Homework 10

April 23, 2020

1 Race asynchronous actions with STM

This homework is the same as the previous one except **you will implement it using STM**.

Recall that in homework 9, you 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.

3 Sample solution for homework 9

Below is the sample solution to homework 9, where I use two MVars `recv` and `send` to synchronize the 'request' threads with 'finish' thread. When a 'request' thread completes downloading a file, it puts its id in `recv` so that 'finish' thread can add it to a list. If the length of the list is less than n, 'finish' thread takes another id from `recv` and puts () in `send` so that the 'request' thread can

continue and save the downloaded file. When the length of the id list reaches n, 'finish' cancels 'request' threads whose ids are not in the list.

```
import Control.Monad (unless)

finish :: [IO () -> IO b] -> Int -> IO [b]
finish actions n =
  do recv <- newEmptyMVar
     send <- newEmptyMVar
     let runAction (f, id) = async $ f $ putMVar recv id >> takeMVar send
     threads <- mapM runAction $ zip actions [1..]

     let g ids = if length ids < n
                 then do id <- takeMVar recv
                         putMVar send ()
                         g (id:ids)
                 else mapM_ f $ zip threads [1..]
                    where f (t, id) = unless (id `elem` lst) $ cancel t
     g []
     mapM wait threads
```

4 Testing

You can test the implementation using the following `main`

```
import Control.Monad
import Control.Concurrent.Async
import Control.Concurrent.STM
import Control.Concurrent.STM.TVar
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.

5 Submission

Please write your solution in a file – `hwk10.hs` and submit it to the dropbox.