Fakultät für Informatik und Mathematik Summer semester 2023
Prof. Dr. Martin Griebl

Exercises for Lecture: Functional Programming
Exercise Sheet 3 (Tuples, Functions, Lists)

Problem 1 (Tuples)

Define a function `sort3 :: Ord a => a -> a -> a -> (a, a, a)` which returns the three input values in ascending order in the result triple. Implement `sort3` by using function `sort2` multiple times. Function `sort2` returns its two argument values in ascending order (it can be defined by a simple case distinction). Use a simple drawing (similar to the depiction of the full adder in the lecture) which shows the flow of minima and maxima in `sort3` to plan the implementation of `sort3`.

Note: Type class `Ord` contains all types which can be compared using `<`, `<=`, etc.

Problem 2 (Function Iteration)

Functions can be composed using the '`.`' operator, i.e., `(f . g) x = f (g x)`. Using this operator, define a function `nTimes n f` that expresses the `n`-fold application of function `f`. This means that the function `mul32` (that multiplies its argument by 32) can be defined by the following code:

```
mul2  x = x * 2
mul32   = nTimes 5 mul2
```

Hint: Use recursion and an `if-then-else` clause.

Problem 3 (Boolean Functions as a Data Type)

Boolean functions can be defined by the number of their arguments together with a function taking a list of boolean values (with the length of the list matching the number of arguments). This representation is given by the type `BoolFun`.

```
data BoolFun = BF { numberArgs::Int, bfun::[Bool]->Bool }
```

Implement a function `isSatisfiable :: BoolFun -> Maybe [Bool]`, that returns (using `Just`) some input for the boolean function where the function returns `True` (if any), and `Nothing`, if there is no such input.

Test `isSatisfiable` using

```
g [x,y] = x && not y
```

i.e., `isSatisfiable (BF { numberArgs=2, bfun=g })` should yield `Just [True,False]`.

Problem 4 (Lists)

Give implementations of the following functions:

(a) `gcdList :: Integral a => [a] -> a`

`gcdList xs` shall return the greatest common divisor of all the numbers in the list `xs`, for example: `gcdList [160,152,-8000,0] == 8`

Hint: Use the predefined function `gcd :: Integral a => a -> a -> a`

(b) `facs :: Integer -> [Integer]`

`facs n` shall return the list of the first `n+1` factorials, i.e., `[0!,1!,...,n!]`.

(c) `genList :: (Integer->a) -> Integer -> [a]`

`genList f n` shall compute a list with `n` elements, namely the function values of `f` at $0, 1, \ldots, n-1$:

`genList f n == [f 0, f 1, ..., f (n-1)]`.

(d) `iterList :: (a -> a) -> a -> Integer -> [a]`

`iterList f x n` shall compute a list of `n` iterated function applications of `f` to `x`:

```
iterList f x 0 == [x]
iterList f x 3 == [x, f x, f (f x), f (f (f x))]
```

(e) `isPrime :: Integer -> Bool` that shall test whether a given number is a prime number. Use list comprehension(s).

Due date: Tuesday, May 09, 2023 at 16:00

Upload your Haskell source codes (`*.hs`) to the `Submissions` folder for this exercise on Stud.IP.