Programmation fonctionnelle et logique
TD3 : récursivité et polymorphisme

Expressions à évaluer

Les expressions suivantes sont évaluées dans l'ordre. Quelles sont leurs valeurs respectives ?

```
let x = 1;;
let y = 4;;
let x = 3 and y = x + 1 in
  let x = x + y and y = x - y
    in (x,y);;
- : int * int = 5, 1
let x = 3
  in let  y = x + 1
    in let x = x + y
      in let y = x - y
        in (x,y);;
- : int * int = 7, 3
```

Calcul de (cos(nx), sin(nx))

Écrire une fonction qui prend en entrée un entier `n` et une paire de valeurs réelles qui sont en fait les valeurs du cosinus et du sinus d'un certain angle `x`, et qui renvoie la paire `(cos(nx), sin(nx))`. Autrement dit, le deuxième argument de la fonction est une paire `(a,b)` telle que `a = cos x` et `b = sin x)`. Le schéma de calcul doit bien évidemment être récursif.

On pourra se servir des formules de trigonométrie suivantes :

```
cos(nx) = cos((n-1)x) cos(x) - sin((n-1)x) sin(x)
sin(nx) = sin((n-1)x) cos(x) - cos((n-1)x) sin(x)
```

```
let trigo (a, b) n =
  let rec trigo_aux = function
      0 -> (1.,0.)
    | 1 -> (a, b)
    | m -> let (c,d) = trigo_aux (m-1)
             in (a *. c -. b *. d, a *. d +. b *. c)
  in trigo_aux n;;
val trigo : float * float -> int -> float * float = <fun>
```

Encore des fonctions opérant sur des tableaux

1. Écrire une fonction récursive qui calcul la somme des éléments d'un tableau entier de taille supérieure ou égale à un.

   ```
   let somme tab =
     let taille = (Array.length tab)
     in let rec somme_aux rang =
   ```

```
    if (rang = taille -1)
      then tab.(rang)
      else tab.(rang) + (somme_aux (rang + 1))
    in if taille = 1
          then tab.(0)
          else tab.(0) + (somme_aux 1);;
val somme : int array -> int = <fun>
```

2. Écrire une fonction récursive qui calcul la somme des éléments d'un tableau de taille supérieure ou égale à un, quel que soit son type.

```
let somme add tab =
  let taille = (Array.length tab)
  in let rec somme_aux rang =
    if (rang = taille -1)
      then tab.(rang)
      else (add tab.(rang) (somme_aux (rang + 1)))
    in if taille = 1
          then tab.(0)
          else (add tab.(0) (somme_aux 1));;
val somme : ('a -> 'a -> 'a) -> 'a array -> 'a = <fun>
```

3. Typer la fonction précédente.
4. Écrire une fonction récursive qui applique une opération binaire (typiquement `+`,`*` ou `max`) aux éléments d'un tableau de taille supérieure ou égale à un, quel que soit son type.
5. Même question mais en ne supposant plus que le tableau est forcément de taille non nulle.

```
let application op neutre tab =
  let taille = (Array.length tab)
  in let rec application_aux rang =
    if (rang = taille)
      then neutre
      else (op tab.(rang) (application_aux (rang + 1)))
    in application_aux 0;;
val application : ('a -> 'b -> 'b) -> 'b -> 'a array -> 'b = <fun>
```

6. Typer la fonction précédente.