Fiche de TD-TP no. 5

Évaluation

Exercice 1. Utilisez d'abord la stratégie *call-by-value* et ensuite la stratégie *call-by-name* pour évaluer les expressions suivantes :

```
f (True && True) [] (replicate 2 '*') where f b x y = if b then x else y
take 2 (repeat '*')
add (Succ Zero) Zero
(\ x - > \ y -> x ) 5 omega where omega = omega + 1
```

Exercice 2. Rappelons les définitions des fonctions `foldr`, `foldl`, `map`, et `(++)` :

```
foldr :: (a -> b -> b) ->  b ->  [a] -> b
foldr f v [] = v
foldr f v (x:xs) = f x (foldr f v xs)

foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f v [] = v
foldl f v (x:xs) = foldl f (f v x) xs

map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = f x:map f xs

(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x:xs) ++ ys = x:(xs ++ ys)
```

Evaluez, par nom et par valeur, les expressions suivantes :

```
exp1 = foldr (&&) True (map even [0..])
exp2 = foldl (&&) True (map even [0..])
exp3 = foldr (++) [] (map (\ x -> [x]) [1,2])
exp4 = foldl (++)  [] (map (\ x -> [x]) [1,2])
```

Exercice 3. Considérez les définitions suivantes des la fonction `fib :: Int -> Int` qui associe à un entier n le n-ème nombre de Fibonacci :

```
fib1,fib2,fib3 :: Int -> Int

fib1 0 = 1
fib1 1 = 1
fib1 n = fib1 (n-1) + fib1 (n-2)

fib2 0 = 1
fib2 1 = 1
fib2 n = fibliste !! (n -1) + fibliste !! (n-2)
fibliste = map fib2 [0..]

fib3 n = fib3acc n 1 1
  where
```

```
fib3acc 0 n m = n
fib3acc k n m =  fib3acc (k -1) m (m + n)
```

Quelle est, à votre avis, la plus (resp. la moins) performante ? Justifiez votre réponse.

Raisonner sur les programmes

Exercice 4. Après avoir rappelé la définition de la fonction `(++)`, démontrez les égalités suivantes :

```
[] ++ ys == ys
xs ++ [] == xs
(xs ++ ys) ++ zs == xs ++ (ys ++ zs)
```

Exercice 5. Rappelez les définitions des fonctions `length` et `replicate` et, ensuite, démontrez l'égalité suivante :

```
length (replicate x n) == n
```

Exercice 6. Démontrez que `fib1` et `fib3` de l'exercice 3 calculent la même fonction.

En TP

Discuter et implémenter le projet.

Exercice 7. Implémentez l'algorithme d'unification comme une fonction de type

```
unify :: [(Terme,Terme)] -> Maybe Substitution
```

Exercice 8. Dans cet exercice on se propose d'écrire un script `testTerm.hs` qui teste votre module `Term.hs` développé avec le TP no 4 (la notation est celle du texte du TP 4 en ligne et à jour).

1. Instanciez les type des variables, des symboles de fonctions, des termes, et de substitutions à la classe `Arbitrary`.
2. Utilisez le module `QuickCheck` pour tester les lois suivantes :

```
(t *! sigma) *! tau ==  t *! (tau @@ sigma)
(rho @@ sigma) @@ tau ==  rho @@ (tau @@ sigma)
```

Exercice 9. Devisez une stratégie pour tester la qualité de la fonction `unify` en utilisant le module `QuickCheck`. Implémentez cette stratégie dans le script `testTerm.hs`.