Homework 9

April 18, 2020

1 Race asynchronous actions

In this homework, you will implement a function `finish` that runs a list of IO actions concurrently but only allows the first n actions complete while cancelling the rest. The `finish` function communicates with these actions through a `canceller` IO monad that is passed to each action.

```
finish :: [IO () -> IO b]  -- a list of actions, each takes a canceller IO monad
       -> Int              -- the number of actions that are allowed to complete
       -> IO [b]           -- output messages of all actions
```

2 Requirement

The primary goal of this function is to run tests shown in the next section, where we download documents from 4 URLs and save them to files. However, we only want to download the 2 documents that return first. The `request` function below takes a monad `canceller :: IO ()` and runs it after completing download but before saving the result to file.

The `canceller` action will send a message to `finish` function, which will count how many cancel messages it has received. If the number of cancel messages has reached n, then it will cancel the rest of the actions. Otherwise, the `finish` function will allow `canceller` monad to complete and the `request` function will proceed to save result to file.

You may find it easier to use `async` library.

3 Testing

You can test the implementation using the following `main`

```
import Control.Concurrent
import Control.Concurrent.Async
import Control.Exception (catch, displayException, SomeException(..))
import qualified Data.ByteString.Lazy as L
import Network.HTTP.Conduit
```

```
finish :: [IO () -> IO b] -> Int -> IO [b]
-- finish actions n =

main = do
  let urls = [("http://www.yahoo.com/", "test1.txt"),
              ("http://www.google.com/", "test2.txt"),
              ("http://www.msn.com/", "test3.txt"),
              ("http://www.bing.com/", "test4.txt")]

  let actions = map request urls
  ms <- finish actions 2
  mapM_ putStrLn ms
 where
    request :: (String, String) -> IO () -> IO String
    request (url, f) canceller = do
         d <- simpleHttp url
         canceller
         L.writeFile f d
         return $ url ++ " done"
         `catch` handler
         where handler = \(SomeException e) -> return (url ++ ": " ++ displayException e)
```

The output looks like the following:

```
http://www.yahoo.com/: AsyncCancelled
http://www.google.com/ done
http://www.msn.com/: AsyncCancelled
http://www.bing.com/ done
```

In this test run, the http requests to google and bing complete first so that the responses are saved in the files "test1.txt" and "test3.txt".

The output message `AsyncCancelled` is due to the cancellation of threads created with `async`. If you use `forkIO` to launch threads, the exception message for killed thread will be different.

4 Submission

Please write your solution in a file – `hwk9.hs` and submit it to the dropbox.