TP n° 2 : Fonctions et récursivité

1 Polymorphisme

EXERCICE 1

Écrire les fonctions `min : 'a -> 'a -> 'a` et `max : 'a -> 'a -> 'a` qui renvoient respectivement le plus petit et le plus grand de leurs arguments. Quel est le type des fonctions `max 0` et `min 0.0` ? Que font-elles ?

Que pensez-vous de :

OCAML
```
let min_bis x y = -(max (-x) (-y));;
```

☞ Remarque 1

Les fonctions `min` et `max` sont prédéfinies en CAML *et dorénavant vous avez le droit de les utiliser*[1].

Voici plusieurs manières équivalentes de définir la fonction `fst : 'a * 'b -> 'a` (pour **f***ir***st**) qui renvoie le premier élément d'un couple :

OCAML
```
let fst = fun (a, b) -> a;;
let fst (a, b) = a;;
let fst couple =
  let (a, b) = couple in
  a
;;
```

EXERCICE 2

Définir de même, *i.e.* de ces trois façons équivalentes, la fonction `snd` (pour **s***eco***nd**) qui renvoie le deuxième élément du couple. Commencez par prévoir son type !

☞ Remarque 2

Les fonctions `fst` et `snd` sont prédéfinies en CAML *et vous pouvez les utiliser.*

EXERCICE 3

Implémenter, après avoir trouvé leur type, les fonctions suivantes :

1. La fonction `duplicate` qui sur un argument `a` s'évalue[2] en le couple `(a, a)`.
2. La fonction `swap` qui sur un couple `(a, b)` s'évalue en le couple `(b, a)`.
3. Une fonction `concat` qui sur deux couples `(a, b)` et `(c, d)` s'évalue en le quadruplet `(a, b, c, d)`.

1. Sauf bien sûr s'il est explicite que vous devez les réimplémenter.
2. On devrait plutôt dire « s'évalue » en CAML et non « renvoie » comme en PYTHON, mais on commettra bientôt l'abus.

2 Fonctions d'ordre supérieur

EXERCICE 4 ✍

Prévoir le type et le comportement, des fonctions CAML suivantes. En donner une description mathématique. Vérifier à l'aide de CAML.

OCAML
```
let fonction1 = fun f -> f 0;;
let fonction2 f = f 0;;
let fonction3 f g = fun x -> (f x) + (g x);;
let fonction4 f g x = (f x) + (f x);;
```

EXERCICE 5

Proposer une implémentation en CAML des fonctions :

- qui à $f : \mathbb{R} \to \mathbb{R}$ associe $\frac{f(0)+f(1)}{2}$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ et $x \in \mathbb{R}$ associe $(f(x))^2$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe f^2 ;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe $f \circ f$;
- qui à $f : \mathbb{R} \to \mathbb{R}$ associe la fonction $x \mapsto f(x+1)$ de $\mathbb{R} \to \mathbb{R}$.

EXERCICE 6

En définissant $f : x \mapsto \sin(\ln(x))$ comme une fonction locale, implémenter la fonction :

$$g : x \mapsto \frac{f(x)}{1 + (f(x+1))^2}$$

3 Récursivité

3.1 Tracer une fonction

La primitive `trace` permet de « tracer » le déroulement d'une fonction et est très utile pour visualiser les appels récursifs.

OCAML
```
let rec factorielle n =
  if n = 0 then
    1
  else
    n * factorielle (n - 1)
;;

#trace factorielle;;

factorielle 10;;

#untrace factorielle;;

factorielle 50;;
```

On pourra utiliser la fonction `trace` dans les exercices suivants afin de bien comprendre le comportement des fonctions récursives.

3.2 Quelques fonctions récursives simples

EXERCICE 7

Écrire une fonction `mult : int -> int -> int` telle que `mult x y` calcule le produit de x par y sans effectuer de multiplications (*i.e.* sans utiliser le symbole « $\star$ »).

EXERCICE 8

Écrire une fonction `somme_termes` qui, à une fonction $f : \mathbb{N} \to \mathbb{N}$ et à un entier n, associe $\sum_{k=0}^{n} f(k)$. Calculer la somme des cubes des entiers naturels inférieurs ou égaux à 100.

EXERCICE 9

Pour certains arguments, une fonction peut ne pas être définie. On le signale en CAML à l'aide de `failwith "message"`, ce qui va déclencher une exception (ici `Failure "message"`) et interrompre le programme[3].

1. Écrire deux fonctions récursives `quotient` et `reste` qui, à partir de deux entiers positifs a et b, renvoient respectivement le quotient et le reste de la division de a par b, sans utiliser bien sûr l'opérateur de division ou l'opérateur `mod`.
2. Écrire une fonction `divmod` qui calcule les deux à la fois et renvoie le couple $(quotient, reste)$.

Bonus : Écrire une version de `divmod` dans laquelle a et b peuvent être négatifs.

3. L'utilisation des exceptions en dehors de ce cas simple est hors programme, mais très utile.