Programmation fonctionnelle

Devoir surveillé intermédiaire

19 février 2017

Durée : 1h15
Documents (notes de cours, TD, TP et mini-haddock) autorisés

Vous composerez votre DS sur deux copies séparées afin de paralléliser la correction.

1 Première copie

À composer sur la première copie

Q 1. Indiquez votre groupe sur votre première copie.

1.1 Typage

Soient les définitions suivantes :

```
v1       = "a" : "bc"
v2       = 'a' : "bc"
v3       = Just True
v4 x y z = (x z, y z)
```

Q 2. Pour chaque v_i, indiquez s'il est bien ou mal typé : s'il est bien typé, donnez son type (le plus général que vous voyez) ; s'il est mal typé, expliquez pourquoi.

Q 3. Donnez le type de la fonction `f` définie ainsi :

```
f True x = x
f _    _ = False
```

et expliquez en français ce qu'elle calcule.

Q 4. Définissez une liste infinie de `'a'` de deux façons différentes :

- en utilisant la fonction `iterate`,
- en utilisant une définition récursive.

Q 5. Définissez une fonction respectant le type suivant :

```
f :: (a -> b) -> (b -> c) -> a -> c
```

puis expliquez en français ce qu'elle calcule.

1.2 Structures de données pour une arborescence

Q 6. Définissez une structure de données `Arborescence` permettant de représenter une arborescence d'un système de fichiers avec des *répertoires* et des *fichiers* ordinaires :

- les répertoires auront un nom (de type `String`) et contiendront une liste d'entrées,
- les fichiers auront un nom (de type `String`).

Q 7. Définissez une fonction permettant de calculer le nombre de fichiers présents dans une `Arborescence`.

2 Seconde copie

À composer sur la seconde copie

Q 8. Indiquez votre groupe sur votre seconde copie.

2.1 Crible d'Ératosthène

L'objectif de cet exercice est de créer une liste `primes :: [Integer]` de nombres premiers en utilisant le crible d'Ératosthène.

Le crible d'Ératosthène est un algorithme qui permet de trouver tous les nombres premiers qui sont inférieurs à un entier N. Pour cela, l'algorithme va supprimer d'une liste des entiers de 2 à N, tous les multiples d'un entier. Tous les entiers qui n'auront pas été supprimés à la fin seront premiers.

Crible d'Ératosthène sur les nombres entiers jusqu'à 10

Illustrons l'algorithme sur la liste `[2..10]`.

`[2,3,4,5,6,7,8,9,10]`

2 est premier, ses multiples ne le sont pas. Supprimons donc tous les multiples de 2.

`[2,3,5,7,9]`

Maintenant que nous savons que 2 est premier, appliquons le même algorithme au reste de la liste, c'est-à-dire `[3,5,7,9]`. 3 n'est pas un multiple de 2 (sinon nous l'aurions supprimé de la liste), il est donc premier. Supprimons alors tous ses multiples.

`[2,3,5,7]`

Maintenant que nous savons que 3 est premier, appliquons le même algorithme au reste de la liste, c'est-à-dire `[5,7]`. 5 n'est pas un multiple de 2 et 3, il est donc premier. Supprimons tous les multiples de 5 (ce qui ne change rien, il n'y en a déjà plus dans notre liste).

`[2,3,5,7]`

7 n'est pas un multiple de 2, 3 et 5, il est donc premier. Supprimons tous les multiples de 7 (ce qui ne change rien ici).

La liste a été parcourue, les nombres premiers sont donc :

`[2,3,5,7]`

Questions

Q 9. Réalisez une fonction `removeMultiples :: Integer -> [Integer] -> [Integer]` qui prend en argument un nombre `n` et une liste `ns` et qui retourne la liste `ns` sans les multiples de `n`.

On préférera une définition qui n'est pas récursive.

```
ghci> removeMultiples 2 [3,4,5,6,7,8,9,10]
[3,5,7,9]
ghci> removeMultiples 3 [5,7,9]
[5,7]
ghci> removeMultiples 5 [7]
[7]
```

Q 10. Réalisez une fonction `eratosthene :: [Integer] -> [Integer]` qui applique le crible d'Ératosthène en utilisant récursivement la fonction précédente. Faites attention à bien garder l'élément dont vous supprimez les multiples.

```
ghci> eratosthene [2,3,4,5,6,7,8,9,10]
[2,3,5,7]
```

Q 11. Définissez la liste (infinie) `primes :: [Integer]` de tous les nombres premiers en utilisant le crible d'Ératosthène.

Q 12. Comment pouvez-vous récupérer la liste des 100 premiers nombres premiers ?

2.2 File

L'objectif de cet exercice est de définir une structure de données pour une file. On rappelle qu'une file de valeurs de type `a` est une structure dans laquelle on peut *enfiler* (ajouter) et *défiler* (extraire) des valeurs de type `a`, en suivant le principe « premier entré, premier sorti ». Dans cet exercice, nous considérons les quatre opérations suivantes :

— obtenir une file vide : `fileVide`,
— tester si une file est vide : `estFileVide`,
— enfiler un élément dans la file : `enfile`,
— défiler un élément de la file : `defile`.

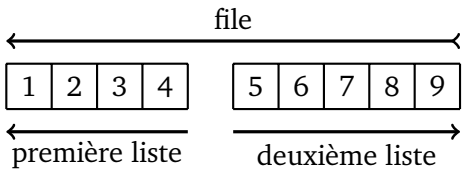

FIG. 1: File contenant les nombres de 1 à 9, dans cet ordre

Une représentation purement fonctionnelle standard pour une file est d'utiliser une paire de listes, comme représenté à la figure 1. Dans la figure, la tête des flèches marque la tête des listes et de la file. La file contenant les nombres de 1 à 9 (dans cet ordre) peut être représentée par la paire de listes `([1,2,3,4],[9,8,7,6,5])`.

La file vide est représentée par la paire de listes vides. Pour enfiler un élément, on l'ajoute en tête de la deuxième liste. Pour défiler un élément :

— si la première liste est non vide, on enlève l'élément en tête de la première liste,
— sinon, on transvase la deuxième liste dans la première (en inversant l'ordre), puis on enlève l'élément en tête de la première liste.

Voici un exemple illustrant les opérations :

```
type File a = ([a],[a])

f1      = fileVide       --            f1 = ([], [])
f2      = enfile 5  f1   --            f2 = ([], [5])
f3      = enfile 7  f2   --            f3 = ([], [7,5])
f4      = enfile 11 f3   --            f4 = ([], [11,7,5])
(x5,f5) = defile f4      -- x5 = 5  et f5 = ([7,11], [])
f6      = enfile 13 f5   --            f6 = ([7,11], [13])
(x7,f7) = defile f6      -- x7 = 7  et f7 = ([11], [13])
(x8,f8) = defile f7      -- x8 = 11 et f8 = ([], [13])
(x9,f9) = defile f8      -- x9 = 13 et f9 = ([], [])
```

Q 13. Définissez `fileVide` et `estFileVide`, y compris leur type.

Q 14. Définissez `enfile` et `defile`, y compris leur type.

Q 15. Expliquez pourquoi cette implémentation est plus efficace que l'utilisation d'une liste (où l'on doit soit ajouter toujours à la fin de la liste, soit retirer toujours le dernier élément de la liste).