Fiche de TD-TP no. 2

Typage

Exercice 1. Trouvez les erreurs de type dans les définitions suivantes, et corrigez les :

```
shuffle [] ys = ys
shuffle (x:xs) ys = x:shuffle ys++xs

map f [] = []
map f (x:xs) = f x:map (f xs)
```

Conseil :

1. devinez ce que la fonction devrait faire,
2. proposez donc son type,
3. vérifiez ensuite la cohérence de l'utilisation de la fonction dans la définition.

Exercice 2. Calculez le type des fonctions définies dans le script suivant :

```
split xs y = ([ x |  x <- xs, x <= y  ],[ x |  x <- xs, x > y  ])

find x (y:ys)
    | x == y = 0
    | otherwise = 1 + (find x ys)

fold op [] x = x
fold op (y:ys) x = y `op` (fold op ys x)
```

Quelles sont les fonctions polymorphes ? (N'oubliez pas de mentionner les contraintes de type.)

Exercice 3. Considérez la fonction `f` définie par :

```
f [] x d = d x
f ((y,z):ws) x d
  | (y x) =  z x
  | otherwise = f ws x d
```

1. Est que on y utilise, dans la définition de `f` : le filtrage, les conditions de garde, la récursion ? Justifiez votre réponse.
2. Écrivez l'expression de type de `f`.
3. Que fait cette fonction ?
4. Proposez une règle de typage pour les définitions par équations avec conditions de garde.

Conditionnels et filtrage

Exercice 4. Considérez la fonction `safetail` qui se comporte exactement comme `tail`, sauf qu'elle envoie la liste vide vers elle même (rappel : `tail []` produit un erreur).
Définissez `safetail` en utilisant :

(a) une expression conditionnelle ;

(b) des équations avec conditions de garde ;

(c) le filtrage par motifs.

Aide : la fonction du prelude `null :: [a] -> Bool` teste si une liste est vide.

Exercice 5. Donnez trois possibles définitions de l'opérateur logique `(||)`, en utilisant le filtrage par motifs.

Fonctions

Exercice 6. Donnez les types de expressions définies dans le script suivant :

```
somme = \x -> \y -> x + y
successeur = somme 1
succ = (+1)
inv 0 = 1
inv _ = 0
inverserLigne = map inv
inverserImage = map inverserLigne
```

Exercice 7. Nous allons denoter B^A l'ensemble des fonctions de domaine A et codomaine B, c'est-à-dire :

$$B^A := \{ f \mid f : A \to B \}.$$

1. Argumentez que pour tout $f : A \times B \to C$ il existe une unique fonction $f' : A \to C^B$ telle que, pour tout $a \in A$ et $b \in B$, on a
$$(f'(a))(b) = f(a, b).$$
2. Argumentez de l'existence d'un fonction bijective $\Lambda : C^{A \times B} \to (C^B)^A$.

Compréhension sur les listes

Exercice 8. Un nombre positif est *parfait* s'il est la somme de tous ses facteurs, sauf lui même. En utilisant la compréhension, définissez une fonction

```
perfects :: Int -> [Int]
```

qui retourne la liste de tous les nombres parfaits, jusqu'à la limite passée en paramètre.

Exercice 9. Un triplet (x, y, z) d'entiers positifs est dit *de Pythagore* si $x^2 + y^2 = z^2$.

1. Définissez, en utilisant la compréhension, une fonction
```
pyths :: Int -> [(Int,Int,Int)]
```
qui envoie un entier n vers la listes de tous les triplets de Pythagore avec composantes dans `[1..n]`.
2. Si (x, y, z) est un triplet de Pythagore, alors (y, x, z) est aussi un triplet de Pythagore. Proposez une solution à l'exercice précèdent, où seulement un triplet parmi (x, y, z) et (y, x, z) apparaît dans la liste retournée.
3. Si (x, y, z) est un triplet de Pythagore, alors $x, y < z$. Proposez un solution de l'exercice précèdent qui utilise cette observation pour accélérer les calculs.

Exercice 10. Le produit scalaire de deux listes d'entiers xs et ys de longueur n est la somme des produits des entiers correspondants :

$$xs \cdot ys = \sum_{i=0}^{n-1} xs_i * ys_i .$$

1. Utilisez la compréhension et les fonctions vues en cours pour définir une fonction qui retourne le produit scalaire de deux listes.
2. Autrement, utilisez l'induction pour définir cette fonction.

En TP

Exercice 11. Vérifiez vos réponse aux exercices de TD 1-4, en utilisant l'interprète Haskell.

Exercice 12. Implémentez vos réponses aux exercices de TD 8, 9 et 10 dans un script. Chargez ce script dans l'interprète, expérimentez avec.

Exercice 13. Considérez le script suivant :

```
carre x = x^2

moyenne ns = sum ns/fromIntegral (length ns)

norme ns =  sqrt (moyenne (map carre ns))

main = print (norme [1..5])
```

1. Ajoutez dans le script la déclaration du type avant chaque fonction définie.
2. En utilisant la notation lambda, éliminez du script toute définition intermédiaire (*carre*, *moyenne*, *norme*) en définissant ainsi le *main* dans une seule expression.

Exercice 14. Considérez l'algorithme `qsort` présenté en premier cours :

```
qsort [] = []
qsort (x:xs) =
    qsort smaller ++ [x] ++ qsort larger
    where
      smaller = [a | a <- xs, a < x]
      larger  = [b | b <- xs, x < b]
```

1. Copiez cet algorithme dans un script, qui sera chargé avec `ghci`.
2. Tapez `:type qsort` dans `ghci` et expliquez la réponse donnée.
3. Tapez `:info Ord` dans `ghci` et decodez la réponse donnée.