03 Lab
More Functional Programming in Scala
Seqs, Streams, and more

Mirko Viroli, Roberto Casadei, Gianluca Aguzzi
{mirko.viroli,roby.casadei,gianluca.aguzzi}@unibo.it

C.D.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2023/2024

Lab 03: Outline

Outline

- Continue with the practice of functional programming
- Exercise with lists and streams (these are key functional data structures!)

Getting started

- Fork repository `https://github.com/unibo-pps/pps-23-24-lab03`
- The repo contains code from lectures 02 and 03
- Solve the exercises of the following slides in a proper module and evaluate your solutions through a main program or test class

Tasks – part 1 (lists)

1. Consider the `Sequence` type, create the following functions:
   a) `def take[A](l: Sequence[A], n: Int): Sequence[A]`

```
val lst = Cons(10, Cons(20, Cons(30, Nil())))
take(lst, 2) // Cons(10, Cons(20, Nil()))
take(lst, 0) // Nil()
take(lst, 5) // Cons(10, Cons(20, Cons(30, Nil())))
```

   b) `def zip[A,B](l:Sequence[A], r:Sequence[B]):Sequence[(A,B)]`

```
val lst1 = Cons(10, Cons(20, Cons(30, Nil())))
val lst2 = Cons("a", Cons("b", Cons("c", Nil())))
zip(lst1, lst2) // Cons((10,a), Cons((20,b), Cons((30,c), Nil())))
```

   c) `def concat[A](l:Sequence[A], r:Sequence[A]): Sequence[B]`

```
val lst1 = Cons(10, Cons(20, Nil))
val lst2 = Cons(30, Cons(40, Nil))
concat(lst1, lst2) // Cons(10, Cons(20, Cons(30, Cons(40, Nil))))
```

   c) `def flatMap[A,B](l: Sequence[A])(f:A => Sequence[B]): Sequence[B]` (hint: use concat)

```
flatMap(lst)(v => Cons(v + 1, Nil()))) // Cons(11, Cons(21, Cons(31, Nil())))
flatMap(lst)(v => Cons(v + 1, Cons(v + 2, Nil())))
// Cons(11, Cons(12, Cons(21, Cons(22, Cons(31, Cons(32, Nil()))))))
```

   d) Write `map` & `filter` in terms of `flatMap`

2. Considering both `Sequence` and `Optional` (`u3.Optionals.Optional`), create the following:
   `def min(l: Sequence[Int]): Optional[Int]`

```
min(Cons(10, Cons(25, Cons(20, Nil())))) // Maybe(10)
```

Tasks – part 2 (more on lists)

3. Consider Person and Sequence as implemented in class slides. Create a function that takes a sequence of Persons and returns a sequence containing only the courses of Student in that list
   - Hint 1: you essentially need to combine filter and map
   - Hint 2: there is a very concise solution that reuses flatMap
4. (Hard) Implement foldLeft function that, starting from a default value, "fold over" sequences by "accumulating" elements via a binary operator.
   - Idea: given a list [3, 7, 1, 5] and a default value 0, a left-fold (resp., right-fold) through e.g. operator + is given by (((0 + 3) + 7) + 1) + 5.
   - Note: dealing with empty sequences requires providing a default or initial value (0 in the previous example) for the accumulation, to be used on left or on right
   - Note: the type of the accumulator may be different w.r.t. the type of the elements that are aggregated (two generic variables should be used)

```scala
val lst = Cons(3,Cons(7,Cons(1,Cons(5, Nil()))))
foldLeft(lst)(0)(_ - _) // -16
```

5. Implement all the above functions as extension methods for the Sequence type

Tasks – part 3 (streams)

6. Consider the `Stream` type discussed in class. Define a function, called `takeWhile(s)(n)`, that returns the first `n` elements of the stream `s` that satisfy a given predicate. .

```
val s = s.iterate(0)(_ + 1)
Stream.toList(Stream.takeWhile(s)(_ < 5)) // Cons(0,
    Cons(1, Cons(2, Cons(3, Cons(4, Nil())))))
```

7. Implement a generic function `fill(n)(k)` that creates a stream of `n` elements, each of which is `k`.

```
Stream.toList(Stream.fill(3)("a")) // Cons(a, Cons(a,
    Cons(a, Nil())))
```

8. Implement an infinite stream for the Pell Numbers: `https://en.wikipedia.org/wiki/Pell_number`

```
val pell: Stream[Int] = ???
Stream.toList(Stream.take(pell)(5)) // Cons(0, Cons(1,
    Cons(2, Cons(5, Cons(12, Nil())))))
```