PROGRAMMING LANGUAGE CONCEPTS (COMP 2212)
SEMESTER TWO, 2018

Home

exercise sheet one: refresher on haskell

The aim of this exercise class is to make sure you are up to speed in the Haskell language. We will simply be revising some of the core Haskell features such as polymorphism, higher-order functions, lists and pattern matching. We will also be making use of Haskell I/O features for reading and writing files.

The demonstrators in the lab will help answer any questions you have about the tutorials you are reading or if you have difficulty understanding the solutions to the exercises.

Task One (Easy warm up)

Write an Haskell function called `zipL` that takes a pair of `[Int]` arguments. If the lists are of the same length `zipL` should return a list of lists of Ints - each of which are exactly two elements long. The first element of each list should come from the first input list and the second element of each list should come from the second input list.

For example, `zipL ([1,2,3,4], [5,6,7,8])` should return `[[1,5],[2,6],[3,7],[4,8]]` Remember: begin by writing the type of the function

Write an Haskell function called `unzipL` that takes a list of two elements lists of `Int` values and returns a pair of lists such that

> `unzipL ( zipL ( list1, list2 ) )` returns `( list1, list2 )`

for any `[Int]` values `list1` and `list2` of the same length.

Similarly, check that your solution always returns `list2e` (for any list of two `Int` element lists `list2e`) when evaluating `zipL ( unzipL ( list2e ) )` .

Now, if you haven't already, generalise your solution to work for lists of any type rather than just `Int`.

Task Two (A bit harder)

In Task One, when writing `zipL` we were allowed to assume that the lists were the same length. Rewrite `zipL` for input lists of possibly different lengths. This means that in the output list of lists we may no longer have exclusively two element lists.

For example, `zipL [1,2,3,4] [5,6]` should return `[[1,5],[2,6],[3],[4]]` and `zipL [1,2] [5,6,7,8]` should return `[[1,5],[2,6],[7],[8]]`

Is it possible to write the inverse function `unzipL` in this case?

Task Three (More challenging)

There is no reason to restrict `zipL` from Task Two to just two input lists. Write a function called `multiZipL :: [[a]] -> [[a]]` that accepts a list of lists and produces a "zipped" version of these lists by taking consecutive elements from each of the input lists to form the next output list.

For example,
`multiZipL [[1,2,3],[4,5,6],[7,8,9]]` should return `[[1,4,7],[2,5,8],[3,6,9]]` and

`multiZipL [[1,2,3],[4,5,6],[7,8,9],[10],[11],[12,13,14,15]]` should return
`[[1,4,7,10,11,12],[2,5,8,13],[3,6,9,14],[15]]`

Hint: think about how many lists there will be in the output and use an auxilliary function with an accumulator to build up the output lists.

Task Four

Let's generalise your function from Task Three to use some I/O (you could also use `zipL` from Task Two if you didn't manage to complete Task Three). Write a function `multiZipF` that reads a CSV file containing a list of integers on each line, the function should then zip the lists as per Task Three and write the output to another CSV file.

Make sure your function has suitable error handling built in to it for cases where the file contains junk data (i.e. non integers)

Page maintained by Pawel Sobocinski, Julian Rathke.