Principes de Programmation
TD4

27 mars 2018

Exercice 1 (Une monade est un foncteur).
On rappelle que la classe `Functor` est définie par :

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

La classe `Monad` est définie par : [1]

```
class Monad m where
  return :: a -> m a
  (=<<)  :: (a-> m b) -> m a -> m b
```

1. Encodez les foncteurs dans les monades. C'est à dire, supposez que `m` est une monade, et implémentez `Functor m`.

Cela ne suffit pas pour dire qu'une monade est toujours un foncteur, il faut vérifier aussi les axiomes de foncteur.

On rappelle qu'un foncteur doit vérifier les axiomes suivants :

```
fmap id      =   id
fmap (f.g)   =   fmap f . fmap g
```

De la même façon, une monade doit vérifier les axiomes suivants :

```
return =<< x       =   x
f =<< (return x)   =   f x
f =<< (g =<< x)    =   (\y -> f =<< (g y) ) =<< x
```

2. (Dificile) Faites ces vérifications. Càd, supposez que `m` vérifie les axiomes de monade, et vérifiez que votre encodage vérifie les axiomes de foncteur.

1. Attention, pour des raisons historiques, il faut définir $(>>=) :: ma- > (a- > mb)- > mb$ plutôt que $=<<...$

Exercice 2 (Les monades).
On rappelle que `Maybe` est définie ainsi :

```
data Maybe a = Something a  |  Nothing
```

1. Montrez que `Maybe` est une monade.

On définit `Either a b` par

```
data Either a b = Left a | Right b
```

2. Montrez que pour tout `a`, `Either a` est une monade.
3. Montrez que les listes forment une monade.

On définit les arbres de préfixe ainsi :

```
data Tree a = Node (Char -> Tree a)  |  Leaf a
```

4. Montrez que les arbres de préfixe forment une monade.

On définit `Futur` qui encode le `yield` de python :

```
type Tick = ()
data Futur a = Now a | Later (Tick -> Futur a)
```

5. Montrez que `Yield` est une monade.

Un type à état est défini (presque) comme suit :

```
type ST s a = (s -> (a,s))
```

où `s` est le type de l'état courant et `a` est le type sur lequel on travaille.

6. Montrez que `ST s` forme une monade (quelque soit `s`).

La monade de continuation [2] est définie par :

```
type Cont r a = (a -> r) -> r
```

7. Montrez que `Cont r` forme une monade (quelque soit `r`).

2. importée de Control.Monad.Trans.Cont