ECS713P Functional Programming

Chandrima Mukherjee(220990378)

INTRODUCTION:

This paper describes the concurrent operations and thread-based features of a Haskell application.

COMPILING AND RUNNING THE APP:

We need the following commands to run the app

- stack setup
- stack build
- stack run

REQUIRED FUNCTIONALITIES:

This programme creates 10 threads for 10 individual users and keeps track of all the messages they have ever received. A hundred messages are simulated by the application, and at random intervals, random users send random messages from a list of customised messages of each users.

ISSUES FACED AND SOLUTION:

When calculating the total number of messages received, the first output was of type (double). We all know that there cannot be 1.5, 2.5, or more messages transmitted or received. Messages can only be delivered or received in an integer number. It had to be transformed to an integer in order to determine the maximum and show it in the best way possible. Let tuser1count = round user1count was used to accomplish that. Int, another method is to alter the messages in "Types.hs" Although MVar Integer can save a lot of time and space, it isn't universal because it won't always be helpful if a programme requires banking activities or computations.

DESIGN CHOICE JUSTIFICATION:

A relatively small number of modules—Main.hs, MessagesSent.hs, and Types.hs—have been included in order to keep the code as straightforward as possible. I originally kept an MVar of type (Rational) - double thinking that it could be generalised for any kind of application, but if we only look at this system, it is better to use an MVar of type (Int) to store the number of messages as it would mean that only one transaction could take place across all the threads, ensuring no interference with one another. The application made frequent use of MVars, and it was this feature that let us figure out how many messages each user actually receives. It became clear when randomising the receiving user that we needed to make the User datatype an instance of the Eq class to enable direct comparison in order to determine whether the random user is the same as the current user or not.

HADDOCK DOCUMENTATION:

We generate haddock documentation for this project using-

```
stack exec -- haddock --html app/Main.hs src/MessagesSent.hs src/Types.hs --hyperlinked-source --
odir=dist/docs
```

EXTRA FUNCTIONALITIES:

An additional feature has been developed to determine the maximum number of messages delivered and to retrieve user information for the user who has received the most messages. We can determine which user has been using the social media platform the longest with the help of this function. The Output has additionally included information about the transaction. Every user is allocated a list of customised messages that are chosen at random before they transmit it to another user.

GITHUB URL:

https://github.com/chandrima0503/haskell-social-media