TD 3 : Traits fonctionnels de SCALA

Université Paris Diderot – Master 2 SRI/LC/LP

21 novembre 2011

1 Manipulation de listes

Exercice 1 (Fonctions classiques sur les listes) *On se donne les définitions suivantes :*

```
abstract class List[+A]
case object Empty extends List[Nothing]
case class Cons[A](h:A, t:List[A]) extends List[A]

object List {
  def foldRight[A, B](as:List[A], b:B, f:(A, B) => B):B = as match {
    case Empty => b
    case Cons(h, t) => f(h, foldRight(t, b, f))
  }

  def foldLeft[A, B](as:List[A], b:B, f:(B, A) => B):B = as match {
    case Empty => b
    case Cons(h, t) => foldLeft(t, f(b, h), f)
  }

  def reduceRight[A](as:List[A], f:(A, A) => A):A = as match {
    case Empty => error("bzzt. reduceRight on empty list")
    case Cons(h, t) => foldRight(t, h, f)
  }

  def reduceLeft[A](as:List[A], f:(A, A) => A):A = as match {
    case Empty => error("bzzt. reduceLeft on empty list")
    case Cons(h, t) => foldLeft(t, h, f)
  }

  def unfold[A, B](b:B, f:B => Option[(A, B)]):List[A] = f(b) match {
    case Some((a, b)) => Cons(a, unfold(b, f))
    case scala.None => Empty
  }
}

abstract class Natural
case object Zero extends Natural
case class Succ(c:Natural) extends Natural
```

1. Complétez le programme SCALA *suivant.*

```
// Compute the sum of two natural numbers.
def add(x:Natural, y:Natural):Natural = error("todo")

// Compute the sum of a list of natural numbers
def sum(is:List[Natural]):Int = error("todo")

// Compute the length of a list.
def length[A](as:List[A]):Natural = error("todo")

// Compute the list that is composed of the application of f to
// every element of as.
def map[A, B](as:List[A], f:A => B):List[B] = error("todo")

// Compute the sublist of as whose elements verify f.
def filter[A](as:List[A], f:A => Boolean):List[A] = error("todo")

// Compute the list that is the concatenation of x and y.
def append[A](x:List[A], y:List[A]):List[A] = error("todo")

// Compute the concatenation of lists of as.
def flatten[A](as:List[List[A]]):List[A] = error("todo")

// Compute the concatenation of the lists that come from the
// application of f to elements of as.
def flatMap[A, B](as:List[A], f:A => List[B]):List[B] = error("todo")

// Compute the maximal element of is.
def maximum(is:List[Natural]):Int = error("todo")

// Compute the list that is the mirror of as.
def reverse[A](as:List[A]):List[A] = error("todo")
```

*2. Cherchez la **classe** `List` dans la documentation de la bibliothèque standard de* SCALA. *Comparez les fonctions proposées par cette bibliothèque avec les fonctions que vous venez d'écrire.*
*3. Il existe aussi un **objet** `List` dans la bibliothèque standard. À quoi sert cet objet ?* □

Exercice 2 (Liste paresseuse (Stream)) *On définit une structure de liste paresseuse :*

```
abstract class Stream[+A] {
  def head :A
  def tail :Stream[A]
  def isEmpty :Boolean
}

class Cons[A] (val head :A, _tail :=> Stream[A]) extends Stream[A] {
  private var tlVal:Stream[A] = _
  override def tail:Stream[A] = {
    if (tlVal == null) { tlVal = _tail }
    tlVal
  }
  def isEmpty = false
}

object #:: {
  def unapply[A](xs:Stream[A]):Option[(A, Stream[A])] =
    if (xs.isEmpty) None
    else Some((xs.head, xs.tail))
}

object cons {
  def apply[A](hd:A, tl:=> Stream[A]) = new Cons(hd, tl)
  def unapply[A](xs:Stream[A]):Option[(A, Stream[A])] = #::.unapply(xs)
}

object empty extends Stream[Nothing] {
   override def isEmpty = true
  override def head =
    throw new NoSuchElementException("head of empty stream")
  override def tail =
    throw new UnsupportedOperationException("tail of empty stream")
}
```

1. En vous référant à la documentation du langage, pouvez-vous expliquer à quoi servent les méthodes `apply` et `unapply` d'un objet en SCALA *?*
2. En quoi les définitions précédentes forment des listes paresseuses ?
3. Que fait le programme suivant ?

```
  def toList[A] (s :Stream[A]) :List[A] =
    s match {
      case x #:: xs => x :: toList (xs)
      case s if s.isEmpty => Nil
    }

  def intFrom (n :Int) :Stream[Int] =
    cons (n, intFrom (n + 1))

  def run1 = {
    var naturals = intFrom (0)
    println (toList (naturals))
  }
```

4. Modifiez la classe `Stream[A]` en rajoutant :
- *une méthode `take (x : Int) : List[A]` qui extrait x éléments de la liste ;*
- *une méthode `filter (p : A => Boolean) : Stream[A]` qui définit la sous-liste paresseuse des éléments vérifiant p.*

5. Implémentez le crible d'Ératosthène à l'aide d'une liste paresseuse. (Indice : Utilisez la méthode `filter`.)

□

2 Définition par compréhension

Exercice 3

1. Lisez la section 6.19 intitulée For comprehensions and for loops du manuel de référence de SCALA.

2. Implémentez une fonction qui affiche tous les entiers inférieurs à 100 qui sont premiers en les testant un à un à l'aide d'une boucle définie par compréhension.

3. Voici un programme :

```
case class Book(title:String, authors:String*)
val books:List[Book] =
  List(
    Book("Structure and Interpretation of Computer Programs", "Abelson, Harold", "Sussman,
Gerald J."),
    Book("Principles of Compiler Design", "Aho, Alfred", "Ullman, Jeffrey"),
    Book("Programming in Modula-2", "Wirth, Niklaus"),
    Book("Elements of ML Programming", "Ullman, Jeffrey"),
    Book("The Java Language Specification", "Gosling, James", "Joy, Bill", "Steele, Guy",
"Bracha, Gilad")
  )
```

Écrivez une expression qui calcule la liste des livres dont le nom d'un des auteurs commence par un 's'.

3. Écrivez une expression qui calcule la liste des auteurs qui sont l'auteur de plusieurs listes de cette liste.

4. Écrivez une fonction qui supprime les duplicats dans une liste.

5. Écrivez les fonctions `map`, `filter` et `flatMap` travaillant sur les listes de l'exercice 1 à l'aide d'une compréhension.

□

3 Fonction partielle

Exercice 4 (Image avec des bords)

1. Lisez la documentation de la classe `PartialFunction` de la bibliothèque standard de SCALA.

2. Implémentez une classe `Image` héritant de `PartialFunction` que l'on peut instancier en fournissant sa hauteur et sa largeur. Une image est une fonction partielle définie uniquement pour des coordonnées comprises dans ses dimensions. On supposera que la couleur d'un pixel est représentée par un entier.

2. Écrire une fonction qui attend deux images I et K et opèrent la convolution de la première par la seconde définie ainsi :

$$(I \star K)(i,j) = \Sigma_{(k,l)\in dom(K)} K(k,l) * I(i + (k - largeur(K)/2), j + (l - hauteur(K)/2))$$

*pour écrire cette fonction facilement, on **étendra** l'image I en lui rajoutant des bords de taille infinie de couleur noire.*

□