

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp
Stephan Swiderski

Exercises *Functional Programming* – Sheet 4

Solutions will be collected until Wednesday, May 9, 2007 in the exercise course.

Exercises can be solved both in English and in German.

Exercise 1 (2+3 points)

In this exercise we want to replace all elements of a given list by the minimum of the list. A straightforward implementation could look like this:

```
replaceByMin :: Ord a => [a] -> [a]
replaceByMin xs = let m = minimum xs in map (\x -> m) xs
```

Unfortunately, this requires two passes over the list. First, `minimum xs` goes through the list and computes the minimum. Then, `map (\x -> m) xs` iterates through the whole list again.

Using lazy programming, it is possible to find the minimum and replace all elements of the list by it in a single iteration over the list.

To this end, implement the following functions in Haskell.

- (a) Write Haskell declarations for the function `findMinAndReplace` which takes a list `xs` and a replacement value `r` and returns a pair containing the list where all elements in `xs` have been replaced by `r` and the minimum of `xs`. Thus, the type of `findMinAndReplace` is `Ord a => [a] -> b -> ([b], a)`.

Your function may only iterate over the list once, i.e., it must compute the minimum and replace the elements at the same time.

- (b) Write Haskell declarations for `replaceByMin` which use `findMinAndReplace` to replace all elements of a list by its minimum in a single iteration.

Hint: Use a local pattern declaration with `findMinAndReplace` on the right.

Exercise 2 (1+4 points)

The two functions `fibs1` and `fibs2` compute the infinite list of Fibonacci numbers starting from 0 and 1, i.e. the list `[0, 1, 1, 2, 3, 5, 8, 13, 21, ...]`. The Fibonacci sequence fib_0, fib_1, \dots is defined by $fib_0 = 0$, $fib_1 = 1$, and $fib_{n+1} = fib_n + fib_{n-1}$.

```
fibs1 = map fib [0..] where
  fib 0 = 0
  fib 1 = 1
  fib (n+2) = fib (n+1) + fib n
```

```
fibs2 = map fib [0..] where
  fib n = fib' n 0 1 where
    fib' n a b | n == 0    = a
               | otherwise = fib' (n-1) (a+b) a
```

- (a) Evaluating the expression `take n fibs1` is of complexity $O(2^n)$, i.e., exponential in n .¹ Analyze the complexity of evaluating the expression `take n fibs2`. To this end, give a lowest upper bounds on its complexity using the O -notation, i.e., give a functions $f(n)$ such that $O(f(n))$ is the respective bound.

Here, you should assume that addition has constant complexity, i.e., evaluating $n + m$ has complexity $O(1)$.

- (b) Implement a function `fibs3` which computes the same list as `fibs1` and `fibs2` by using a cyclic data structure. Analyze the complexity of evaluating `take n fibs3` depending on n by giving a lowest upper bound as in (a).

Exercise 3 (2+1+2 points)

In the following Haskell functions you are not allowed to use any predefined functions except for `(>>)`, `(>>=)`, `return`, `putChar`, and `getChar` as well as arithmetic functions (like `+` and `div`).

- (a) Write Haskell declarations for the IO function `getString :: IO String` which reads a line of input from the console. A line is terminated by the character `'\n'` which stands for ‘new line’. You can test your function with the following expression (which for the input `abc` should print `abccba`):

```
getString >>= \s -> putStr s >> putStr (reverse s)
```

- (b) Write Haskell declarations (including a type declaration) for the IO function `putStrings` which takes an `Int n` and a `String s` and prints n copies of s .
- (c) Write a program in Haskell which asks the user for his name. It then prints out the user’s name surrounded by stars. For example if the user types the name `Peter Schneider-Kamp`, the output would be:

```
*****
* Peter Schneider-Kamp *
*****
```

¹In fact $O(\text{fib}_n)$ is a tighter bound on the complexity of this expression.