IAP1 - feuille de TP n0 2
Représentation creuse des polynômes -

1 Polynômes à une indéterminée

L'objet de cet exercice est la manipulation de polynômes à une variable, codés par des listes. On parle alors de polynômes creux. Un monôme est représenté par un couple dont le premier composant est le coefficient et le second composant le degré du monôme. Dans la suite le type `polynôme` désigne le type `int*int list`.

On impose deux conditions : les monômes d'un polynôme devront toujours être triés par degré décroissant ; aucun coefficient ne doit être nul. Le polynôme nul est représenté par la liste vide []. Les monômes ont tous des degrés différents.

Ces contraintes seront à prendre en considération dans les fonctions qui manipuleront les polynômes, il est, en effet, impossible de les *faire entrer* dans les déclarations de type. Ces contraintes doivent être respectées à chaque manipulation de polynôme. On appelle ces contraintes un *invariant de représentation.*

Par exemple le polynôme $X^5 - 2X^4 + 1$ sera représenté par la liste [(1,5);((-2),4); (1,0)]

1. Ecrivez la fonction `dg_max` : `polynôme` → `int`. Elle retourne le degré du polynôme, degré maximal des monômes. Par convention, on posera que le degré du polynôme nul est -1.

   Solution :

   ```
   type : polynôme -> int
   arg p
   pre : p polynôme bien formé (i.e. qui respecte l'invariant de représentation)
   post retourne le degré du polynôme
   *)
   let dg_max p = match p with
   | [] -> -1
   | (c,d)::_ -> c;;
   ```

   La précondition ici est importante car si on donne en argument un polynôme qui ne respecte pas l'invariant de représentation, le résultat peut ne pas être correct.

2. Écrivez une fonction `opposé`: `polynôme` → `polynôme` qui renvoie le polynôme dont les coefficients sont les opposés des coefficients du polynôme paramètre. On vérifiera (informellement) que l'invariant de reprsentation est bien conservé.

   Solution :

   ```
   (*interface oppose
   type : polynôme -> polynôme
   arg p
   pre : p polynôme bien formé
   post retourne le polynôme opposé de p
   *)
   let rec opposé p = match p with
   | [] ->  []
   | (c,d)::p' -> ((-c), d)::(opposé p')
   ```

   Le polynôme construit vérifie lui aussi l'invariant de représentation : seuls les coefficients sont modifiés (ils étaient non nuls, on en prend l'opposé donc ils restent non nuls!)

3. Pour rendre plus lisible ces polynômes, écrire la fonction `string_of_polynôme` qui les transforment en chaînes de caractères. Elle prend en paramètre le polynôme lui-même et le nom de l'indéterminée. Elle suit les règles usuelles de présentation des polynômes. Néanmoins, le monôme aX^n sera mis sous la forme $aX\hat{}n$ (le coefficient ne sera pas écrit s'il vaut 1). On commence par écrire la fonction `string_of_monôme` qui traduit un monôme en une chaîne de caractères. On utilise, dans le texte de cette fonction, la fonction prédéfinie `string_of_int`, qui transforme un entier en la chaîne de caractères correspondante. Par exemple, `string_of_int 4` vaut `"4"`.

```
(*interface string_of_monôme
type : (int*int list)* string  -> string
arg m, v  (nom de l'indéterminée du polynôme)
pre : monome bien formé
post transforme en une chaine de caracteres un monome
*)
let string_of_monôme (m,v) = match m with
| (c,  0) -> string_of_int c
| (1, 1) -> v
| (c, 1) -> (string_of_int c) ^ v
| (1,  d) -> v ^ "^" ^ (string_of_int d)
| (c, d) -> (string_of_int c) ^ v ^ "^" ^ (string_of_int d);;

(*interface string_of_polynôme
type : (int*int list) * string  -> string
arg m, v (nom de l'indéterminée du polynôme)
pre : polynôme bien formé
post transforme en une chaine de caracteres un
polynôme à un indéterminée nommée v
*)
let rec string_of_polynôme (p, v) = match p with
| [] -> ""
| m::p' ->
    let sm = string_of_monôme (m, v)
    and sp' = string_of_polynôme (p', v) in
    if sm = "" then sp' else
      if sp' = "" then sm else sm ^ "+" ^ sp';;
```

De même que l'on a écrit X au lieu de X^1 et X^n au lieu de $1X^n$, on pourrait écrire $-aX^n$ au lieu de $+-aX^n$ et proposer d'autres améliorations. Ceci relève de ce qui est généralement nommé du terme anglo-saxon *pretty-printing* mais dépasse notre but ici.

4. Écrivez une fonction `somme`: `polynôme * polynôme` → `polynôme` qui renvoie le polynôme somme des deux polynômes paramètres. On vérifiera (informellement) que l'invariant de reprsentation est bien conservé.
On peut assez facilement atteindre une complexité linéaire en s'inspirant de la fusion de deux listes triées (en effet, les polynômes sont des listes de monômes ordonnés selon les degrés décroissants).

   Solution : L'énoncé donne la solution. La liaison `as` dans les motifs permet de nommer le couple de tête. Elle permet donc de récupérer les premiers monômes `m1` et `m2` de chaque polynôme.

```
(* interface somme
type : polynôme * polynôme -> polynôme
arg p
```

```
pre : p1, p2 polynômes bien formés
post retourne la somme de p1 et p2
*)
let rec somme (p1, p2) = match p1,p2 with
| [],_ -> p2
| _,[] -> p1
| ((c1,d1) as m1)::r1, ((c2,d2) as m2)::r2 ->
    if d1 > d2 then
      m1::somme (r1, p2)
    else if d1 < d2 then
      m2::somme (p1, r2)
    else (* d1 = d2 *)
      let c = c1 + c2 in
      if c = 0 then
        somme (r1, r2)
      else
        (c,d1)::somme (r1, r2)
```

5. Écrivez une fonction `deriver`: `polynôme` → `polynôme` qui renvoie le polynôme dérivé du polynôme passé en argument. On vérifiera (informellement) que l'invariant de représentation est bien conservé.

   D'abord dériver les monômes.

```
let deriver_monome (c,d) =
  if d = 0 then (0,0)
  else (c*d , d-1);;
```

   Puis on itère la dérivation des monômes. Il faut juste faire attention à ne pas laisser trainer les monômes de coefficent nul.

```
(* interface deriver
type polynôme -> polynôme
arg p
pre p polynôme bien formé
post construit le polynôme bien formé dérivé de p
*)
let rec deriver p = match p with
  | [] -> []
  | m::rem ->
     let m' = deriver_monome m in
     if fst m' = 0 then
       deriver rem
     else
       m'::deriver rem
;;
```

6. Écrivez une fonction `produit`: `polynôme * polynôme` → `polynôme` qui renvoie le produit des polynômes passés en arguments. On vérifiera (informellement) que l'invariant de reprsentation est bien conservé.

   Solution :

   On commence par écrire la fonction qui fait le produit de deux monômes.

```
let monome_monome ((c1,d1),(c2,d2)) = (c1*c2 , d1+d2)
```

On remarque que le produit de deux monômes non nuls est encore un monôme non nul. On peut commencer par écrire une fonction qui permet d'obtenir le polynôme produit d'un monôme non nul et d'un polynôme.

```
let rec monome_polynôme (m,p) = match p with
 [] -> []
|m'::p' ->  (monome_monome (m,m'))::(monome_polynôme (m,p'));;
```

Puis on réalise le produit de deux polynômes en utilisant la fonction `somme`. En effet si l'invariant est conservé naturellement par la fonction précédente, il faut faire attention de ne pas faire apparaître de monômes de même degré en faisant le produit

```
let rec produit (p,p') = match p with
 [] -> []
|m::r ->  somme ((monome_polynôme (m,p'), produit (r,p'));;
```

7. Écrivez une fonction `evaluer`: `polynôme * int` → `int` qui calcule la valeur d'un polynôme en un point. Vous chercherez une solution efficace en vous inspirant de la méthode de Horner.

Solution : On reprend la fonction `puissance` (méthode dichotomique) déjà écrite.

let rec puissance(x,n) = if n=0 then 1 else let p = puissance(x,n/2) in if n mod 2 = 0 then p*p else x*p*p;;

let rec evaluer(p,x) = match p with [] -¿ 0 — (c,d)::r -¿ c*puissance(x,d) + evaluer(r,x);;

La solution la plus simple consiste ensuite à calculer la somme de la valeur des monômes, sans économiser les calculs.

Mais on peut faire un peu mieux (moins de multiplications) en s'inspirant de la méthode de Horner qui s'appuie sur la remarque suivante :

$$a_n x^n + \ldots + a_1 x + a_0 = (a_n x^{n-1} + \ldots + a_1) * x + a_0$$

Ainsi la valeur de $a_n x^n + ... + a_1 x + a_0$ par rapport à X est égale à la valeur de $(a_n x^{n-1} + ... + a_1$ par rapport à X) * X + a_0.
Par exemple, $X^4+3X^3-6X^2+2X+1 = (((1X+3)X-6)X+2)X+1 = ((((0X+1)X+3)X-6)X+2)X+1$. Dans cet exemple, tous les degrés entre 4 et 0 sont présents. Si certains degrés manquent, on peut faire comme si les coefficients des monômes manquants étaient nuls. Comme dans l'exemple suivant : $X^4 - 6X^2 + 1 = (((1X + 0)X - 6)X + 0)X + 1$.

Ainsi on peut calculer la valeur par additions et multiplications successives. On va appeler `r` la somme accumulée et `d` le nombre de fois où il faut multiplier par x.

```
let rec horner (d, r, x, p) = match p with
| [] ->
   if d = 0 then r
   else horner (d-1, (r*x), x, [])
| (c,deg)::r' ->
   if deg = d then
     horner (d, (r + c), x, r')
   else
     horner (d-1, (r*x), x, p) ;;

let evaluer (p, x) = match p with
| [] -> 0
| (c,d)::t -> horner (d,c,x,t) ;;
```

2 Polynômes à deux indéterminées

On considére maintenant un polynôme à deux indéterminées X et Y comme un polynôme à une indéterminée X dont les coefficients sont des polynômes à une indéterminée Y.

On adopte les mêmes conventions que ci-dessus.

1. Quel est le type de la représentation d'un tel polynôme ?
   Solution : ((int*int)list, int) list

2. Comment représentez-vous le polynôme $X^4+2X^2Y+5X^2+5Y^2$? Il est considéré comme le polynôme suivant : $X^4 + X^2(2Y + 5) + 5Y^2$.
   Solution

   ```
   [([(1,0)] , 4); ([(2,1);(5,0)], 2) ; ([(5,2)], 0)]
   ```

3. Écrire une fonction `sommme2` qui réalise la somme de deux polynômes en X et Y (supposés bien formés). Le résultat sera encore un polynôme bien formé.
   Solution : c'est le même algorithme que précédemment mais la somme des coefficients se fait avec la fonction somme définie dans la première partie.

   ```
   let rec somme2ind  (p, q) = match (p,q) with
   | ([], _) -> q
   | (_, []) -> p
   | (c,d)::p1, (c',d')::q1 ->
       if d > d' then (c,d)::(somme2ind (p1, q))
       else
         if d < d'
         then (c',d')::(somme2ind (p, q1))
         else let s = somme (c,c') in
              if s = [] then somme2ind (p1, q1)  else (s,d)::(somme2ind (p1,q1));;

   val somme2ind :
     ((int * 'a) list * 'b) list * ((int * 'a) list * 'b) list ->
     ((int * 'a) list * 'b) list = <fun>

   let monpol2 = [([(1,0)] , 4); ([(2,1);(5,0)], 2) ; ([(5,2)], 0)];;

   let monpol2' = [([(-1,0)] , 4); ([(2,1)], 3) ; ([(5,1)], 0)];;

   somme2ind (monpol2, monpol2);;
   - : ((int * int) list * int) list =
   [([(2, 0)], 4); ([(4, 1); (10, 0)], 2); ([(10, 2)], 0)]

   somme2ind (monpol2, monpol2');;
   - : ((int * int) list * int) list =
   [([(2, 1)], 3); ([(2, 1); (5, 0)], 2); ([(5, 2); (5, 1)], 0)]
   ```

4. Écrire une fonction qui permet de transformer un polynôme à deux indéterminées en une chaîne de caractères.

   Solution :
   La fonction demandée prendra cette fois en argument le nom des 2 indéterminées. L'algorithme suit

celui mis en oeuvre à la question précédente. Il convient d'appeler la fonction `string_of_polynôme` pour transformer les coefficients. Le coefficient 1 est ici le monome de coefficient 1 et de degré 1 soit [(1,1)].

```
(*interface string_of_monôme2
type : (int*int list)*int list * string * string -> string
arg m, v (nom de l'indéterminée dans les coefficients),
       w  (nom de l'indéterminée du polynôme)
pre : monome bien formé
post transforme en une chaine de caracteres un monome à Deux indéterminées
*)
let string_of_monôme2 (m,v,w) = match m with
| (c,  0) -> string_of_polynôme (c,v)
| ([(1,0)], 1) -> w
| (c, 1) -> "("^(string_of_polynôme (c,v))^")" ^ w
| ([(1,0)],  d) -> w ^ "^" ^ (string_of_int d)
| (c, d) -> "("^(string_of_polynôme (c, v)) ^")" ^ w ^ "^" ^ (string_of_int d);;

(*interface string_of_polynôme2
type : (int*int list)*int list * string * string -> string
arg m, v (nom de l'indéterminée du polynôme)
pre : polynôme bien formé
post transforme en une chaine de caracteres un
polynôme à un indéterminée nommée v
*)
let rec string_of_polynôme2 (p, v, w) = match p with
| [] -> ""
| m::p' ->
    let sm = string_of_monôme2 (m, v, w)
    and sp' = string_of_polynôme2 (p', v, w) in
    if sm = "" then sp' else
      if sp' = "" then sm else sm ^ "+" ^ sp';;

string_of_polynôme2 ([([(1,0)] , 4); ([(2,1);(5,0)], 2) ; ([(5,2)], 0)], "Y", "X");;
- : string = "X^4+(2Y+5)X^2+5Y^2"
```