Programming Paradigms 2025
Session 5: Recursion

Problems for solving and discussing

Hans Hüttel

7 October 2025

Problems that we will definitely talk about

1. ***(10 minutes)***

   The function reverse appears in the Haskell prelude. It will reverse a list such that e.g. reverse [1,2,3] evaluates to [3,2,1] .

   Now it is your task to define your own version of this function, rev. First try to find out what the type of rev should be and follow the overall approach described in Section 6.6.

2. ***(15 minutes)*** A list $[a_1, a_2, \ldots, a_n]$ is *descending* if $a_1 \geq a_2 \geq \ldots \geq a_n$.

   Write a function descending that will return True if a list is descending and False otherwise. As examples, descending [6,5,5,1] should return True and descending ["plip","pli","ppp"] should return False. What is its type?

3. ***(20 minutes)***

   The function isolate takes a list l and an element x and returns a pair of two new lists (l1 ,l2). The first list l1 is a list that contains all elements in l that are not equal to x. The second list l2 is a list that contains all occurrences of x in l.

   - isolate [4,5,4,6,7,4] 4 evaluates to ([5,6,7],[4,4,4]) .
   - isolate ['g','a','k','a'] 'a' evaluates to (['g','k'], ['a','a']).

   Define isolate in Haskell *without* using fst, snd, head or tail. What should the type of isolate be? ***Major hint:*** Place the recursive call in a let- or where-clause and use pattern matching to find the components in the result of that call.

4. ***(20 minutes)*** The function wrapup is a function that takes a list and returns a list of lists. Each list in this list contains the successive elements from the original list that are identical.

   For instance, wrapup [1,1,1,2,3,3,2] should give us the list [[1,1,1],[2],[3,3],[2]] and wrapup [True,True,False,False,False,True] should give us the list [[True,True],[False,False,False],[True]].

   Define wrapup in Haskell using recursion[1] but *without* using fst, snd, head or tail. ***Major hint:*** Recall the definition of the isolate function from before.

5. ***(15 minutes)***

   A former minister of science and education has decided to get a university degree and is now trying to define a Haskell function triples that takes a list of tuples (each tuple has exactly 3 elements) and converts that list of tuples into a tuple of lists.

   triples [(1,2,3), (4, 5, 6), (7, 8, 9)] should produce ([1,4,7], [2, 5, 8], [3, 6, 9]).

   The minister wrote the following piece of code and a type specification but ran into problems. What seems to be wrong?

   ```
     triples :: Num a => [(a,a,a)] -> ([a],[a],[a])

   triples [] = ()
   triples [(a,b,c)]= ([a],[b],[c])
   triples (x:xs,y:ys,z:zs) = [x,y,z] : Triples [(xs,ys,zs)]
   ```

[1] No list comprehension – that was last week!

Can you fix these issues? How can Section 6.6 help you here?

More problems to solve at your own pace

Here, too, Section 6.6 is helpful.

a) The function rle is a function that, when given a list xs produces a list of pairs of elements of xs and integers[2]. This list of pairs has its elements appears in the order that they appeared originally and contains (x, n) if there are n successive occurrences of x in the list. For instance

```
rle ['a','a','a','g','g','b','a','a']
```

should give us the list `[('a',3),('g',2),('b',1),('a',2)]` and

```
rle [1,1,1,2,2,1,3,3]
```

should give us `[(1,3),(2,2),(1,1),(3,2)]`.

Define rle in Haskell. First try to find out what the type of rle should be and follow the overall approach described in Section 6.6.

b) Define a function amy that will tell us if any elements of a list satisfy a given predicate.

For instance, if

```
odd x = ((x `mod` 2) == 1)
```

then

```
amy odd [2,5,8,3,7,4]
```

should return True, whereas

```
amy odd [2,8,42]
```

should return False.

c) Create a function frequencies that, given a string s, creates a list of pairs `[(x1,f1),....(xk,fk)]` such that if the character `xi` occurs a total number of `fi` times throughout the list s, then the list of pairs will contain the pair `(xi,fi)`.

As an example of this,

```
frequencies "regninger"
```

should return the list

```
[('r',2),('e',2),('g',2),('n',2),('i',1)]
```

First find out what the type of the function should be.

d) A theorem in number theory states that every non-zero real number x can be written as a *continued fraction*. This is a potentially infinite expression of the form

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\cdots \cfrac{1}{a_n}}}+}}} \tag{1}$$

For rational numbers, the a_i's will eventually all be 0, so the continued fraction is finite; for irrational numbers, the continued fraction will be infinite. See e.g. [1] for more.

The goal of this problem is to write a Haskell function cfrac that will, given a real number r and a natural number n, finds the list of the first n numbers in the continued fraction expansion of r. What should the type of cfrac be?

Bibliography

[1] Wikipedia. Continued fractions. `https://en.wikipedia.org/wiki/Continued_fraction`.

[2]This function computes what is called a run-length encoding, thus its name.