INFO0054 Programmation Fonctionnelle – Exercises

Christophe Debruyne

Exercises 4: ADTs and Recursion

Exercise 1:

> **Attention!**
>
> We have seen this example in class. This is a warm up exercise. Try not to look at the slides (especially for the first part of the exercise).

The first two Fibonacci numbers are 0 and 1. The nth Fibonacci number is always the sum of the previous two Fibonacci numbers. The sequence begins with 0, 1, 1, 2, 3, 5,... You may assume that n=0 corresponds with the first Fibonacci number.

Define `fib1`, a recursive function to get the nth Fibonacci number. Is your definition tail recursive? Justify your answer. If `fib1` is tail-recursive, define a version `fib2` that is recursive, but not tail recursive. If `fib1` is recursive, `fib2` should use a local tail-recursive function.

```scala
scala> val x = List.range(0,15).map(fib1)
val x: List[Int] = List(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377)
```

Exercise 2:

Define `sumInt1`, a recursive function that takes as argument a natural number (positive integer in Scala) n and returns the sum of all natural numbers lower than or equal to n. Is your definition tail recursive? Justify your answer. If `sumInt1` is tail-recursive, define a version `sumInt2` that is recursive, but not tail recursive. If `sumInt1` is recursive, `sumInt2` should use a local tail-recursive function.

Exercise 3:

Define `power1`, a recursive function taking as arguments a number x and a natural number n, and returning x^n. Is your definition tail recursive? Justify your answer. If `power1` is tail-recursive, define a version `power2` that is recursive, but not tail recursive. If `power1` is recursive, `power2` should use a local tail-recursive function.

Hard(er): We know that if n is even, then $x^n = (x * x)^{\frac{n}{2}}$. Modify the solutions above to render it more efficient.

Exercise 4:

Define `taken1`, a recursive function taking as arguments a list `l` and a natural number `n`. The function returns a new list containing the first `n` elements of that list (or fewer if the list does not contain enough elements).

What kind of recursion is this?

Is your definition tail recursive? Justify your answer. If `taken1` is tail-recursive, define a version `taken2` that is recursive, but not tail recursive. If `taken1` is recursive, `taken2` should use a local tail-recursive function.

Can you come up with two strategies for implementing this function using tail recursion? What are those two strategies and which one is more efficient?

Exercise 5:

Create an ADT for `LTree`s, which are labelled binary trees. The constructors are `LLeaf` and `LBranch`.

Now use your ADT to create a representation for the following mathematical expression on Doubles: $((3.0 + 5.0) + (3.0 - 4.0)) \times (3.0/2.0)$. You can use the String objects "ADD", "SUB", "DIV", and "MUL" to represent the arithmetic operations. By mixing String and Int objects, you will obtain a `LTree[Matchable]`

Then create a function `compute` in `LTree`'s companion object that, given an `LTree` containing an arithmetic expression, computes the result. You may assume that there the tree contains valid values.

Question: Given the lecture on exception handling, how could you solve this exercise using `Option` or `Either`?

Exercise 6:

Using your ADT `LTree`, define a function `transform` that takes as input a `Tree[Int]` and a function `f: (Int,Int)=>Int`, and returns a `LTree[Int]` in which the values of each `LBranch` are computed using the function and the labels of each of its trees. You will need to rely on a function to retrieve the label. For reasons beyond this course, you will need to choose a function name that is different from the names of the labels in your constructors: e.g., `value`.

References

[1] Paul Chiusano and Rnar Bjarnason. 2015. Functional Programming in Scala (2nd. ed.). Manning Publications Co., USA.