# Programmation Fonctionnelle - Exercices

23 novembre 2017

## 1 Types algébriques

### 1.1 Sommes de multiples

**Q1.** On rappelle (cf. TLANG) que tout entier naturel $n$ peut s'écrire comme la somme d'un multiple de 3 et d'un multiple de 4, à l'exception de 1, 2, et 5 : $\forall n \in \mathbb{N} \setminus \{1, 2, 5\}, \exists i, j \in \mathbb{N} : n = 3i + 4j$. Proposer la définition d'un type algébrique `Entier` permettant de représenter tous les entiers, sous cette forme quand c'est possible, plus explicitement sinon.

**Q2.** Écrire le type et la définition d'une fonction `valeur` qui étant donné un `Entier`, donne sa veleur sous forme d'un `Integer`. Donner également deux exemples représentatifs d'appels à cette fonction.

### 1.2 Coordonnées des points dans le plan

Un point dans le plan peut être représenté par ses coordonnées cartésiennes (abscisse, ordonnée) ou par ses coordonnées polaires (angle, distance à l'origine).

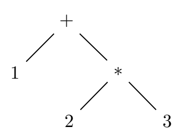
**Q1.** Définir un type algébrique `Point` qui permet ces deux représentations, exclusives l'une de l'autre.

**Q2.** Écrire le type et la définition d'une fonction `dOrigine` qui, à partir d'un `Point`, donne sa distance à l'origine.

**Q3.** Donner un exemple concret d'appel à la fonction `dOrigine`.

### 1.3 Expressions arithmétiques

**Q1.** Une expression arithmétique peut être représentée par un arbre binaire, avec des nœuds de nature variant selon l'opérateur correspondant. Donnez la définition d'un type algébrique `ExprA` permettant de représenter sous cette forme une expression arithmétique sur des entiers, et utilisant uniquement les opérateurs `+` et `*`. Un exemple d'une telle expression est `1+2*3`. L'arbre correspondant (qu'on ne demande pas de construire) est :




**Q2.** Écrire le type et la définition d'une fonction `evaluate` qui évalue une expression arithmétique donnée sous la forme de la question précédente. Par exemple, pour l'arbre représentant `1+2*3`, le résultat doit être `7`.

# 2 Récursivité

## 2.1 Algorithme d'Euclide

Si $r$ est le reste de la division euclidienne de $a$ par $b$ alors le PGCD de $a$ et $b$ est le PGCD de $b$ et $r$. Écrire le type et la définition d'une fonction récursive `euclide` qui calcule le PGCD de $a$ et $b$.

## 2.2 Sommes d'entiers et nombres parfaits

**Q1.** Écrire une fonction récursive qui calcule la somme des $n$ premiers entiers ;

**Q2.** Écrire une fonction récursive terminale qui calcule la somme des $n$ premiers entiers ;

**Q3.** Écrire une fonction récursive terminale qui teste si un nombre est parfait.

## 2.3 Suite de Syracuse

Étant donné un entier $x$, la *suite de Syracuse* de $x$ est définie par : $u_0 = x$, $u_{n+1} = u_n/2$ si $u_n$ est pair et $u_{n+1} = 3u_n + 1$ si $u_n$ est impair. Écrire le type et la définition d'une fonction `syracuse` qui, à partir de deux entiers `x` et `n`, donne le terme de rang `n` de la suite de Syracuse de `x`. Par exemple `syracuse 14 10` vaut `20`.

## 2.4 Expressions bien parenthésées

Écrire le type et la définition d'une fonction **récursive terminale** `isMatched` qui étant donnée une chaîne de caractères ne contenant que des parenthèses ouvrantes et fermantes, vérifie que l'expression représentée est bien parenthésée. P. ex., `isMatched "((())()())"` vaut `True` alors que `isMatched "())()"` vaut `False`.

## 2.5 Chemin de plus petite somme

Écrire le type et la définition d'une fonction `minPath` qui trouve le chemin de plus petite somme dans une matrice d'entiers représentée par une liste de listes, pour aller du début de la première ligne à la fin de la dernière ligne, en ne se déplaçant que d'une case vers la droite ou une case vers le bas à chaque fois.

P. ex., calculons `minPath [[1, 12, 57, 74], [32, 42, 72, 3], [1, 55, 45, 2]]`. On a la matrice :

$$\begin{bmatrix} \mathbf{1} & \mathbf{12} & 57 & 74 \\ 32 & \mathbf{42} & \mathbf{72} & \mathbf{3} \\ 1 & 55 & 45 & \mathbf{2} \end{bmatrix}$$

Le chemin de plus petite somme est donné en gras et vaut : $1 + 12 + 42 + 72 + 3 + 2 = 132$.

## 2.6 Puissances de fonctions

La puissance $n^{\text{e}}$ d'une fonction $f$, d'un ensemble quelconque dans lui-même, est définie comme la composition de $n$ fois la fonction $f$. Par exemple $\forall x, f^3(x) = f(f(f(x)))$. Écrire le type et la définition d'une fonction `puissancef` qui, à partir d'une fonction `f` d'un ensemble quelconque dans lui-même, d'un élément de cet ensemble `x` et d'un entier `n`, donne la valeur de $f^n(x)$. Par exemple, `puissancef (+1) 2 3` vaut `5`.

## 2.7 Éléments d'indices pairs dans une liste

**Q1.** Écrire le type et la définition d'une fonction `rangsPairs` qui, à partir d'une liste d'éléments quelconques `xs`, donne la sous-liste composée des éléments d'indices pairs (0 étant considéré comme pair). Par exemple, `rangPairs [1..10] = [1,3,5,7,9]` vaut `5`.

**Q2.** Même question en n'utilisant pas la récursion mais seulement `zip`, `filter` et `map`.

### 2.8 Fonctions drop, take et splitAt

**Q1.** Écrire le type et la définition d'une fonction `drop` qui étant donnés un entier `n` et une liste `xs`, donne la liste `xs` privée de ses `n` premiers éléments. Si `n` est plus grand que la longueur de `xs`, la fonction donne la liste vide. Par exemple, `drop 4 [4,5,6,7,8,9]` vaut `[8,9]`.

**Q2.** Écrire le type et la définition d'une fonction **récursive terminale** `take` qui étant donnés un entier `n` et une liste `xs`, donne la liste des `n` premiers éléments de `xs`. Si `n` est plus grand que la longueur de `xs`, la fonction donne tout `xs`. Par exemple, `take 3 [4,5,6,7,8,9]` vaut `[4,5,6]`.

**Q3. Sans utiliser de récursion explicite**, écrire le type et la définition d'une fonction `splitAt` qui étant donnés un entier `n` et une liste `xs`, donne le couple `(as,bs)` tel que la concaténation de `as` et `bs` est `xs` et la longueur de `as` est `n`.

Par exemple, `splitAt 2 [4,5,6,7,8,9]` vaut `([4,5],[6,7,8,9])`.

## 3 Utilisation de map

### 3.1 map de Matrices

**En utilisant la fonction `map` et sans utiliser de récursion explicite**, écrire le type et la définition d'une fonction `map2D` qui applique une fonction `f` à tous les éléments d'une matrice (représentée comme une liste de liste) et renvoit la matrice des résultats. Par exemple, `map2D (+1) [[1,2,3],[4,5,6],[7,8,9]]` vaut `[[2,3,4],[5,6,7],[8,9,10]]`.

### 3.2 Plus petite puissance supérieure ou égale à une valeur

Écrire le type et, **en utilisant** la fonction `map`, la définition d'une fonction `petitePuissance` qui, à partir de deux entiers `x` et `y`, donne la valeur de la plus petite puissance de `x` qui est supérieure ou égale à `y`. Par exemple, `petitePuissance 3 100` vaut 243.

### 3.3 Chiffre de César

On s'intéresse au *chiffre de César* qui permet de chiffrer un message sur l'alphabet $\{a, \ldots, z\}$ en décalant chaque lettre de 13 rangs vers $z$ (avec une rotation si besoin) : par exemple $a$ donne $n$ et $s$ donne $f$. On se donne les fonction `ord :: Char -> Int` donnant le rang d'une lettre (supposé pour l'exercice compris entre 0 et 25) et `chr :: Int -> Char` donnant la lettre correspondant à un rang.

**En utilisant de manière explicite l'opérateur de composition `(.)`, et sans utiliser de récursion explicite**, écrire le type et la définition d'une fonction `cesar` qui permet de chiffrer un message donné comme une chaîne de caractères. Par exemple, `cesar "coin"` vaut `"pbva"` et `cesar "pbva"` vaut `"coin"`.

## 4 Utilisation de filter

### 4.1 Sous-listes croissantes

**En utilisant la fonction filter**, écrire le type et la définition d'une fonction `maxSub` qui renvoit la taille de la plus grande sous-liste strictement croissante dans une liste d'entiers. Par exemple, `subMax [4,3,2,3,7,5,6]` vaut 4, car la plus grande sous-liste strictement croissante est `[2,3,5,6]`.

### 4.2 Somme des éléments pairs dans une liste

Écrire le type et, **en utilisant** la fonction `filter` et en faisant apparaître explicitement l'opérateur de composition de fonctions `(.)`, la définition d'une fonction `sommePairs` qui, à partir d'une liste d'entiers, donne la somme des éléments de la liste qui sont pairs. Par exemple, `sommePairs [1,2,3,4]` vaut `6`.

## 5 Utilisation de fold

### 5.1 Inversion d'une liste

**En utilisant la fonction** `foldl`, écrire le type et la définition d'une fonction `reverse` qui étant donnée une liste donne la liste écrite à l'envers. Par exemple, `reverse [1,2,3,4]` vaut `[4,3,2,1]`.

### 5.2 Valeur d'un polynôme en un point

On considère des polynômes à coefficients entiers donnés par la liste de ces coefficients. Par exemple la liste $[1,2,3]$ correspond au polynôme $1+2x+3x^2$. Écrire le type et, **en utilisant** soit la fonction `foldl`, soit la fonction `foldr`, la définition d'une fonction `evalPoly` qui, à partir d'une liste d'entiers représentant un polynôme et d'un réel $z$, donne la valeur du polynôme en $z$. Par exemple, `evalPoly [1,2,3] 4.0` vaut `57.0`.

### 5.3 map et fold

**Q1.** Proposer une définition de `map` en fonction de `foldl` ou expliquer pourquoi ce n'est pas possible.

**Q2.** Proposer une définition de `foldl` en fonction de `map` ou expliquer pourquoi ce n'est pas possible.

## 6 Listes en compréhension

### 6.1 Argmin

**En utilisant une liste en compréhension**, écrire le type et la définition d'une fonction `argminf` qui, pour toute fonction `f:Int->Int` strictement croissante, et tout entier `n`, donne la valeur minimale de `x` telle que `f x > n`. Par exemple, `argminf (+2) 10` vaut `9`.

### 6.2 Produit de deux matrices

On rappelle que si les coefficients de deux matrices carrées d'entiers $A$ et $B$ de taille $n$ sont respectivement $a_{ij}$ et $b_{ij}$ pour $i,j$ entre 0 et $n-1$, alors le coefficient d'indice $i,j$ du produit est donné par $\sum_{k=0}^{n} a_{ik} * b_{kj}$. Écrire le type et la définition d'une fonction `produitMat` qui, à partir de deux matrices supposées carrées d'entiers `xss` et `yss`, représentées par des listes de listes d'entiers, donne la matrice correspondant à leur produit. Par exemple, `produitMat [[1,2],[3,4]] [[5,6],[7,8]]` vaut `[[19,43],[22,50]]`.