Lab 2

Functional Programming (ITI0212)

2021.02.02

1. The type `One` from the lecture is built in to Idris as `Unit`. The type `Zero` from the lecture is built in to Idris as `Void`.
   (a) Write a function of type `Bool -> Unit`. How many different functions of this type are there?
   (b) How many functions are there of type `Bool -> Bool`? Write them all.
   (c) Write a function of type `Nat -> Unit`. How many different functions of this type are there?
   (d) How many functions are there of type `Unit -> Nat`? Write one of them down.
   (e) How many functions are there of type `Void -> Void`. Write them all down.
   (f) How many functions are there of type `Nat -> Void`? Write them all down.
   (g) How many functions are there of type `Void -> Nat`? Write them all down.

2. Recall the `Shape` type from the lecture:

   ```
   data Shape : Type where
     Circle : Nat -> Shape
     Rectangle : Nat -> Nat -> Shape
     IsoTriangle : Nat -> Nat -> Shape
   ```

   with the idea being that `Circle k` is the circle of radius `k`, `Rectangle a b` is the rectangle with length `a` and width `b`, and `Isotriangle a b` is the isoceles triangle with base width `a` (one side) and leg length `b` (two sides).

   (a) Write a function `area : Shape -> Double` that computes the area of a `Shape`.
   (b) Write a function `regular : Shape -> Bool` that returns `True` if the input `Shape` is regular (that is, all of its sides are of equal length), and returns `False` otherwise.

(c) Add a type constructor to the `Shape` type to represent regular n-sided polygons. Update your `area` and `regular` functions to account for this new type constructor.

(d) Is our representation of isoceles triangles a good one? Put another way, is is possible to specify every isoceles triangle in the way we have chosen? Does every instance of `(IsoTriangle a b) : Shape` give an isoceles triangle?

3. (a) Write a function `monus : Nat -> Nat -> Nat` that subtracts the second argument from the first. If the second argument is greater than the first, the result should be zero.

(b) Use pattern matching to write a function `even : Nat -> Bool` that returns `True` in case it's input is an even number, and `False` otherwise.

(c) Write a function `odd : Nat -> Bool` that does the same, but for odd numbers.