CS331: Assignment #2

Due on Wednesday, January 18, 2022

Siddharth Charan 190101085

Assignment: Introduction

In this assignment we were asked to do 3 tasks. 1st one was to watch videos. Second one was to write a program. And 3rd one was to solve exercise of chapter 3 of book "Programming in Haskell". There are 5 questions in exercise, which are discussed below. And codes are attached in separate file.

Question 3.1: What are the types of the following values?

(a) ['a','b','c'] :: [Char]
(b) ('a','b','c') :: (Char, Char, Char)
(c) [(False,'O'),(True,'1')] :: [(Bool, Char)]
(d) ([False,True],['0','1']) :: ([Bool], [Char])
(e) [tail, init, reverse] :: [[a] -> [a]]

Question 3.2: Write down definitions that have the following types; it does not matter what the definitions actually do as long as they are type correct.

(a) bools :: [Bool]
(b) nums :: [[Int]]
(c) add :: Int -> Int -> Int -> Int
(d) copy :: a -> (a,a)
(e) apply :: (a -> b) -> a -> b
Answers :
(a) bools = [True, True, False]
(b) nums = [[a],[b]] where {a::Int;a = 3;b::Int;b = 1;}
(c) add a b c = a + b + c + temp where {temp::Int; temp =0;}
(d) copy t = (t, t)
(e) apply f a = f a

Question 3.3: What are the types of the following functions?

(a) second xs = head (tail xs) :: [a] -> a
(b) swap (x,y) = (y,x) :: (a,b) -> (b,a)
(c) pair x y = (x,y) :: a -> b -> (a,b)
(d) double x = x*2 :: Num a :: a -> a
(e) palindrome xs = reverse xs == xs :: [a] -> Bool
(f) twice f x = f (f x) :: (a -> a) -> a -> a

Question 3.4: Check your answers to the preceding three questions using GHCi

I am attaching screenshots of compiler to show that, my answers are actually correct:

For Question 1:

```
C:\Cs331_Assignments\Assignment2>ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :t ['a','b','c']
['a','b','c'] :: [Char]
ghci> :t ('a','b','c')
('a','b','c') :: (Char, Char, Char)
ghci> :t [(False,'0'),(True,'1')]
[(False,'0'),(True,'1')] :: [(Bool, Char)]
ghci> :t ([False,True],['0','1'])
([False,True],['0','1']) :: ([Bool], [Char])
ghci> :t [tail,init,reverse]
[tail,init,reverse] :: [[a] -> [a]]
ghci>
```

For Question 2:

```
C:\Users\SIDDHARTH CHARAN>ghci
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> bools = [True, True, False]
ghci> :t bools
bools :: [Bool]
ghci> nums = [[a],[b]] where {a::Int;a = 3;b::Int;b = 1;}
ghci> :t nums
nums :: [[Int]]
ghci> add a b c = a+b+c+ temp where {temp::Int;temp = 0;}
ghci> :t add
add :: Int -> Int -> Int -> Int
ghci> copy t = (t,t)
ghci> :t copy
copy :: b -> (b, b)
ghci> apply f a = f a
ghci> :t apply
apply :: (t1 -> t2) -> t1 -> t2
ghci>
```

Question 3:

```
C:\Cs331_Assignments\Assignment2>ghci
GHCi, version 9.2.1: https://www.haskell.org/gh
ghci> second xs = head(tail xs)
ghci> :t second
second :: [a] -> a
ghci> swap (x,y) = (y,x)
ghci> :t swap
swap :: (b, a) -> (a, b)
ghci> pair x y = (x,y)
ghci> :t pair
pair :: a -> b -> (a, b)
ghci> double x = x*2
ghci> :t double
double :: Num a => a -> a
ghci> palindrome xs = reverse xs == xs
ghci> :t palindrome
palindrome :: Eq a => [a] -> Bool
ghci> twice f x = f(f x)
ghci> :t twice
twice :: (t -> t) -> t -> t
ghci>
```

Question 3.5: Why is it not feasible in general for function types to be instances of the Eq class? When is it feasible?

To show that 2 functions are instance of a Eq class, we need to show that those functions are equivalent. But, to show two functions are equivalent, we must check output of the functions are equal for all the inputs. However, in general this is impossible because the kinds of inputs for general function type are infinite. So we cannot checkout every case. For example we can not check :

f1:: Int->Int->Int->Int
f1 a b c = a*(b*c)

f2::Int->Int->Int->Int
f2 ab c = (a*b)*c

However,We could check if these two functions were accepting finite inputs. Like **Bool->Bool** for example.

f1 :: Bool->Bool
f1 a = a

f2 :: Bool->Bool
f2 a = a|a

Here ,we can check as only valid inputs are True or False and we can easily determine equivalance just by comparing outputs for these 2 inputs.
That is the main reason why, it is not feasible in general for function types to be instances of Eq class.Functions with finite input cases are checkable. Hence, when function types have finite input cases, then only it is feasible.