Interrogation écrite 3 - Corrigé commenté

INF 201 — IMA1— 30/03/2022 — 30 minutes

Exercice 1. (/2) On rappelle que la factorielle de $n \in \mathbb{N}$ est définie par:

$$n! = \prod_{i=1}^{n} i = n \times (n-1) \times \cdots \times 1 \text{ et } 0! = 1$$

Donner la spécification de `fact(n)`:

Quelques erreurs sur le profil, `float` → `int`.

Donner une implémentation **récursive** en OCaml de `fact(n)`:

Vu en TP.

Exercice 2. (/8) On rappelle qu'en arithmétique le PGCD de deux nombres est le plus grand entier qui les divise simultanément. Par exemple $\text{PGCD}(15, 12) = 3$. On a alors $\forall k \in \mathbb{N}, \text{PGCD}(k, 0) = k$ et on rappelle aussi la propriété suivante pour $a > b \geq 0$:

$$\text{PGCD}(a, b) = \text{PGCD}(b, a - b)$$

Calculer à la main:

PGCD(21,14)= 7

PGCD(121,132)= 11

Donner une implémentation de `pgcd` en OCaml, votre fonction devra être **récursive** et devra traiter les cas $a < b$ ou $b < a$.

Exemples:

```
# pgcd 12 15;;
- : int = 3
```

```
# pgcd 15 12;;
- : int = 3
```

```
# pgcd 1024 577;;
- : int = 1
```

```
let rec pgcd a b = match b with
  | 0 -> a
  | b when a < b -> pgcd b a
  | _ -> pgcd b (a-b)
;;
```

Peu d'implémentation qui fonctionnent vraiment. Il fallait bien évidemment utiliser la formule donnée au dessus et non pas tenter de réinventer quelque chose avec le modulo …

Montrer que votre fonction termine en spécifiant une **mesure**:a

On utilise par exemple la mesure suivante

$$m(a, b) = \max(a, b)$$

Comme $a \in \mathbb{N}$ et $b \in \mathbb{N}$ on a $m(a, b) \in \mathbb{N}$. De plus, si $a > b$ alors $\max(a, b) = a > a - b$, donc $\max(a, b) > \max(a - b, b)$. En échangeant a et b on peut faire le même raisonnement. On a donc une mesure qui est une suite décroissante à valeurs dans $\mathbb{N}$, elle est donc convergente et par conséquent le programme termine.

Exercice 3. (/10) On souhaite écrire des fonctions sur le type `list` de OCaml. Évidemment on n'utilisera **aucune** fonction du module `List` dans cet exercice. Toutes les fonctions devront être **récursives**. Pour chaque fonction on donnera le profil et un exemple significatif. Il n'est pas nécessaire de spécifier les types dans la fonction OCaml.

Dans cet exercice beaucoup d'oubli dans les profils: il fallait mettre `'a list` et non `list` comme je l'ai vu bien souvent. Soyez aussi cohérent, ne pas mélanger notations mathématiques et informatique comme par exemple: `'a list` → `int` et non pas `'a list` → $\mathbb{N}$.

Question 1. Écrire une fonction `longueur` qui renvoie la longueur d'une liste. On pose `longueur [] = 0`.

PROFIL: `'a list → int` EXEMPLE: `longueur []=0  longueur [1,2] = 2`

```
let rec longueur l = match l with
  |[] -> 0
  |t::q -> 1 + longueur q
;;
```

Question assez bien réussie.

Question 2. Écrire une fonction `concatener` qui concatène deux listes ensemble. **Interdiction** d'utiliser la primitive `@` !

PROFIL:`'a list -> 'a list -> 'a list` EXEMPLE: `concatener [1;2] [3;4] = [1;2;3;4]`

```
let rec concatener l1 l2 = match l1,l2 with
  |[],[] -> []
  |_,[] -> l1
  |[],_ -> l2
  |t1::q1,_ -> t1::(concatener q1 l2)
;;
```

Beaucoup de confusion et d'implémentations qui ne restituent pas l'ordre des éléments.

Question 3. Écrire une fonction `retourner` qui renverse l'ordre des éléments d'une liste. On pourra faire appelle à `concatener`.

PROFIL:`'a list -> 'a list` EXEMPLE: `retourner [1;2;3] = [3;2;1]`

```
let rec retourner l = match l with
  |[] -> []
  |t::q -> concatener (retourner q) [t]
;;
```

Attention impossible d'écrire `t::q -> q::t` ça ne marche pas comme ça !

Question 4. Écrire une fonction `comparer` qui teste l'égalité de deux listes, elle renvoie `true` si tous les éléments sont égaux et `false` si au moins un élément est différent. *Attention aux cas ou les listes sont de longueurs différentes ...*

PROFIL: `'a list -> 'a list -> bool` EXEMPLE: `comparer [1;2;3] [1;2] = false comparer ['a';'b'] ['a';'b'] = true`

```
let rec comparer l1 l2 = match l1,l2 with
  |[],[] -> true
  |_,[] -> false
  |[],_ -> false
  |t1::q1,t2::q2 -> t1=t2 && comparer q1 q2
;;
```

Inutile de faire appelle à la fonction longueur. Il suffit de traiter les cas avec des listes vides.

Question 5. Écrire une fonction `appliquer` qui applique une fonction sur tous les éléments d'une liste. Pour l'exemple on pourra prendre une liste d'entiers et la fonction carré.

PROFIL:`('a -> 'b) -> 'a list -> 'b list` EXEMPLE: `appliquer (fun x -> x*x) [1;2;3] = [1; 4; 9]`

```
let rec appliquer f l = match l with
  |[] -> []
  |t::q -> (f t)::(appliquer f q)
;;
```

Le profil a souvent été raté, la fonction était mieux réussie, il s'agit d'une introduction à l'ordre supérieur.

Question 6. Montrer que la fonction `longueur` termine en spécifiant une mesure.

On pose comme mesure le cardinal de la liste qui est bien à valeur dans $\mathbb{N}$:

$$m(l) = |l|$$

Puis on applique à l'équation récursive:

$$m(l) = m(t :: q) = |t :: q| = 1 + |q| > |q| = m(q)$$

On a donc une mesure qui est une suite décroissante à valeurs dans $\mathbb{N}$, elle est donc convergente et par conséquent le programme termine.

Trop d'imprécisions dans cette démonstration que nous avons pourtant fait plusieurs fois en TD !