My Report

Me

Monday 17th January, 2022

Abstract

We give a toy example of a report in *literate programming* style. The main advantage of this is that source code and documentation can be written and presented next to each other. We use the listings package to typeset Haskell source code nicely.

Contents

1 How to use this?

To generate the PDF, open `report.tex` in your favorite LaTeXeditor and compile. Alternatively, you can manually do `pdflatex report; bibtex report; pdflatex report; pdflatex report` in a terminal.

You should have stack installed (see `https://haskellstack.org/`) and open a terminal in the same folder.

- To compile everything: `stack build`.
- To open ghci and play with your code: `stack ghci`
- To run the executable from Section 3: `stack build && stack exec myprogram`
- To run the tests from Section 4: `stack clean && stack test --coverage`

2 The most basic library

This section describes a module which we will import later on.

```
module Basics where

import Control.Monad
import System.Random

thenumbers :: [Integer]
thenumbers = [1..]

somenumbers :: [Integer]
somenumbers = take 10 thenumbers

randomnumbers :: IO [Integer]
randomnumbers = replicateM 10 $ randomRIO (0,10)
```

We can interrupt the code anywhere we want.

```
funnyfunction :: Integer -> Integer
funnyfunction 0 = 42
```

Even in between cases, like here. It's always good to cite something [Knu11].

```
funnyfunction n | even n    = funnyfunction (n-1)
                | otherwise = n*100
```

Something to reverse lists.

```
myreverse :: [a] -> [a]
myreverse [] = []
myreverse (x:xs) = myreverse xs ++ [x]
```

That's it, for now.

3 Wrapping it up in an exectuable

We will now use the library form Section 2 in a program.

```
module Main where

import Basics

main :: IO ()
main = do
  putStrLn "Hello!"
  print somenumbers
  print (map funnyfunction somenumbers)
  myrandomnumbers <- randomnumbers
  print myrandomnumbers
  print (map funnyfunction myrandomnumbers)
  putStrLn "GoodBye"
```

The output of the program is something like this:

```
Hello!
[1,2,3,4,5,6,7,8,9,10]
[100,100,300,300,500,500,700,700,900,900]
[1,3,0,1,1,2,8,0,6,4]
[100,300,42,100,100,100,700,42,500,300]
GoodBye
```

Note that the above `showCode` block is only shown, but it gets ignored by the Haskell compiler.

4 Simple Tests

We now use the library QuickCheck to randomly generate input for our functions and test some properties.

```
module Main where

import Basics

import Test.Hspec
import Test.QuickCheck
```

The following uses the HSpec library to define different tests. Note that the first test is a specific test with fixed inputs. The second and third test use QuickCheck.

```
main :: IO ()
main = hspec $ do
  describe "Basics" $ do
    it "somenumbers should be the same as [1..10]" $
      somenumbers `shouldBe` [1..10]
    it "funnyfunction: result is within [1..100]" $
      property (\n -> funnyfunction n `elem` [1..100])
    it "myreverse: using it twice gives back the same list" $
      property $ \str -> myreverse (myreverse str) == (str::String)
```

To run the tests, use `stack test`.

To also find out which part of your program is actually used for these tests, run `stack clean && stack test`
Then look for "The coverage report for ... is available at ... .html" and open this file in your

browser. See also: `https://wiki.haskell.org/Haskell_program_coverage`.

5 Optional: Profiling

The GHC compiler comes with a profiling system to keep track of which functions are executed how often and how much time and memory they take. To activate this RTS, we compile and execute our program as follows:

```
stack clean
stack build --profile
stack exec --profile myprogram -- +RTS -p
```

Results are saved in the file `myprogram.prof` which looks as follows. Note for example, that funnyfunction was called on 14 entries.

```
                                                     individual      inherited
COST CENTRE       MODULE          no. entries  %time %alloc   %time %alloc

MAIN              MAIN            227       0    0.0    0.8     0.0  100.0
 main             Main            455       0    0.0   29.6     0.0   38.6
  funnyfunction   Basics          518      14    0.0    0.6     0.0    0.6
  randomnumbers   Basics          464       0    0.0    0.3     0.0    8.4
   randomRIO      System.Random   468       0    0.0    0.0     0.0    8.0
   ...
```

For many more RTS options, see the GHC documentation online at `https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/profiling.html`.

6 Conclusion

Finally, we can see that [LW13] is a nice paper.

References

[Knu11] Donald E. Knuth. *The Art of Computer Programming. Combinatorial Algorithms, Part 1*, volume 4A. Addison-Wesley Professional, 2011.

[LW13] Fenrong Liu and Yanjing Wang. Reasoning about agent types and the hardest logic puzzle ever. *Minds and Machines*, 23(1):123–161, 2013.