TD : premiers pas en Caml

Exercice 1. Discuter le code suivant :

newton.ml

```
let rec until predicat changer x =
  if predicat(x) then x
  else
    until predicat changer (changer(x)) ;;

let deriv f x dx = (f(x +. dx) -. f(x)) /. dx ;;
let abs x = if x > 0.0 then x else -. x ;;

let newton f epsilon =
  let
    ok y = abs(f y) < epsilon
  and
    ameliorer y = y -. (f(y) /. (deriv f y epsilon))
  in
    until ok ameliorer
;;

let racine_carre x epsilon =
  newton (function y -> y *. y -.x) epsilon x ;;

let racine_cube x epsilon =
  newton (function y -> y *. y *. y -.x) epsilon x ;;
```

Types élémentaires

Exercice 2. Donner des exemples de fonctions ayant les types suivants :

1. `(int -> int) -> int`
2. `int -> (int -> int)`
3. `(int -> int) -> (int -> int)`

Exercice 3. Donner 3 des exemple de prédicats.

Exercice 4. Proposer un type adapté aux nombres complexes. Définir les opérations élémentaires sur les complexes : la somme, le zéro, la multiplication, l'unité, la division, et l'extraction d'une racine. Définir une fonction, qui étant donné une triple de nombres complexes a, b, c retourne les deux solutions complexes de l'équation $ax^2 + bx + c = 0$.

Fonctions

Exercice 5. Les noms des paramètres formels dans une définition ne sont pas importants. Peut on renommer la variable `x` en `y` dans l' expression
`function x -> (function y -> y + x)`
?

Curryfication

Exercice 6. Soit $A \times B$ le produit cartésien des ensembles A et B, soit $A \Rightarrow B$ l'ensemble des fonctions de A à valeurs vers B. Démontrer le fait suivant : *il existe une bijection entre les fonctions de $A \times B$ vers C et les fonctions de A vers $B \Rightarrow C$.*

Exercice 7. Expliquer les fonctions suivantes et en donner les types :

```
#let curry f = function x -> (function y -> f(x,y));;
#let plus (x, y) = x + y;;
#curry plus;;
#let uncurry f = function (x,y) -> f x y ;;
```

Calculer les types des expressions suivantes :

```
#let compose f g = function x -> f(g(y));;
#uncurry compose;;
#compose curry uncurry ;;
#compose uncurry curry ;;
```

Récursivité

Exercice 8. Traduire la fonction C suivante en CAML (fonctionnel) :

```
int sommeborne(int n, int f(int))
{
  int x = 0, i=0;

  while ( i <= n)
    x += f(i++);

  return x;
}
```

Exercice 9. Définir une fonction CAML `iter` dont le type est

```
iter : int -> ('a -> 'a) -> 'a -> 'a
```

et dont la sémantique est $iter\ n\ f\ x = f^n(x)$. Généraliser cette fonction à une fonction `fold` dont le type est

```
fold : int -> (int -> 'a -> 'a) -> 'a -> 'a
```

et dont la sémantique est $fold\ n\ f\ x = f(n, \ldots f(1, f(0, x))$. Utiliser la fonction `fold` pour résoudre l'exercice précédent.

Exercice 10. Le schéma de récursion primitive explique qu'on peux définir une fonction f à partir des fonctions g et h de la façon suivante :

$$\begin{aligned} f(0, x) &= g(x) \\ f(n+1, x) &= h(n, x, f(n, x))\,. \end{aligned}$$

A quel ensemble appartient ici x ? Donner les "types" de g et h.
Écrire une fonction CAML `prim_rec` qui prend g et h en paramètre, et retourne f définie comme ci-dessus.