Programmation Fonctionnelle L2 – Examen Partiel 2020

1 Types

Considérer le programme suivant.

```
1  f1 'a' = 'b'
2  f1 _ = 'c'
3  f2 [] = [['a']]
4  f3 x | head x == head (tail x) = "a"
5       | otherwise = "b"
6  f4 x y = x y y
7  f5 x = if head x == 'a' then "a" else 0
8  f6 x y = map (x y)
9  f7 x = map x x
10 f8 x = zip x x
11 f9 x y = x ++ f8 y
12 f10 = f6 head
13 f11 x y = map (x+) y
14 f12 x y = zipWith (x+) y
```

- Trouver les lignes qui donnent une erreur de compilation. Expliquer quelle est l'erreur.
- Pour les expressions `f1,f2,..,f12` pour lesquels le programme n'a pas d'erreur de compilation, donner le type inféré par le compilateur.

(4 points)

Solution :

```
1  f1 'a' = 'b'
2  f1 _ = 'c' -- f1 :: Char -> Char
3  f2 [] = [['a']] -- f2 :: [a]->[[Char]]
4  f3 x | head x == head (tail x) = "a"
5       | otherwise = "b" -- f3 :: Eq a => [a] -> [Char]
6  f4 x y = x y y -- f4 :: (a->a->b)->a->b
7  -- f5 x = if head x == 'a' then "a" else 0
8  -- Error type de "a" != type de 0
9  f6 x y = map (x y) -- f6 :: (a->b->c)->a->[b]->[c]
10 -- f7 x = map x x
11 -- Error x est soit liste soit fonction
12 f8 x = zip x x -- f8 :: [a]->[(a,a)]
13 f9 x y = x ++ f8 y -- f9 :: [(a,a)]->[a]->[(a,a)]
14 f10 = f6 head -- f10 :: [a->b]->[a]->[b]
15 f11 x y = map (x+) y :: f11 :: Num b => b->[b]->[b]
16 -- f12 x y = zipWith (x+) y
17 -- Premier argument de zipWith doit prendre deux arguments
```

Barème : $0,33$ points par fonction (pour le bon type ou pour détection d'erreur de type)

2 Types – Part 2

Pour chacun des types suivants, donner une expression de Haskell qui a ce type, c'est-à-dire, le compilateur infère le type donné pour votre expression.

```
1 String -> String -> String
2 (String -> String) -> String
3 (Char,Char) -> (Char -> Char)
4 Num a => [a] -> [a] -> [a]
5 Ord a => a -> a -> Bool
6 [a] -> [b] -> [a]
7 Bool -> Maybe Char
8 Maybe a
```

(2 points)

Solution :

```
1 f1 "a" "b" = "ab"
2 f2 g = (g "x")++"y"
3 f3 ('a','b') 'c' = 'd'
4 f4 [x] [y] = [x+y]
5 f5 x y = x>y
6 f6 [x] [y] = [x]
7 f7 t = if t then Just 'a' else Nothing
8 f8 = Nothing
```

Barème : $0,25$ points par type. Évidemment il y a d'autres solutions possibles, vous pouvez les vérifier avec ghci.

3 Dérivées

Écrire une fonction `delta` qui étant donné une liste de `Integer` retourne une autre liste de `Integer` dont chaque élément et la différence de deux éléments consécutifs de la liste donnée. Vous pouvez supposer que la liste donnée a au moins deux éléments. (2 points)

Exemples d'utilisation:

```
*Main> delta [1,5,6,2,3]
[4,1,-4,1]
-- NB [5-1, 6-5, 2-6, 3-2]
*Main> delta [1..5]
[1,1,1,1]
*Main> delta [1,4..15]
[3,3,3,3]
```

Solution :

```
1 delta :: [Integer] -> [Integer]
2 delta [x,y] = [y-x]
3 delta (x:y:xs) = (y-x):delta (y:xs)
```

4 Médian

Écrire une fonction qui étant donnée une liste retourne son élément médian, c'est-à-dire, un élément x tel que $\leq 50\%$ des éléments de la liste sont supérieurs à x et $\leq 50\%$ des éléments de la liste sont inférieurs à x. **NB:** si la liste a taille paire, il y a deux éléments médians possibles que vous pouvez retourner. Vous pouvez supposer que les éléments de la liste donnée sont distincts (pas de doublons). (2 points)

Exemples d'utilisation:

```
*Main> median [1..9]
5
*Main> median [9,8..1]
5
*Main> median [1,3,2,4,5,6,7]
4
```

Solution :

```
1 median :: Ord a => [a] -> a
2 median [x] = x
3 median [x,y] = x
4 median xs = median $ filter (\x -> x/=mmax && x/=mmin) xs
5   where
6     mmax = foldl (max) (head xs) xs
7     mmin = foldl (min) (head xs) xs
```

NB : on accepte aussi une solution qui trie la liste, mais il faut écrire un algorithme de tri.

5 Map, Fold, etc.

Quels sont les résultats d'évaluation des lignes suivantes ? (4 points)

```
filter (>5) [ x+y | x<-[1..3], y<-[2..6] ]

map (((+) . head) [2..5]) [3..6]
```

```
foldl  (\x _ -> (head x)+(head (tail x)):x)  [1,1]  "Haskell!"
```

```
zipWith (^) [1..5] [2..4]
```

Solution :

```
Prelude> filter (>5) [ x+y | x<-[1..3], y<-[2..6] ]
[6,7,6,7,8,6,7,8,9]
Prelude> map (((+) . head) [2..5]) [3..6]
[5,6,7,8]
Prelude> foldl  (\x _ -> (head x)+(head (tail x)):x)  [1,1]  "Haskell!"
[55,34,21,13,8,5,3,2,1,1]
Prelude> zipWith (^) [1..5] [2..4]
[1,8,81]
```

Barème : 1 point par question

6 Polynômes

Pour cet exercice vous devez donner un programme de Haskell qui peut manipuler des polynômes. On utilise la déclaration suivante :

```
type Poly = [(Int,Int)]
```

L'idée est qu'on va représenter un polynôme par une liste de paires de `Int`. Chaque paire représente un monôme : le premier élément est le degré du monôme, et le deuxième le coefficient. Donc, la paire `(5,3)` représente le monôme $3x^5$, alors que la liste `[(0,7),(1,3),(3,5)]` représente le polynôme $5x^3+3x+7$.
NB: Pour cet exercice on ne considère que des entier non–négatifs pour les coefficients et les degrés.

Une liste de type `[(Int,Int)]` est une représentation valide d'un polynôme si elle satisfait les conditions suivantes : (i) tous les monômes ont un coefficient strictement positif et un degré non–négatif (ii) pour chaque $d \geq 0$ il existe au plus un monôme de degré d dans la liste.

Donc, les listes `[(1,1),(3,0),(4,2)]` et `[(1,1),(1,3)]` ne sont pas valides.

On vous demande de programmer (au moins) les fonctions suivantes.

1. `evalPoly` qui étant donné un polynôme et un `Int` x_0, retourne la valeur du polynôme évalué pour $x = x_0$.

2. `addPoly` qui étant donné deux polynômes retourne une représentation valide de leur somme (qui est aussi un polynôme).

3. `multPoly` qui retourne une représentation valide du produit de deux polynômes.

4. `poly2string` retourne une chaîne de caractères qui est un "bon affichage" du polynôme donné. Un bon affichage satisfait au moins les conditions suivantes : (i) les monômes de degré $d \geq 2$ et coefficient c sont affichés comme cx^d (ii) les monômes de degré $d = 1$ sont affichés comme cx (iii) les monômes de degré $d = 0$ sont affichés comme c (iv) pour les monômes avec coefficient $c = 1$ on n'affiche pas le coefficient. Donc, par exemple, on affiche $x^4 + 2x^2 + 5x + 7$ plutôt que $1x^4 + 2x^2 + 5x^1 + 7x^0$. Pour simplifier la tâche, vous pouvez afficher les monômes dans n'importe quel ordre.

(6 points)

Exemples d'utilisation :

```
*Main> poly2string [(0,4),(1,2),(3,1),(5,7)]
"7x^5+x^3+2x+4"
*Main> poly2string $ addPoly [(0,4),(1,2),(5,7)] [(1,6),(3,8)]
"7x^5+8x^3+8x+4"
-- NB : (7x^5+2x+1) + (8x^3+6x)
*Main> poly2string $ multPoly [(0,1),(1,2),(2,1)] [(0,1),(1,1)]
"x^3+3x^2+3x+1"
-- NB : (x^2+2x+1)*(x+1)
*Main> evalPoly [(0,5),(1,3),(2,2)] 4
49
-- NB : 2x^2+3x+5 pour x=4
```

Solution :

```
-- Assume all int >=0 coefficients and degrees
type Poly = [(Int, Int)]

deg :: Poly -> Int
deg xs = foldr (\(a,b) y -> max a y) 0 xs

getcoeff :: Poly -> Int -> Int
```

```haskell
getcoeff xs y = foldr (\(a,b) z -> if a==y then b else z) 0 xs

poly2string :: Poly -> String
poly2string xs = foldl (\y x -> printMonom x ++ "+" ++ y)
                   (printMonom $ head xs) (tail xs)

printMonom :: (Int,Int) -> String
printMonom (d,0) = ""
printMonom (1,1) = "x"
printMonom (0,x) = show x
printMonom (1,x) = show x ++ "x"
printMonom (d,1) = "x^" ++ show d
printMonom (d,x) =  show x ++ "x^" ++ show d


addPoly :: Poly -> Poly -> Poly
addPoly xs ys = [ (d,getcoeff xs d + getcoeff ys d)
             | d<-[0..(max (deg xs) (deg ys))], getcoeff xs d + getcoeff ys d /=0]

evalPoly :: Poly -> Int -> Int
evalPoly xs x = foldr (\(a,b) s-> b*x^a+s) 0 xs

multPoly :: Poly -> Poly -> Poly
multPoly xs ys = fix brute
   where
      brute = [ (d1+d2,getcoeff xs d1 * getcoeff ys d2)
         | d1<-[0..deg xs], d2<-[0..deg ys], getcoeff xs d1 * getcoeff ys d2 /=0 ]
      fix [] = []
      fix ((a,b):xs) = (a,b+foldr (\(c,d) s-> d+s) 0 (filter (\(c,d) -> c==a) xs))
```

Barème : 1 point pour eval, add, 2 points pour mult, print.

Annexe

Consignes

- Essayer de justifier vos réponses aussi bien que possible.
- Pour toutes les fonctions que vous définissez, vous devez préciser une signature de type.
- Pour les exercices de programmation, vous avez le droit de définir d'autres fonctions secondaires en plus des fonctions demandées (mais il faut aussi bien préciser leurs types).
- Vous pouvez utiliser toutes les fonctions de Haskell de la liste suivante. (Pour chaque fonction on vous rappelle aussi son type). Toute autre fonction que vous utilisez doit être implémentée.

Fonctions de base

```
(:) :: a -> [a] -> [a]
head :: [a] -> a
tail :: [a] -> [a]
last :: [a] -> a
(++) :: [a] -> [a] -> [a]
(!!) :: [a] -> Int -> a
length :: [a] -> Int
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
reverse :: [a] -> [a]
repeat :: a -> [a]
(.) :: (b -> c) -> (a -> b) -> a -> c
($) :: (a -> b) -> a -> b
map :: (a -> b) -> [a] -> [b]
zip :: [a] -> [b] -> [(a, b)]
filter :: (a -> Bool) -> [a] -> [a]
concatMap :: (a -> [b]) -> [a] -> [b]
foldl :: (b -> a -> b) -> b -> [a] -> b
foldr :: (a -> b -> b) -> b -> [a] -> b
dropWhile :: (a -> Bool) -> [a] -> [a]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
fst :: (a, b) -> a
snd :: (a, b) -> b
(^) :: (Num a, Integral b) => a -> b -> a
(+) :: Num a => a -> a -> a
(-) :: Num a => a -> a -> a
(*) :: Num a => a -> a -> a
(==) :: Eq a => a -> a -> Bool
(/=) :: Eq a => a -> a -> Bool
(/) :: Fractional a => a -> a -> a
div :: Integral a => a -> a -> a
mod :: Integral a => a -> a -> a
max :: Ord a => a -> a -> a
min :: Ord a => a -> a -> a
(<) :: Ord a => a -> a -> Bool
(<=) :: Ord a => a -> a -> Bool
id :: a -> a
```

Solution :