TD de programmation fonctionnelle et logique

Corrigé du TP 1 : listes et récursivité

Fonctions classiques sur les ensembles triés

On considère des ensembles triés représentés par des listes. Exemple

```
# let l1 = [1;3;5;7;9];;
val l1 : int list = [1; 3; 5; 7; 9]

# let l2 = [2;3;6;8;9];;
val l2 : int list = [2; 3; 6; 8; 9]
```

Écrire des fonctions réalisant :

1. L'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>
```

Exemple :

```
# intersection l1 l2;;
- : int list = [3; 9]
```

2. L'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>
```

Exemple :

```
# union l1 l2;;
- : int list = [1; 2; 3; 5; 6; 7; 8; 9]
```

3. La différence ;

```
# let rec différence l1 l2 =
    match (l1,l2) with
      t1::r1,t2::r2 -> if t1=t2
                       then (différence r1 r2)
```

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

Exemple :

```
#différence l1 l2;;
- : int list = [1; 5; 7]
```

4. La différence symétrique ;

```
# let différence_symétrique l1 l2 =
    différence (union l1 l2) (intersection l1 l2);;
val différence_symétrique : 'a list -> 'a list -> 'a list = <fun>
```

Exemple :

```
#différence_symétrique l1 l2;;
- : int list = [1; 2; 5; 6; 7; 8]
```

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

```
# let rec concatene l1 l2 =
    match l1 with
      []   -> l2
    | t::r -> t :: concatene r l2 ;;
val concatene : 'a list -> 'a list -> 'a list = <fun>

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

Exemple :

```
#produit_cartésien l1 l2;;
- : (int * int) list =
[(1, 2); (1, 3); (1, 6); (1, 8); (1, 9); (3, 2); (3, 3); (3, 6); (3, 8);
(3, 9); (5, 2); (5, 3); (5, 6); (5, 8); (5, 9); (7, 2); (7, 3); (7, 6);
(7, 8); (7, 9); (9, 2); (9, 3); (9, 6); (9, 8); (9, 9)]
```