2) :: (plus p1 q2)
  |(a1, n1)::q1, (a2, n2)::q2 when a1 + a2 = 0 -> (plus q1 q2)
  |(a1, n1)::q1, (a2, n2)::q2 -> (a1+a2, n1) :: (plus q1 q2);;
```

Solution de l'exercice 8 - Produit de polynômes

```
let rec p_monome (a, n) p =
  match p with
  |[] -> []
  |(b, m)::q -> (a*b, n+m) :: (p_monome (a, n) q);;

let rec fois p1 p2 =
  match p1 with
  |[] -> []
  |m::q -> plus (p_monome m p2) (fois q p2);;
```

Solution de l'exercice 9 - Dérivée

```
let rec derivee p =
   match p with
    |[] -> []
    |[a, 0] -> []
    |(a, n)::q -> (a*n, n-1) :: (derivee q);;
```

Solution de l'exercice 10 - Polynômes de Tchebychev

```
let rec tch n =
  match n with
  |0 -> [1, 0]
  |1 -> [1, 1]
  |n -> plus (fois [2, 1] (tch (n-1))) (fois [-1, 0] (tch (n-2)))
     ;;
```

Solution de l'exercice 11 - Division euclidienne

```
let rec division a b =
   match a, b with
   |_, [] -> failwith "Ceci ne devrait pas arriver"
   |[], _ -> [], []
   |(c, n)::aa, (d, m)::bb when n < m -> [], b
   |(c, n)::aa, (d, m)::bb ->
                  let a1 = plus a (fois [-c, n-m] b) in
                  let q1, r  = division a1 b in
                  ((c, n-m) :: q1), r;;
```

Solution de l'exercice 12 - Calcul

```
let quotient a b = fst(division a b)

let rec cycloto n =
   let rec calcul k p =
      if k > n/2 then p
      else if n mod k = 0
           then calcul (k+1) (quotient p (cycloto k))
           else calcul (k+1) p in
   calcul 1 [1, n; -1, 0];;
```

Solution de l'exercice 13 - Changement d'échelle bis

```
let dilatation b = List.map (fun (a, n) -> (a* (puissance b n), n)
   );;
```

Solution de l'exercice 14 - Pour aller plus loin

```
let derivee = List.map (fun (a, n) -> (a*n, n-1));;
```

Les termes de degré 0 vont donner un couple `(0, -1)` ce qui contrevient à deux conditions : on doit filtrer les termes de degré 0.

```
let derivee p =
   let p1 = List.filter (fun (a, n) -> n > 0) p in
   List.map (fun (a, n) -> (a*n, n-1)) p1;;
```