Programmation fonctionnelle et logique

TP1

Fonctions classiques sur les ensembles triés

On considère des ensembles triés représentés par des listes. Écrire des procédures réalisant les fonctions suivantes :

1. Intersection ;

```
let rec intersection l1 l2 =
  match (l1,l2) with
    t1::r1,t2::r2 -> if t1=t2
                      then t1::(intersection r1 r2)
                      else if t1<t2
                             then (intersection r1 l2)
                             else (intersection l1 r2)
  | _ -> [];;
val intersection : 'a list -> 'a list -> 'a list = <fun>
```

2. Union ;

```
let rec union l1 l2 =
  match (l1,l2) with
    t1::r1,t2::r2 -> if t1=t2
                      then t1::(union r1 r2)
                      else if t1<t2
                             then t1::(union r1 l2)
                             else t2::(union l1 r2)
  | [],l2 -> l2
  | l1,[] -> l1;;
val union : 'a list -> 'a list -> 'a list = <fun>
```

3. Différence ;

```
let rec difference l1 l2 =
  match (l1,l2) with
    t1::r1,t2::r2 -> if t1=t2
                      then (difference r1 r2)
                      else if t1<t2
                             then t1::(difference r1 l2)
                             else (difference l1 r2)
  | l1,[] -> l1
  |   _   -> [];;
val difference : 'a list -> 'a list -> 'a list = <fun>
```

4. Différence symétrique ;

```
let difference_symetrique l1 l2 =
  difference (union l1 l2) (intersection l1 l2);;
val difference_symetrique : 'a list -> 'a list -> 'a list = <fun>
```

5. Produit cartésien (résultat donné dans l'ordre lexicographique).

```
let rec produit_cartesien l1 l2 =
  match (l1,l2) with
      [t1],t2::r2 -> (t1,t2) :: produit_cartesien [t1] r2
    | t1::r1,t2::r2 -> concatene (produit_cartesien [t1] l2)
                                (produit_cartesien  r1  l2)
    |    _    -> [];;
val produit_cartesien : 'a list -> 'b list -> ('a * 'b) list = <fun>
```

Le tri fusion

Le tri fusion est un algorithme basé sur la paradigme « Diviser Pour Régner ». En effet, pour trier une liste d'éléments, on coupe la liste en deux listes de même longueur qu'on trie par tri fusion. On fusionne ensuite les deux listes ainsi triées en une seule liste elle aussi triée. Le tri d'une liste vide ou d'une liste à un seul élément étant évident.

Écrire le tri fusion au moyen d'une fonction **split** qui décompose une liste en deux sous-listes de longueur égales à une unité près en sans calculer de longueur de liste.

```
let rec split = function
   t1::t2::r -> let x,y = split r in t1::x,t2::y
 |    [t1]   -> [t1],[]
 |     []    -> [],[];;
val split : 'a list -> 'a list * 'a list = <fun>
```

```
let rec tri_fusion = function
    [] -> []
 | [x] -> [x]
 |  l  -> let x,y = split l in union (tri_fusion x) (tri_fusion y);;
val tri_fusion : 'a list -> 'a list = <fun>
```