Technical University of Munich
Chair for Logic and Verification
Prof. Tobias Nipkow, Ph.D.
J. Rädle, L. Stevens, K. Kappelmann, MC Sr Eberl

WS 2020/21
15.01.2020
Deadline: 08.02.2021, 23:59

Functional Programming and Verification

Sheet 12

Tutorial Exercises

Exercise T12.1 Redexes

Identify all redexes in the following `Integer`-expressions. Determine for each redex whether it is innermost, outermost, both, or neither.

1. `1 + (2 * 3)`
2. `(1 + 2) * (2 + 3)`
3. `fst (1 + 2, 2 + 3)`
4. `fst (snd (1, 2 + 3), 4)`
5. `(\x -> 1 + x) (2 * 3)`

Exercise T12.2 Reductions

Evaluate the following expressions according to the evaluation strategy as defined in the lecture (slide with "principles of lazy evaluation"):

```
map (*2) (1 : threes) !! 1
(\f -> \x -> (x + f 2) + x) (\y -> y * 2) (3 + 1)
head (filter (/=3) threes)
```

Which evaluations do not terminate?

The functions used in the expressions above are defined as follows:

```
map _ [] = []
map f (x:xs) = f x : map f xs

filter _ [] = []
filter f (x:xs) | f x = x : filter f xs
                | otherwise = filter f xs

(x:xs) !! n | n == 0 = x
            | otherwise = xs !! (n - 1)

threes = 3 : threes
```

Exercise T12.3 Nooooooonacci

The lecture presented the following implementation of `fib` which produces an infinite list containing all Fibonacci numbers, i.e. `fibs = [0,1,1,2,3,5,8,13,...]`.

```
fibs :: [Integer]
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

Which functions can we use to evaluate the function partially? Discuss what happens if one part of the list is evaluated and later accessed for a second time.

Now, consider an alternative implementation for `fibs`.

```
fibs2 :: [Integer]
fibs2 = map fib [0..]
  where
    fib 0 = 0
    fib 1 = 1
    fib n = fibs2 !! (n - 1) + fibs2 !! (n - 2)
```

Compare the latter implementation with the former one. Which function performs better and how could the slower function be improved?

Why does the following implementation not give raise to a similar sharing behaviour?

```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Since normal Fibonacci numbers are boring, we want to generalise them to n-onacci numbers. We can construct the n-onacci numbers by letting $f_0 = 0, f_1 = 0, \ldots, f_{n-2} = 0, f_{n-1} = 1$ and $f_a = f_{a-n} + f_{a-n+1} + \cdots + f_{a-1}$. Implement the function `nonaccis :: Int -> [Integer]` in two ways:

1. Come up with a function `zipWithN ([a] -> b) -> [[a]] -> [b]` and define `nonaccis` analogously to `fibs`.
2. Use `fibs2` as a template to define `nonaccis`.

Homework

****The FPV-Programming Contest****

We are sorry to say: no, there is no *Wettbewerb* task this week. However, we are organising the *FPV-Programming Contest*!

The contest will be of a similar nature as the ICPCs: teams of 2–3 students try to solve as many programming challenges as possible in a given time frame. The more challenges you solve and the quicker you finish, the better. During the competition, there will be a live scoreboard, listing all teams, their rank, and their solved problems. Needless to say, you will only be allowed to solve the challenges using Haskell.

Not only will the top 30 teams be awarded prestigious *Wettbewerb* points, but you will also collect a coin if you manage to solve at least a minimum number of tasks. Besides, it is a nice way to see how much you progressed this semester and to have a fun evening with your team mates! :)

The contest will take place on the late afternoon/evening of the 10th of February. Please vote for your preferred timeslot on Zulip: here. You can sign up by sending an e-mail to fpv@in.tum.de (click the link for a pre-filled e-mail) using the subject "contest registration" and the following e-mail body format:

```
{"teamName": "teamname123","username":"lrzId1"}
{"teamName": "teamname123","username":"lrzId2"}
{"teamName": "teamname123","username":"lrzId3"}
```

where you need to replace `teamname123` with your team's name (lowercase letters and digits only!) and the lrzIds by your team member's LRZ-IDs (e.g. "ga00aa"). If you participate in a team of 2, simply remove on line entry. If you are looking for a team, you can do so in this thread on Zulip. See you at the contest!

You need to collect 4 out of 6 points (P) to collect a coin.

Exercise H12.1 To Infinity And Beyond! [1+1+1+1P]

For the following exercises, keep in mind that in Haskell `xs ++ ys = xs` if `xs` is infinite.

a) Implement a function `allBinaries :: [String]` that returns the infinite, ordered list of all binary numbers, least significant bit first, no trailing zeros, i.e. `allBinaries = ["0","1","01","11","001",...]`.

b) Implement a function `elements :: [[a]] -> [a]` that returns all elements occuring in the possibly infinite list of possibly infinite lists. Here are some valid examples:

```
elements [[1,2,3],[4,2]] = [1,2,3,4,2]
elements [[1,2,3],[4,2]] = [1,2,4,3,2]
elements [[1,2,3],[4,2]] /= [1,2,3,4]
elements [[1..],[0]] /= [1..] ++ [0]
```

Hint: follow an approach as done in Cantor's proof of the countability of the rationals (cf. this picture).

c) We define the types

```
data Tree = Node Tree Tree | Leaf
data Direction = L | R -- left and right
type Path = [Direction]
```

Write a function `allFinitePaths :: Tree -> [Path]` that takes a possibly infinite binary tree `t :: Tree` and returns a list of all finite paths from the root to any leaf of `t`.

d) We define a type of propositional formulas as follows:

```
type Var = Integer
data Formula = V Var | Formula :&: Formula | Not Formula
```

Now, given a possibly infinite list of variables `vs`, we are interested in the possibly infinite list of all finite formulas containing variables in `vs`. Write a function `allFormulas :: [Var] -> [Formula]` that is doing precisely that.

Hint: Make use of `elements`.

Exercise H12.2 Stonks only go up! [2P]

Your task is to write a `main` function that determines the average price of a specific stock in a given time range. You are first given the name of the stock and, separated by a space, the inclusive time range as a tuple, e.g. `BB (0,5)`.

Then, a number of ticker prices on separate lines in the format `GME,5,319` where `5` is the timestamp and `319` is the price at that time. Read the prices from stdin until the user enters `quit`. You should then compute the average price of the specified stock over all timestamps in the range you were given in the beginning and print it to stdout. Use whole number division (`div`) to compute the average. If there are no prices in the given range, you should print `0`. You may assume that the input is well-formed.

Here is an example interaction with the program (Lines starting with `>>` are inputs from the user):

```
>> BB (0, 5)
>> TSLA,1,5
>> BB,1,10
>> GME,1,20
>> BB,2,21
>> GME,3,10
>> BB,7,30
>> quit
15
```

Note: the timing of this exercise may seem awfully convenient but it is really an old exam exercise.

> I am incapable of conceiving infinity, and yet I do not accept finity.
> — Simone de Beauvoir