COM2001 - Spring Semester 2017–2018

Assignment 2: Theory

Deadline (MOLE): 3pm Friday 18th May (Week 12)

This assignment comprises four questions, and tests your understanding of various theoretical topics covered during the Spring semester. Some of these topics may not be introduced until relatively late in the course, so don't worry if you don't understand all of the questions right away. Make a note of the things you don't understand, so that you can recognise them when they eventually appear in lectures. Don't panic if you find parts of the questions a bit tricky – they are constructed so that competent students can do well, while still leaving plenty of scope for high-flyers to show their capabilities.

MARKING

This assignment is worth 50% of this semester's mark for the module (i.e. 25% of the entire module mark). The total number of marks available for this assignment is 250. A breakdown of the number of marks for each question is included in the assignment. It is important that you *explain your answers* in a way that makes them easy to understand. If you are unable to provide formal mathematical proofs when requested to do so, you should still try to provide informal (i.e. English language) proofs, as these will still count in your favour if the underlying reasoning is sound.

WHAT TO SUBMIT

You should submit your work via MOLE as a single PDF file.

Given the mathematical nature of these questions, hand-written solutions are acceptable *provided they are legible*. If we cannot read or understand your solutions, we will assign a mark of zero. Hand-written solutions should be scanned to form a single PDF file prior to submission. University printers can be used to scan work free of charge, see `https://www.sheffield.ac.uk/cics/students/print-copy-scan`.

ADDITIONAL RESOURCES

These questions assume you are familiar with (a) Haskell's type-inference rules, and (b) the rules for constructing a step-counting function. The relevant material is covered in (a) *Week 02 – Constraints and type identification*, and (b) *Week 10 – Complexity* (these slides will be made available on MOLE no later than Tuesday of Week 8).

DROP-IN SESSIONS

The Wednesday morning practical sessions only run until Week 6 (Weeks 7 and 12 are reading weeks). When we return after Easter, the Tuesday and Thursday sessions will continue as normal during Weeks 8 to 11, but the Wednesday sessions will be replaced by drop-in surgeries. During Weeks 8–11, I will be in my office (Room 111 on the first floor of Regent Court) every Wednesday from 11am until noon, and you are encouraged to drop-in with any questions you may have about the assignment or the course material more generally. Don't worry that I might be busy doing something else during drop-in sessions (I won't), and don't make an appointment to see me – just turn up with your questions!

Question 1. Consider the following code. Suppose `T` is a data type which has been declared to be an instance of the class `Distinguished`. For which finite defined values `str` of type `Stream T` does the equality

```
sLen str == sLen (sRev str)
```

hold? Justify your answer by providing a detailed formal proof that your answer is correct.

```
class Distinguished a where
  special :: a

data Nat = Zero | Succ Nat
  deriving (Eq, Show)

data Stream a = None | Append (Stream a) a
  deriving (Eq, Show)

sPop :: Stream a → Stream a
sPop s = case s of
  None            → None                  -- [sPop.0]
  Append None _   → None                  -- [sPop.1]
  Append s' x     → Append (sPop s') x   -- [sPop.n]

sTop :: Distinguished a ⇒ Stream a → a
sTop s = case s of
  None            → special               -- [sTop.0]
  Append None x   → x                     -- [sTop.1]
  Append s' _     → sTop s'               -- [sTop.n]

sRev :: Distinguished a ⇒ Stream a → Stream a
sRev s = case s of
  None            → None                  -- [sRev.0]
  Append None _   → s                     -- [sRev.1]
  _               → Append (sRev s') x'  -- [sRev.n]
    where
      s' = sPop s                         -- [sRev.s']
      x' = sTop s                         -- [sRev.x']

sLen :: Stream a → Nat
sLen s = case s of
  None          → Zero                    -- [sLen.0]
  Append s' _   → Succ (sLen s')          -- [sLen.n]
```

MARKS AVAILABLE FOR QUESTION 1: 60
Correct answer: 10
Sensible proof structure: 20
Clear explanation: 30

Question 2. Given the following code, show in detail how Haskell determines the type of the function `h`. You should make it clear at each point of your proof which type-inference rule you are using.

```
f :: Num a ⇒ Either Bool a → b
g :: c → Either c Int

f = f
g = g
h = \x  →  x f g
```

MARKS AVAILABLE FOR QUESTION 2: 50
Correctly identifying the type of h: 10
Clear explanation: 40

Question 3. A client asks for a program which takes two finite defined lists (possibly of different types) and implements the function

$$\delta(list_1, list_2) = \begin{cases} length(list_1) - length(list_2) & \text{if } list_2 \text{ is not longer than } list_1 \\ -\delta(list_2, list_1) & \text{otherwise} \end{cases}$$

A programmer provides the following implementation:

```
diff :: [a] → [b] → Int
diff xs ys = case (xs, ys) of
  ([ ], [ ])      →   0
  ( xs, [ ])      →   length xs
  ([ ], ys )      → -(length ys)
  (x:xs, y:ys)    →   diff xs ys
```

Give a detailed formal proof that `diff` is a totally correct implementation of the function δ.

MARKS AVAILABLE FOR QUESTION 3: 70
Clear statement of what needs to be proven: 5
Proof of correctness: 25
Proof that correctness is total: 15
Clear explanation: 25

Question 4. Consider the function `sRev :: Distinguished a ⇒ Stream a → Stream a` defined as part of the program provided on page 2. Show in detail how to convert this into a function

`sRevT :: Distinguished a ⇒ Stream a → Stream a`,

where `sRevT s` outputs the number of steps involved in computing `sRev s`, and hence determine the worst-case time complexity of `sRev`. You may assume that Haskell is an eager, rather than a lazy, language.

MARKS AVAILABLE FOR QUESTION 4: 70
Correct construction of `sRevT` and other relevant functions: 40
Stating clearly which rule is used at each stage of the construction of `sRevT`: 10
Stating clearly what 'size' parameter you are using for the cost function: 5
Converting `sRevT` into a cost function: 10
Correct complexity class: 5

END OF ASSIGNMENT