Interrogation écrite 2

INF 201 — IMA3&4 — 27/02/2026 — 25 minutes

Exercice 1. (/10) On souhaite avoir un type qui représente soit un scalaire appartenant à $\mathbb{R}$, soit un vecteur dans le plan $\mathbb{R}^2$. Mathématiquement on écrira:

$$\text{euclidien} = \{\text{Scalaire}(s), s \in \mathbb{R}\} \cup \{\text{Vec2}(v), v \in \mathbb{R} \times \mathbb{R}\}$$

Question 1. (/2) En déduire une implémentation en OCaml du type `euclidien`:

```
type euclidien = Scalaire of float | Vec2 of float*float;;
```

Question 2. (/5) On souhaite réaliser une fonction `produit_scalaire` qui effectue le produit scalaire de deux types `euclidien`. Le produit scalaire de deux réels sera juste leur produit tandis que le produit scalaire d'un réel λ et d'un vecteur v sera $(\lambda \quad 0)\cdot v$. Donner le profil de la fonction:

```
   val produit_scalaire : euclidien -> euclidien -> float
```

compléter les exemples:

```
   produit_scalaire (Scalaire(3.)) (Vec2(1., 1.))  = 3.
   produit_scalaire (Vec2(1.,1.)) (Vec2(1., -1.)) = 0.
```

et enfin l'implémenter:

```
let produit_scalaire a b = match a,b with
  |Scalaire(x), Scalaire(y) -> x*.y
  |Scalaire(x), Vec2(y1, y2)|Vec2(y1, y2), Scalaire(x) -> x*.y1
  |Vec2(x1, y1), Vec2(x2, y2) -> x1*.x2 +. y1*.y2
;;
```

Question 3. (/3) On souhaite en déduire une fonction `norme` qui donne la longueur d'un vecteur. Pour un `Scalaire` on renverra la valeur absolue à l'aide de la fonction `abs_float` tandis que pour un `Vec2` on utilisera la fonction `sqrt: float -> float` après un appel judicieux à `produit_scalaire`. Compléter les exemples.

```
   norme (Scalaire (-2.)) = 2.
   norme (Vec2 (1., 1.)) = 1.414...
```

Puis donner une implémentation:

```
let norme x = match x with
  |Scalaire(x) -> abs_float x
  |Vec2(_) -> sqrt (produit_scalaire x x)
;;
```

Exercice 2. (/11) De façon analogue à ce qui a été vu en CM/TD on définit les entiers de Peano de la façon suivante:

$$\text{peano} = \{Z\} \cup \{S(p), p \in \text{peano}\}$$

Le type peano peut-être implémenté en OCaml de la façon suivante:

```
type peano = Z | S of peano;;
```

Question 4. (/2) Donner une implémentation non récursive d'une fonction `est_nul : peano -> bool` qui dit si un entier de Peano est nul:

```
let est_nul p = Z = p;;
```

Question 5. (/4) On aimerait avoir une fonction `int_vers_peano : int -> peano` qui donne la représentation de Peano d'un entier positif.

Compléter les exemples:

```
int_vers_peano 0 = Z
int_vers_peano 3 = S (S (S Z))
```

Donner une implémentation récursive de cette fonction:

```
let rec int_vers_peano n = match n with
  |0 -> Z
  |n -> S (int_vers_peano (n-1))
;;
```

Question 6. (/2) On suppose l'existence d'une fonction `peano_vers_int` qui transforme un entier dans la représentation de Peano en un entier

Compléter les exemples:

```
peano_vers_int Z = 0
peano_vers_int (S (S Z)) = 2
```

Que peut-on dire de la fonction `peano_vers_int` ∘ `int_vers_peano`:

C'est la fonction identité sur les entiers de peano.

On peut aussi dire que les fonctions sont bijection réciproque de l'une de l'autre.

Question 7. (/3) Finalement on veut écrire une fonction `lire: peano -> string` qui « lit » le nombre de peano dans le langage naturel:

```
lire Z = "Zero"
lire (S (S (S Z))) = "Successeur␣de␣Successeur␣de␣Successeur␣de␣Zero"
```

Donner une implémentation récursive de cette fonction:

```
let rec lire p = match p with
  |Z -> "Zero"
  |S(p')␣->␣"Successeur␣de␣"^(lire␣p')
;;
```