Homework 7

March 31, 2020

1 Find files with state monad

You should implement a find function similar in purpose to the find functions in Lecture 11/12 to find files that satisify certain boolean conditions. Instead of specifying a predicate to control recursion and a predicate to filter files, you will implement a find function that folds the files using file states.

Intuitively, you can think of a file state as a function `s -> (FileAction, s)` where *s* is the accumulation value. A file state takes a prior accumulation value and returns a new accumulation value and a file action that tells the find function whether to stop, skip a directory, or continue.

```
data FileAction = Done | Skip | Continue
```

Specifically, we define the file state type as a State monad that is stacked on top of IO monad and yields a file action.

```
type FileState s = StateT s IO FileAction
```

The find function will then take as argument of a getState function and a directory path and return a file state.

```
find :: (FileInfo -> FileState s)  -- get file state from file info
     -> FilePath                   -- entry path to find files
     -> FileState s                -- resulting file state

find getState path = ...
```

2 Requirement

The first parameter of the `find` function is a `getState` function that returns a file state that decides how to fold the file information, whether to skip a directory, and whether to stop.

For example, we can define a `getState` function to save the path, size, and the last modification time of a file in a list if the file is a PDF larger than 4 MB or it is executable. The function also returns a stop action if 10 files have been found and it returns a skip action if the path is modified 2018 or earlier.

The `find` function does not need to access the accumulation value of the file state. It only needs to use the action wrapped in file state to determine whether to stop and whether to recursively find files in a directory.

3 Testing

You can test the implementation using the following `main`

```
main = do
   let downloads = "C:\\Users\\tzhao\\Downloads"

   let yearP = ((\(x,_,_) -> x) . toGregorian . utctDay) <$> timeP
   let recurseP = yearP >? 2018

   let filterP = (extP ==? ".pdf" &&? sizeP >? 2^22) ||? execP

   let iter :: FileInfo -> FileState [(FilePath, Integer, UTCTime)]
       iter info = do
            let eval = \p -> runFileP p info

            s <- get
            let r = (eval pathP, eval sizeP, eval timeP)

            let s' = if eval filterP then r : s else s
            put s'

            return if length s' >= 10 then Done
                   else if eval recurseP then Continue else Skip

   results <- execStateT (find iter downloads) []

   forM_ results $ \(p,n,t) ->
                          handle (\(SomeException _) -> return ())
                        $ putStrLn
                        $ (take 100 p) ++ "\t"
                           ++ show (round $ fromInteger n / 2^10) ++ " KB\t"
                           ++ show (utctDay t)
```

This main function is included the provided template file. A similar function called ‘foldTree’ is in Chapter 9 of *Real World Haskell*, You can study it for reference. The control flow of this find function is similar. What is different is our use of State monad requires no explicit extraction of accumulation value.

4 Submission

Please write your solution in a file – `hwk7.hs` and submit it to the dropbox.