

Real-world Functional Programming

Coursework Part II Report

14274056 Junsong Yang (psyjy3)

December 10, 2019

1 Task II.1

```
6 data Fork = MkFork (TVar Bool)
7
8 newInfoBuf :: IO (TChan String)
9 newInfoBuf = newTChanIO
10
11 newFork :: IO Fork
12 newFork = do
13   fork <- newTVarIO False
14   return (MkFork fork)
15
16 takeForks :: Fork -> Fork -> STM ()
17 takeForks (MkFork l) (MkFork r) = do
18   isUsedL <- readTVar l
19   isUsedR <- readTVar r
20   if isUsedL || isUsedR then retry
21   else do writeTVar l True
22           writeTVar r True
23
24 putForks :: Fork -> Fork -> STM ()
25 putForks (MkFork l) (MkFork r) = do
26   writeTVar l False
27   writeTVar r False
28
29 hungry :: String -> String
30 hungry name = name ++ " is hungry."
31
32 eating :: String -> String
33 eating name = name ++ " is eating."
34
35 thinking :: String -> String
36 thinking name = name ++ " is thinking."
37
```

(a) Dinning Phhilosopher Part I

```
38 philosophers :: [String]
39 philosophers = ["Aristotle", "Kant", "Spinoza", "Marx", "Russel"]
40
41
42 randomDelay :: IO ()
43 randomDelay = do
44   waitTime <- randomRIO (1,3)
45   threadDelay (waitTime * 1000000)
46
47 putBuf :: TChan String -> String -> STM ()
48 putBuf buf str = writeTChan buf str
49
50 getBuf :: TChan String -> STM String
51 getBuf buf = do
52   str <- readTChan buf
53   return str
54
55 printBuf :: TChan String -> IO ()
56 printBuf buf = do
57   str <- atomically $ getBuf buf
58   putStrLn str
59   printBuf buf
60
61
62 dining :: TChan String -> String -> (Fork, Fork) -> IO ()
63 dining buf name (left, right) = forever $ do
64   atomically $ putBuf buf (hungry name)
65   atomically $ takeForks left right
66   atomically $ putBuf buf (eating name)
67   randomDelay
68   atomically $ putForks left right
69   atomically $ putBuf buf (thinking name)
70   randomDelay
71
72 main = do
73   forks <- replicateM 5 newFork
74   infoBuf <- newInfoBuf
75   let diningPhil = map (dining infoBuf) philosophers
76       forkPairs = zip forks (tail . cycle $ forks)
77       withForks = zipWith ($) diningPhil forkPairs
78   mapM_ forkIO withForks
79   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 at the same time. Otherwise, this function will keep retrying until both Forks can be required. The putForks function is simple just release the two Forks by setting the boolean to true.

The next three functions hungry, eating, thinking are just dummy functions 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 strings 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 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 try to acquire the Fork using takeForks function. If the forks are acquired 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 will be released using putForks function and corresponding log will be put into the TChan. Finally, the function ends with another random delay.

The main function will first initialise 5 Forks using newFork function and an infoBuf using newInfoBuf functions. Then the philosophers' names will be bundled with the dinning function using map. Then, an infinite list of pairs 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 functions by mapping forkIO function to the runnable dinning functions while the printBuf function is called to print the logs of those running threads.

2 Task II.2

Figure 2: newtype Bounded

(a) Recursive Statistics for newtype

(b) Statistics using foldMap for newtype

Figure 3: TaskI.5 3