Lab 2

Functional Programming (ITI0212)

2022-02-04

This week we learned about inductive types and recursive functions. *Inductive types* are user-defined types with any number of *element constructors*. These specify the possible ways of creating elements of the given type, and each may take different numbers and types of arguments. *Recursive functions* on inductive types use *case analysis* or *pattern matching* in order to specialize the function being defined for the possible element constructors. These functions may call themselves using *recursive calls* to compute the result for the current case using the results for other cases.

Task 1
An important function in digital circuit design is the `xor` function, which takes two `Bool` inputs and returns `True` just in case they differ. Write this function in Idris.

Task 2
The two-element type `Bool` is used to represent the truth or falsity of a proposition. But sometimes we are not so sure about things. Write a four-element type called `Prob` with elements named `Definitely`, `Likely`, `Doubtful`, and `Impossible`.

Task 3
Write a negation function for `Prob`,

```
not  :  Prob -> Prob
```

that sends each element in the above list to the corresponding element of the reversed list (e.g. `Definitely` ↦ `Impossible`).

Task 4
Write a conjunction function for `Prob`,

```
and  :  Prob -> Prob -> Prob
```

according to the following table:

↓ `and` →	`Definitely`	`Likely`	`Doubtful`	`Impossible`
`Definitely`	`Definitely`	`Likely`	`Doubtful`	`Impossible`
`Likely`	`Likely`	`Likely`	`Doubtful`	`Impossible`
`Doubtful`	`Doubtful`	`Doubtful`	`Doubtful`	`Impossible`
`Impossible`	`Impossible`	`Impossible`	`Impossible`	`Impossible`

Challenge: try to write this definition using as few clauses as possible.

Task 5
Write the multiplication function for natural numbers.

```
mul  :  Nat -> Nat -> Nat
```

Hint: try using recursion on the first argument.

Task 6
The *factorial* function $n!$ on the natural numbers can be characterized by the following recursive specification:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise.} \end{cases}$$

Turn this recursive mathematical specification into a recursive function definition in Idris:

```
fact  :  Nat -> Nat
```

Task 7
Extend the `Shape` type from lecture 2 by adding a constructor to represent a regular n-sided polygon with a specified side-length:

```
  RegularPolygon  :  (sides : Nat) -> (length : Double) -> Shape
```

Task 8
Write a function called `perimeter` that returns the linear distance along the boundary of a shape.

Hint: it may help to recall the Pythagorean theorem and to `:search` for functions from `Double` to `Double` and from `Nat` to `Double`.