Principes de Programmation
TD3

14 mars 2018

Exercice 1 (Monoïdes).
La classe `Monoid` est définie par :

```
class Monoid m where
  mempty :: m
  (<>)   :: m -> m -> m
```

Un monoid `m` doit vérifier les axiomes suivants :

```
  mempty <> x         ~=   x
  x <> mempty         ~=   x
  (x <> y) <> z       ~=   x <> (y <> z)
```

1. Donnez des exemples de monoides.
2. Montrez que `[a]` est un monoid. [1]
3. Montrez que `(a -> a)` est un monoid. [2]
4. (bonus) Montrez que si `m` est un monoid, alors `(a -> m)` est un monoid. Vous avez le droit d'utiliser la rêgle suivante pour vos réductions :

   ```
   (\x.f x) ~= f
   ```

5. (bonus) On définit : [3]

   ```
   newtype Dual a = Dual a

   instance Monoid a => Monoid (Dual a) where
     mempty               = Dual mempty
     (Dual x) <> (Dual y) = Dual (y <> x)
   ```

   Que fait `Dual` ?

1. Uniquement valable sur les listes finies
2. Malheureusement, celui-ci n'est pas implémenté dans Haskell. La raison est qu'il a fallu choisir avec celui de la question suivante dont le type est en conflit.
3. "newtype" est comme "data" mais avec un seul constructeur : c'est un wraper

Exercice 2 (Foldables (Bonus)).
La classe `foldable` est définie par :

```
class Foldable struct where
  foldr   :: (a -> b -> b) -> b -> struct a -> b
  foldl   :: (b -> a -> b) -> b -> struct a -> b
  foldMap :: Monoid m => (a -> m) -> struct a -> m
```

Il n'y a pas d'équations explicite entre ces définitions, mais il n'est demandé qu'une fonctions, les autres étant traduites.

1. Écrire `foldl`, `foldr` et `foldMap` pour les listes.
2. Comment écrire `foldMap` à partir de `foldr` ?
3. Vérifiez que la traduction est correcte sur les listes.
4. Comment écrire `foldMap` à partir de `foldl` ?
5. Vérifiez que la traduction est correcte sur les listes.
6. Comment écrire `foldr` à partir de `foldMap` ? [4]
7. Vérifiez que la traduction est correcte sur les listes.
8. Comment écrire `foldl` à partir de `foldMap` ?
9. Vérifiez que la traduction est correcte sur les listes.
10. Montrez que `Tree` est foldable.
11. Montrez que `Maybe` est foldable.

4. On supposera que $(a \to a)$ est bien un monoid. Dans Haskell, il est utilisé un wraper qui complexifie les choses.