

Prof. Dr. Jürgen Giesl
René Thiemann

Exercises *Functional Programming* – Sheet 4

Solutions will be collected until Thursday, Nov 14, 2002 in the exercise course.

Exercise 1 (7 points)

The application of the function `foldr` (\oplus) `e` `xs` yields a term in which the list elements are combined with the operator \oplus in right-associative way.

$$\text{foldr } (\oplus) \text{ e } [a_1, \dots, a_n] = a_1 \oplus (a_2 \oplus \dots (a_n \oplus \text{e}) \dots)$$

- (a) Dual to `foldr` define the function `foldl` where the parentheses associate to the left.

$$\text{foldl } (\oplus) \text{ e } [a_1, \dots, a_n] = (\dots((\text{e} \oplus a_1) \oplus a_2) \dots \oplus a_n)$$

- (b) Give a function `f` such that `foldr f` \neq `foldl f`.
- (c) Define `f` and `e` such that `foldl f e = id` where `id xs = xs` for all `xs :: [a]`
- (d) Define the function `elem :: Eq a => a -> [a] -> Bool` using either `foldl` or `foldr`. Define it in a way so that `elem 5 [1..]` results in `True`. As usual, `elem n xs` returns `True` if `n` is an element of the list `xs`.
- (e) Define the function `map` using `foldr`.
- (f) The function `remdups` removes adjacent duplicates from a list. For example, `remdups [1,2,2,3,3,3,1,1] = [1,2,3,1]`. Define `remdups` using either `foldr` or `foldl`.
- (g) Given a list `xs = [x1, x2, ..., xn]` of numbers, the sequence of successive maxima `ssm xs` is the longest subsequence `[xj1, xj2, ..., xjm]` such that `j1 = 1` and `xjs < xjt` for `js < jt`. For example, the sequence of successive maxima of `[3,1,3,4,9,2,10,7]` is `[3,4,9,10]`. Define `ssm` in terms of `foldl`.

Exercise 2 (6 points)

- (a) Under what conditions on `xs` and `ys` does the following equation hold?

$$[x \mid x \leftarrow xs, y \leftarrow ys] = [x \mid y \leftarrow ys, x \leftarrow xs]$$

- (b) Define a function `pairs` such that `pairs n` is a list of all pairs of distinct integers $1 \leq x, y \leq n$ with $x \neq y$.

- (c) Write a program `triples :: Int -> [(Int,Int,Int)]` such that `triples n` finds all triples (a, b, c) in the range $0 < a, b, c \leq n$ such that $a^2 + b^2 = c^2$.
- (d) How can the function `elem :: Eq a => a -> [a] -> Bool` be defined using a list comprehension and an equality test?
- (e) A natural number n is *perfect* if it is the sum of its divisors that are smaller than n . I. e., 6 is perfect because the divisors of 6 are 1, 2, and 3. Define a list of all perfect numbers.
- (f) Make a list of all natural numbers which are not dividable by any square number.

$$\{n \in \mathbb{N} \mid \forall m > 1. m^2 \nmid n\}$$

Exercise 3 (1 + 2 + 2 + 3* points)

- (a) Define `dither` by

```
dither = yes
      where yes = "Yes." ++ no
            no   = "No!" ++ maybe
            maybe = "Maybe?" ++ yes
```

What will be printed if `dither` is typed in a session? Draw the graph of the cyclic structure that represents this value.

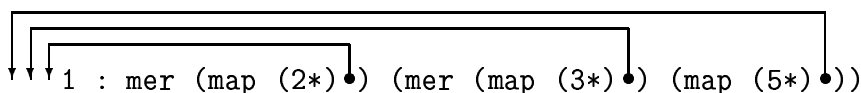
- (b) Given the function

```
mer :: [Int] -> [Int] -> [Int]
mer (x:xs) (y:ys)
  | x < y    = x : mer xs (y:ys)
  | x == y   = x : mer xs ys
  | x > y    = y : mer (x:xs) ys
```

we define `hamming` by

```
hamming :: [Int]
hamming = 1 : mer (map (2*) hamming)
                (mer (map (3*) hamming)
                    (map (5*) hamming))
```

Initially, `hamming` will be represented by the following cyclic structure:



After the first seven elements of `hamming` have been printed, the above structure will have reduced to the following:

```

1:2: 3: 4: 5:6:8: mer ( map (2*) ) (9:mer (map (3*)) ) (10:map (5*))

```

Draw the four cyclic structures that represent hamming after the first 1, 2, 3, and 4 elements have been printed.

(c) There is a predefined function `zipWith` defined as follows:

```

zipWith :: (a->b->c) -> [a] -> [b] -> [c]
zipWith _ [] _ = []
zipWith _ _ [] = []
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys

```

Using the the function `addLists :: Num a => [a] -> [a] -> [a]`, `addLists = zipWith (+)` we can define the infinite lists

```

ones, nats :: [Int]
ones = 1:ones
nats = 1:addLists ones nats

```

Define the list of Fibonacci-numbers `fibs = [0,1,1,2,3,5,8,...]` in the same way. What time-complexity has the function `\n->take n fibs` in your implementation?

Using the function `addLists`, define a function `runningSums :: [Int] -> [Int]` which calculates the running sums `[0, a0, a0 + a1, a0 + a1 + a2, ...]` of a list `[a0, a1, a2, ...]`.

(d)* The data type `Tree` is defined as follows:

```

data Tree a = Node a [Tree a]

```

Define the function `minTree :: Ord a => Tree a -> Tree a` which replaces all values in all nodes with the minimum of all nodes' values in only *one* pass through the tree.

*This is a bonus exercise with bonus points