Real-world Functional Programming

Coursework Part II Report

14274056 Junsong Yang (psyjy3)

December 10, 2019

1 Task II.1

```haskell
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."

```

(a) Dinning Phhilosopher Part I

```haskell
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
```

(b) Dinning Phhilosopher Part II

Figure 1: Dinning Phhilosopher Part I

Figure 1 contains two figures that show the full implementation of dinning philosopher using STM. Starting with figure 1 (a), the Fork type is defined with a constructor called MkFork which take a TVar with a Bool as a input. If the boolean inside the TVar is false then this fork is available to be used otherwise this fork is already taken. The newInfoBuf function will return a TChan with String wrapped in IO monad to be used later to store logs that indicate the running state of each thread. The newFork function will return a Fork wrapped in IO monad with the boolean value set to false.

The next two function takeForks and putForks are related to require and release resources. The takeForks function

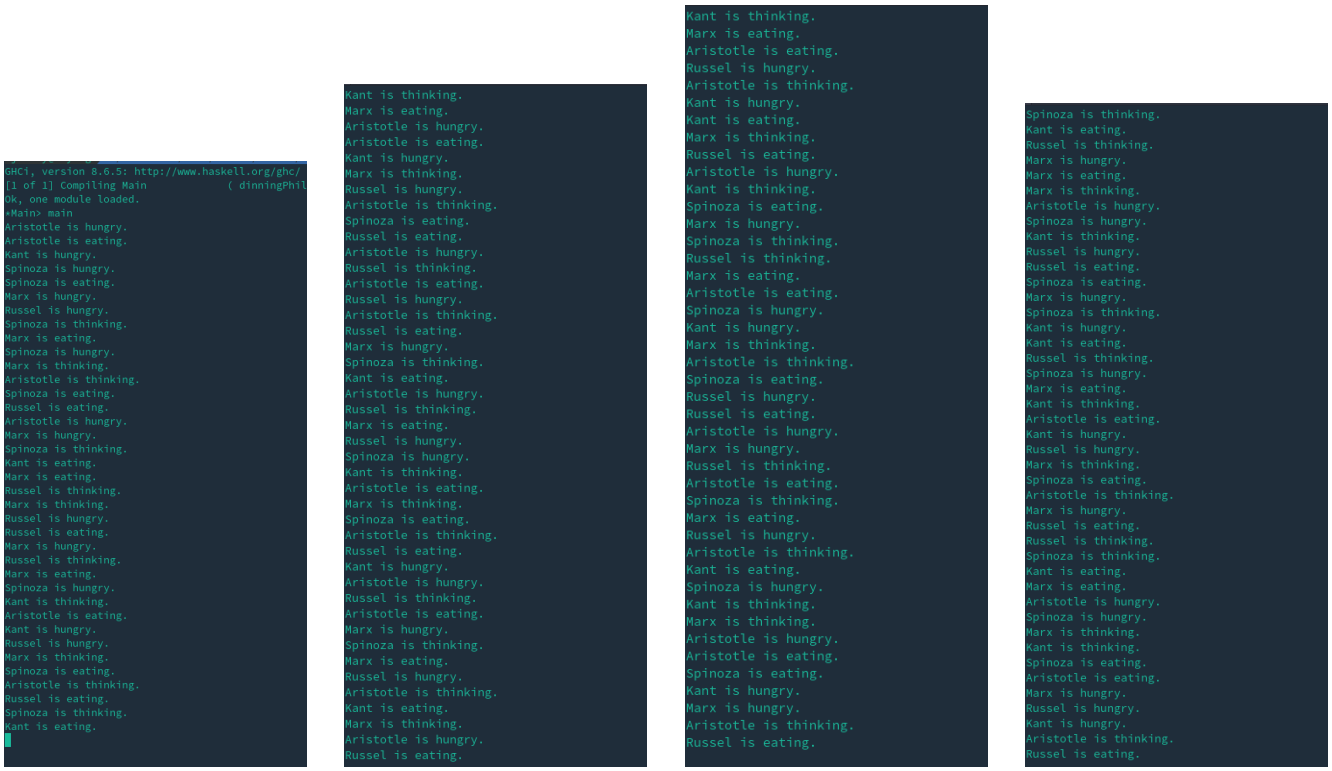
takes two Fork as input. This function will first check if the two Forks are both available. The two Forks can only be used if they are both available as the same time. Otherwise, this function will keep retrying untill both Forks can be required. The putForks function is simple just release the two Forks by setting the boolean to true.

The next three functoins hungry, eating, thinking are just dummy functoin that concatenate the name of philosopher with corresponding information. These will later be put into the infoBuf(TChan String). The names of philosophers are defined as a list of string as figure 1 (b) shows.

The randomDelay will call threadDelay to delay the running thread randomly from 1 to 3 seconds. The next three functions, putBuf, getBuf printBuf are operations related to the infoBuf(TChan String) that is used to store the logs for running thread. The putBuf function will just store string to the infoBuf(TChan String), the getBuf function will return the string stored in TChan and wrap in STM monad. The printBuf will just print the string stored in TChan.

The dinning function is contains the implementation of dinning philosophers. This function takes a TChan String(used to store logs), a string(indicates philosopher's names), and a pair of Fork as input. This function will first store a log in the TChan that indicates the philosopher is hungry. Then, this function will trying to aquire the Fork using takeForks function. If the forks are aquired successfully, another log will be stored in the TChan that suggests the philosopher is eating. Followed by a random delay from 1 to 3 seconds, the forks qill be released using putForks function and corresponding log will be put into the TChan. Finally, the function ended with another random delay.

The main function will first initialise 5 Forks using newFork function and a infoBuf using newInfoBuf functions. Then the philosophers' name will be bundled with the dinning function using map. Then, a infinite list of pair of forks will be generated such that each fork in the pair is distinct from the other. Followed by coupling pairs of forks with the dinning function, a list of runnable functions is made. Finally, the main function will run those function by mapping forkIO function to the runnable dinning functions while the printBuf function is called to print the logs of those running thread.



(a) running at 10s (b) running at 1m (c) running at 2m (b) running at 4m

Figure 2: Dinning Philosopher Running at Defferent Point of Time

Figure 2 contains four figures that shows this implementation running non-stop for four minutes. These sample output of the running program suggests that this implementation is working and without deadlocks.

2 Task II.2

Figure 3: newtype Bounded

(a) Recursive Statistics for newtype (b) Statistics using foldMap for newtype

Figure 4: TaskI.5 3