The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, SPRING SEMESTER 2022-2023

ADVANCED FUNCTIONAL PROGRAMMING (COMP2003)

Time to Answer the Exam Paper: TWO HOURS

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) Given the type declaration

```
data List a = Nil | Cons a (List a)
```

for which `Nil` represents an empty list and `Cons x xs` represents a non-empty list, show how the parameterised type `List` can be made into an instance of the `Functor` class, stating the type of the function you define. [5 marks]

b) Define a recursive function `append :: List a -> List a -> List a` that appends two lists together to give a single list. [3 marks]

c) Using `append`, show how `List` can also be made into an instance of the `Applicative` and `Monad` classes, stating the type of all functions. [12 marks]

d) Define a *non-monadic* function

```
replace :: List a -> Int -> (List Int, Int)
```

that replaces every element in a list by a unique or *fresh* integer, by taking the next fresh integer as an additional argument to the function, and returning the next fresh integer as an additional result. [5 marks]

e) Using the parameterised type

```
newtype ST a = S (Int -> (a, Int))
```

of state transformers whose state is an integer, define a *monadic* function

```
replace' :: List a -> ST (List Int)
```

that has the same behaviour as `replace`, but is defined using the `do` notation. There is no need to define the `ST` monad in your answer. [10 marks]

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

a) Given the type declaration

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

define functions `leaves` and `nodes` of type `Tree a -> Int` that count the number of leaves and nodes in a tree, respectively. [6 marks]

b) Prove the following property by induction on the tree `t`: [7 marks]

```
leaves t = nodes t + 1
```

c) Given the function definitions

```
map f (Leaf x)   = Leaf (f x)
map f (Node l r) = Node (map f l) (map f r)

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

prove the following property by induction on the tree `t`: [10 marks]

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

d) Given the definitions

```
data Nat = Zero | Succ Nat

take Zero     _      = []
take (Succ n) (x:xs) = x : take n xs

repeat x = x : repeat x
```

calculate a recursive function

```
replicate :: Nat -> a -> [a]
```

that satisfies the following specification: [12 marks]

```
replicate n x = take n (repeat x)
```

Question 3: The Maybe Monad [overall 30 marks]

Write a short essay (maximum 500 words) that explains how Haskell functions that may fail can be written using the >>= operator for the `Maybe` monad.

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 Haskell, but has no experience with the `Maybe` monad.

– Please include a word count at the end of your essay. Any Haskell definitions you make should be included in the word count.

[30 marks]