TP n° 2 : Fonctions et récursivité

Dans ce TP, nous vous proposons de mettre en application l'idée qu'en CAML une fonction est un objet comme un autre et de profiter de l'occasion pour étudier quelques fonctionnelles. La deuxième partie traite ensuite de fonctions récursives. Avant de commencer ce sujet il est conseillé d'avoir terminé le TP d'introduction à CAML.

1 Fonctions d'ordre supérieur

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 :

```
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].

EXERCICE 2 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 3

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

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

EXERCICE 4

Imaginer et écrire des fonctions simples mais non triviales ayant les types suivants :

- `int -> (float -> float)`
- `int -> float -> float`
- `(int -> float) -> float`
- `(int -> float) -> (int -> float)`
- `int -> (float -> int) -> float)`
- `int -> (float -> int -> float)`
- `(int -> float -> int) -> float`
- `int -> float -> int -> float`

On donnera la définition mathématique avant de se lancer dans l'implémentation.

EXERCICE 5 *Différences finies*

L'opérateur des différences finies Δ associe à toute suite $(u_n)_{n\in\mathbb{N}}$ la suite $(u_{n+1} - u_n)_{n\in\mathbb{N}}$. Écrire une fonction `delta` qui réalise cette transformation. On commencera par en donner le type.

1. Sauf bien sûr s'il est explicite que vous devez les réimplémenter.

2 Récursivité

2.1 Tracer une fonction

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

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

trace "factorielle";;

factorielle 5;;

untrace "factorielle";;

factorielle 50;;
```

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

EXERCICE 6

Réécrire la fonction factorielle en utilisant une définition par cas.

2.2 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[2].

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

2.3 Récursivité croisée

On rappelle que le mot-clé `and` permet de définir des fonctions mutuellement récursives :

```
let rec f arg [args] =
  ... g ...
and g arg [args] =
  ... f ...
```

EXERCICE 10 *Moyenne arithmético-géométrique*

Soit $a, b \in \mathbb{R}_+$, on définit deux suites positives $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$, de premiers termes $u_0 = a$ et $v_0 = b$ et satisfaisant les relations de récurrence :

$$u_{n+1} = \sqrt{u_n v_n} \qquad v_{n+1} = \frac{u_n + v_n}{2}$$

1. Montrer que les suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ sont adjacentes.
2. Pour $a = 1$ et $b = 2$, calculer u_{10} et v_{10}. A-t-on `(u 10) = (v 10)` ? Expliquer.
3. Écrire une fonction `suites` telle que `suites a b` renvoie le couple des suites $((u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}})$. Quel est son type ?
4. En utilisant la fonction précédente, écrire une fonction `termes` telle que `termes a b n` renvoie le couple des n-ième termes (u_n, v_n).
5. En utilisant la fonction précédente, calculer u_{10} et v_{10} pour $(a, b) \in \{(1, 1000), (2, 487), (500, 501)\}$.

EXERCICE 11

Que font les fonctions f et g ?

1.
```
let rec f n = (n = 0) || g (n - 1)
and g n = (n <> 0) && f (n - 1);;
```

2.
```
let rec f n = if n = 0 then 1 else 3 + g (n - 1)
and g n = if n = 0 then 0 else 1 + f (n - 1);;
```

3. (à implémenter bien entendu)

$$f : (m, n) \to \begin{cases} n & \text{si } m = 0 \\ g(m-1, n+1) & \text{sinon} \end{cases} \qquad g : (m, n) \to \begin{cases} n & \text{si } m = 0 \\ f(m-1, n)+1 & \text{sinon} \end{cases}$$

3 Bonus

EXERCICE 12

Écrire une fonction `nombre_ecriture` qui, à deux entiers m et n, renvoie le nombre d'écritures (sans tenir compte de l'ordre) de n comme somme de m entiers strictement positifs [3].

3. Indication : On pourra chercher une relation de récurrence en séparant deux cas, selon que l'entier 1 est utilisé ou non dans la décomposition.