Functional Programming – Advanced Concepts and Applications
WS 2023/24

Till Mossakowski
Benjamin Junge

4. Excercise sheet

Issued: 2023-11-06
Due: 2023-11-13 & 2023-11-14

$\boxed{4.1}$ *map, foldl, foldr, filter*

Implement the following functions using only `map`, `filter`, `foldr`, `foldl` and other non recursive functions.

(1) calculating the product of a list of integers:

```
product' :: [Int]→ Int
```

e.g:

```
product' [1 .. 5] ⇝ 120
```

(2) testing if all elements of a given integer list are odd.

```
allOdd :: [Int] → Bool
```

e.g:

```
allOdd [5,7] ⇝ True
allOdd [4]   ⇝ False
```

(3) calculating the table of values of the function $x^2 + 3x + 5$ for arguments 0 to 150.

```
xSquaredPlusThreeXPlusFive :: [(Integer, Integer)]
```

e.g:

```
xSquaredPlusThreeXPlusFive !! 1 ⇝ (1, 9)
```

(4) returning all integers that have the same string key.

```
getByKey :: [(String, Int)] → String → [Int]
```

e.g:

```
getByKey [("a",2), ("b",3), ("a",6), ("c",4)] "a" ⇝ [2,6]
```

$\boxed{4.2}$ *Underscores, Types*

(1) The Underscore has a special meaning in haskells function definitions. Given a programming language N of your choice answer the following questions:

- Is _ a legal expression or name in N?
- If it is a legal expression/name: what are the differences in its use compared to the underscore in haskell? If it isn't a legal expression/name: is there a different construct in N that is similar to how _ is used in haskell?
- Discuss the differences of the following two expressions in `ghci`:
  "`let f _ _ = undefined`" and "`let g x x = undefined`".

(2) The prelude of ghci defines the functions `foldl1` and `foldr1`. What are the differences of those two functions to their normal counterparts `foldl` / `foldr` ?

$\boxed{4.3}$ *Streams and higher order functions*

Given `nat = [0 ..]` as the sequence of all natural numbers as a base, construct the following streams by only using `nat`, `map`, `filter`, `fold` and any non recursive functions:
You may use `drop`, `take` and `!!`.

(1) `ev` is the stream of all even natural numbers.
e.g:

```
take 10 ev ⇝ [0,2,4,6,8,10,12,14,16,18]
```

(2) `harmonic` is the stream of all elements of the harmonic sequence $a_n = \frac{1}{n}$.
e.g:

```
take 10 harmonic ⇝
[1.0,0.5,0.3333333333333333,0.25,0.2,0.16666666666666666,0.14285714285714285,
0.125,0.1111111111111111,0.1]
```

(3) `triangle` is the stream of all triangular numbers which are defined as the sequence $a_n = \sum_{i=0}^{n} i$
do not use the closed-form expression for this sequence!
e.g:

```
take 10 triangle ⇝[1,3,6,10,15,21,28,36,45,55]
```

(4) `palin` is the stream of all elements of the palindrome number sequence. A number is an element of this sequence if it stays the same when the order of its digits is inverted.
e.g:

```
take 20 palin ⇝[0,1,2,3,4,5,6,7,8,9,11,22,33,44,55,66,77,88,99,101]
```

4.4 *Advanced folding and filtering*

Using only foldr, filter, map and (.) implement the following two functions:

```
dfold :: (b → c → c) → c → (a → b → b) → b → [[a]] → c
nfilter :: [(a → Bool)] → [a] → [a]
```

dfold f i g j x is a function that first folds the inner lists of x with the function g and initial value j and then folds the outer list with the function f and initial value i.
e.g:

```
dfold (+) 0 (*) 1 [[1,2,3],[1,4,1],[2,2,2,1]] ⇝ 18
```

nfilter p x takes a list of unary predicates a → Bool and uses all predicates to filter x.

```
nfilter [even,odd] [1..100] ⇝ []
nfilter [even,(λx → x > 20),(λx→ x < 40)] [1..100] ⇝ [22,24,26,28,30,32,34,36,38]
```

4.5 **Function family and stream of streams*

Using only nat, map, fold, filter, take (or !!) and other non recursive functions define the following stream:
f_a = [f_0,f_1,..] is the stream of *functions* $f_a(x) = x^2 + x + a$ for all $a \in \mathbb{N}$.
define the stream of streams f_all = [[f_0(0),f_0(1),..],[f_1(0),f_1(1),..]..] which contains all values of all functions of the function family $f_a(x)$ with $x \in \mathbb{N}$.
Implement the function ttake :: a → a → [[a]] → [a] s.t. ttake a b f_all produces the list of the first a elements of the bth stream in f_all.
e.g:

```
ttake 10 0 f_all ⇝ [0,2,6,12,20,30,42,56,72,90]
ttake 10 1 f_all ⇝ [1,3,7,13,21,31,43,57,73,91]
ttake 10 10 f_all ⇝ [10,12,16,22,30,40,52,66,82,100]
```