Lab 2

Functional Programming

2024-02-02

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 argument using the results for other arguments.

Place your work for this lab in a module named `Lab2` in a file named `Lab2.idr` in the `lab` directory of your course repository.

Task 1
An important boolean function is `xor`, which takes two `Bool` inputs and returns `True` just in case they differ:

↓ `xor` →	`True`	`False`
`True`	`False`	`True`
`False`	`True`	`False`

Write this function in Idris using definition by pattern matching.

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

For example:

```
Lab2> Definitely `and` Likely
Likely
Lab2> Impossible `and` Doubtful
Impossible
```

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

Task 5
Recall the following type from lecture, which is meant to represent geometric shapes.

```
data  Shape : Type  where
  -- circle shape with given radius:
  Circle  :  (radius : Double) -> Shape
  -- rectangle shape with given width and height:
  Rectangle  :  (width : Double) -> (height : Double) -> Shape
  -- isosceles triangle shape with given base and height:
  IsosTriangle  :  (base : Double) -> (height : Double) -> Shape
```

Write a function with the following type that computes the area of a shape.

```
area  :  Shape -> Double
```

Hint: `:search Double`.

Task 6
Write the multiplication function for natural numbers.

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

Hint: try using recursion on the first argument.

Task 7
The *factorial* function $n!$ on the natural numbers can be characterized by the following recursive specification:

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

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

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