The University of Nottingham

F COMPUTER SCIENCE

A , SPRING SEMESTER 2021-2022

ADVAN AL PROGRAMMING (COMP2003)

This is a take-home and open-book exam with answers to be submitted in Moodle no later than the date/time indicated in the Moodle dropbox.

Answer ALL THREE QUESTIONS

Submit your answers in a single PDF file, with each page in the correct orientation, to the dropbox in the module's Moodle page. You are recommended to write/draw your answers on paper and then scan them to a PDF file. Alternatively you may also type/draw your answers into electronic form directly and generate a PDF file.

Your solutions should include complete explanations and should be based on the material covered in the module. Make sure your PDF file is easily readable and does not require magnification. Text/drawing which is not in focus or is not legible for any other reason will be ignored.

Use the following naming convention for your PDF file: `StudentID_COMP2003`.
Include your student ID number at the top of each page in your PDF file.
Please do not include your name.

Staff are not permitted to answer assessment or teaching queries during the period in which your examination is live. If you spot what you think may be an error on the exam paper, note this in your submission but answer the question as written.

You must produce the answers by yourself only. You must adhere to the University's Policy on Academic Integrity and Misconduct. You are also not allowed to share this exam paper with anyone else or post it anywhere online.

Question 1: Monads And More **[overall 35 marks]**

a) Show h d type `Result` declared below can be made into an instanc ass, and explain your definition: [5 marks]

```
data    | ERROR String
        ing Show
```

b) Show how `Result` can be made into an instance of the `Applicative` class, and illustrate your definition using two simple examples. [5 marks]

c) Explain what happens if more than one error string arises while evaluating an applicative-style expression for the `Result` type. [3 marks]

d) Prove the following applicative law for `g :: a -> b` and `x :: a`, and explain what this law means in practical terms: [5 marks]


```
pure g <*> pure x = pure (g x)
```


e) Define a *non-monadic* function `sumeven :: [Int] -> Result Int` that returns the sum of a list of integers if they are all even, and otherwise returns a suitable error string. Your definition should be recursive. [5 marks]

f) Assuming that the `Result` type has already been made into an instance of the `Monad` class, show how the function `sumeven` can be redefined in an equivalent but more concise manner using the `do` notation. [5 marks]

g) Given the definitions

```
fmap g mx = mx >>= (return . g)

(f.g) x = f (g x)
```

prove the following property using the monad laws: [7 marks]

```
fmap g mx >>= f  =  mx >>= (f.g)
```

Question 2: Reasoning About Programs **[overall 35 marks]**

a) Using t definition, show how equational reasoning could be used in at False = True:

```
isEmp
isEmp
```

Explain why the reasoning in your proof is incorrect. [5 marks]

b) Given the definitions


```
map f []     = []
map f (x:xs) = f x : map f xs

(f.g) x = f (g x)
```


prove the following property by induction on the list xs: [7 marks]


```
map f (map g xs)  =  map (f.g) xs
```


c) Given the definitions

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
              deriving Show

mirror (Leaf x)   = Leaf x
mirror (Node l r) = Node (mirror r) (mirror l)
```

prove the following property by induction on the tree t: [6 marks]

```
mirror (mirror t)  =  t
```

d) Given the definition

```
mult
mult                    t xs
```

explain using the expression `mult [2,3,4]` why this definition is potentially inefficient in memory usage. [5 marks]

e) Given the specification

```
mult' n xs  =  n * mult xs
```

calculate a recursive definition for `mult'` using constructive induction on the list `xs`. You may assume standard arithmetic properties. [7 marks]

f) Given the revised definition `mult xs = mult' 1 xs`, explain using the same example as 2(d) why this definition is potentially more efficient. [5 marks]

Question 3: The State Monad **[overall 30 marks]**

Write a sh[illegible]500 words) that explains the simple parameterised type of st[illegible]ared below, and how suitable monadic functions `return` a[illegible]ined for this type:

```
type [illegible](a, State)
```

Further instructions:

- Your essay should have a clear narrative structure, rather than simply being Haskell definitions. You may assume your audience is familiar with the basics of monads, but has no experience with the state monad.
- Please include a word count at the end of your essay. Your Haskell definitions should be included in the word count.

[30 marks]