Till Mossakowski
Benjamin Junge

8. Excercise sheet

Issued: 2023-12-04
Due: 2023-12-11 & 2023-12-12

8.1 *To do or not to do*

The lecture introduced the do-notation as syntactic sugar for using the operators »= and ». Given the following code in do-notation, write the corresponding version of the code that uses »= and » instead.

1.
```
sequence (x:xs) = do
 y ← x
 ys ← sequence xs
 return y:ys
```

2.
```
travFS :: (FilePath → IO ()) → FilePath → IO ()
travFS action p = catch
 (do cs ← getDirectoryContents p
     let cp = map (p </>) (cs \\ ["." , ".."])
     dirs ← filterM doesDirectoryExist cp
     files ← filterM doesFileExist cp
     mapM_ action files
     mapM_ (travFS action) dirs)
(λe → putStrLn ( "ERROR: "++ show ) (e :: IOError))
```

8.2 *Better arbitrary for sorted list*

As seen before the standard generators of QuickCheck might not generate sufficient data. To fix this problem we can define our own generators for our custom types by instantiating the class Arbitrary.
Write a generator for the custom type:

```
data SortedList a = SortedList [a]
instance Arbitrary a ⇒ Arbitrary (SortedList a) where
        arbitrary = ??
```

Test your arbitrary generator by doing a quickCheck with the axioms from 7.2. You may also sample your generator by using `sample (arbitrary :: Gen (SortedList X))` where X is a concrete type with standard generator like String, Int or Bool.

8.3 *Arbitrary generator for distinct pairs*

Consider the type **data** `Distinct a = Distinct (a,a)` which represents pairs of values s.t. for all `Distinct (a,b)` it holds that $a \neq b$.
Write a generator for this custom type s.t. the distinctness property holds.

```
instance (Arbitrary a,Eq a) ⇒ Arbitrary (Distinct a) where
        arbitrary = ??
```

Sample your generator and write/execute a QuickCheck axiom to test the distinctness property. *If sample always starts with a value like Distinct(0,0) you may consider it not breaking the distinctness property.*

8.4 *File manipulation*

Write a function `alter :: IO ()` that has the following behaviour:
It first asks for a file name and waits for the users input.
If the file doesn't exist the function will exit by printing "File doesn't exist!".
If the file exists it will read the contents, format every character to upper case and reverses every word of the input. Then it will write the altered contents into a file called 'U.'X, where X is the original file name.
Test your function accordingly. *hint: import System.Directory might be useful.*

8.5 **Rolling Dice*

While QuickChecks arbitrary can be abused to generate random numbers it is rather clunky. A better alternative is System.Random. Write the module DiceRoller which utilizes System.Random with the following functions:

```
roll    :: Int → IO Int
mroll   :: Int → Int → IO ()
rollLoop  :: IO ()
mrollLoop  :: IO ()
```

`roll t` rolls a t sided die. (e.g. the result of roll 5 is one of [1,2,3,4,5])
`mroll t c` rolls a t sided die c times and prints it formatted to the screen.
`rollLoop` is a function that takes a user input (Integer) and rolls the corresponding die, printing the result and waiting for the next input. Input 'q' ends the loop.
`mrollLoop` is similarly a function that takes a die size t and a times value c from the user (one after the other) and uses mroll t c to print the results. After that it will wait for the next input. Input 'q' ends the loop.

e.g:

```
*DiceRoller> mroll 10 5
8, 8, 10, 7, 6,
Done rolling

*DiceRoller> rollLoop
What kind of die should be rolled?(q for quit)
29
Rolled: 6
What kind of die should be rolled?(q for quit)
29
Rolled: 25
What kind of die should be rolled?(q for quit)
q
Done

*DiceRoller> mrollLoop
What kind of die should be rolled?(q for quit)
2
How many times should the die be rolled?
20
2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2,
Done rolling
What kind of die should be rolled?(q for quit)
4
How many times should the die be rolled?
9
3, 1, 1, 3, 2, 2, 4, 3, 2,
Done rolling
What kind of die should be rolled?(q for quit)
q
Ended rolling dice.
```