Real-world Functional Programming

Coursework Part II Report

14274056 Junsong Yang (psyjy3)

December 9, 2019

1 Task II.1

```haskell
import Control.Monad
import Control.Concurrent
import Control.Concurrent.STM
import System.Random

data Fork = MkFork (TVar Bool)

newInfoBuf :: IO (TChan String)
newInfoBuf = newTChanIO

newFork :: IO Fork
newFork = do
  fork <- newTVarIO False
  return (MkFork fork)

takeForks :: Fork -> Fork -> STM ()
takeForks (MkFork l) (MkFork r) = do
  isUsedL <- readTVar l
  isUsedR <- readTVar r
  if isUsedL || isUsedR then retry
  else do writeTVar l True
          writeTVar r True

putForks ::  Fork -> Fork -> STM ()
putForks (MkFork l) (MkFork r) = do
  writeTVar l False
  writeTVar r False

hungry :: String -> String
hungry name = name ++ " is hungry."

eating :: String -> String
eating name = name ++ " is eating."

thinking :: String -> String
thinking name =  name ++ " is thinking."

philosophers :: [String]
philosophers = ["Aristotle", "Kant", "Spinoza", "Marx", "Russel"]


randomDelay :: IO ()
randomDelay = do
  waitTime <- randomRIO (1,3)
  threadDelay (waitTime * 1000000)

putBuf :: TChan String -> String -> STM ()
putBuf buf str = writeTChan buf str
```


Figure 1: Dinning Philosopher Part I

```haskell
thinking :: String -> String
thinking name =  name ++ " is thinking."

philosophers :: [String]
philosophers = ["Aristotle", "Kant", "Spinoza", "Marx", "Russel"]


randomDelay :: IO ()
randomDelay = do
  waitTime <- randomRIO (1,3)
  threadDelay (waitTime * 1000000)

putBuf :: TChan String -> String -> STM ()
putBuf buf str = writeTChan buf str

getBuf :: TChan String -> STM String
getBuf buf = do
  str <- readTChan buf
  return str

printBuf :: TChan String -> IO ()
printBuf buf = do
  str <- atomically $ getBuf buf
  putStrLn str
  printBuf buf


dinning :: TChan String -> String -> (Fork, Fork) -> IO ()
dinning buf name (left, right) = forever $ do
  atomically $ putBuf buf (hungry name)
  atomically $ takeForks left right
  atomically $ putBuf buf (eating name)
  randomDelay
  atomically $ putForks left right
  atomically $ putBuf buf (thinking name)
  randomDelay

main = do
  forks <- replicateM 5 newFork
  infoBuf <- newInfoBuf
  let dinningPhil     = map (dinning infoBuf) philosophers
      forkPairs       = zip forks (tail . cycle $ forks)
      withForks = zipWith ($) dinningPhil forkPairs
  mapM_ forkIO withForks
  printBuf infoBuf
~
~
1 - 1.9k dinningPhilosophers.hs  Haskell  Git-master  unix | 80: 0  Bottom
```

Figure 2: Dinning Philosopher Part II

2 Task II.2

Figure 3: newtype Bounded

(a) Recursive Statistics for newtype

(b) Statistics using foldMap for newtype

Figure 4: TaskI.5 3