UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics, and Science

School of Computer Science and Statistics

Integrated Computer Science Programme — Hilary Term 2014
Year 4 Annual Examinations

CS4012: Topics in Functional Programming

Monday, 13th January — Luce Upper — 09:30–11:30

Mr. Glenn Strong

Instructions to Candidates:

Attempt **three** questions.
(all questions are worth equal marks)

There is a Reference to useful functions at the end of the paper (p5).

Materials permitted for this examination:

none

Q1. In this question you are to write a Haskell Domain Specific Language which implements "turtle graphics". In this system we imagine that there is a small robot (called the turtle) which holds a pen. As the turtle moves about on a sheet of paper the pen leaves a trail, and this is used to draw pictures.

The turtle understands only four primitive operations:

- `step` (which moves the turtle forward a single step)
- `right` (which instructs the turtle to rotate 90 degrees right)
- `up` which lifts the pen from the paper (subsequent movement will not leave a trail)
- `down` which lowers the pen to the paper (subsequent movement will leave a trail)

For this question you must implement an embedded DSL in Haskell which simulates the turtle graphics. The result of running a program in this DSL should be a list of line segments to draw (represented as pairs of points).

The library should allow for programs like this:

```
drawing = do
  down;  forward ;  right ;  up;  forward ;  down ; right ;  forward
```

to produce a list of line segments:

```
[ ( (0,0), (0,1) ), ( (1,1), (1,0) ) ]
```

(a) Give a definition for a Haskell data type `Program` which will form the basis for the turtle graphics monad, and which will allow Haskell programs such as the example above (i.e. make `Program` a monad).

[8 marks]

(b) Provide implementations of the four primitive operations required of the turtle graphics language

[5 marks]

(c) Make `Program` an instance of `Monad`.

[8 marks]

Continued on next page...

(d) Provide a 'draw' function with the following type (asume the turtle begins at the origin, facing north).

```
draw :: Program -> [ Line ]
```

[4 marks]

(e) Give a definition of a function `repeat` which will run a turtle graphics program repeatedly so that a square could be drawn via the expression `repeat 4 (do forward ; right )`.
[3 marks]

(f) We could extend the set of primitives to allow for more than four possible facings; if we change the definition of `right` so that it rotates the turtle only 45 degrees right then there will be eight possible facings.

Taking your existing solution indicate systematically where it would need to change in order to support this change. You do not need to provide the full implementation, but explain any assumptions or restrictions that are relevant to your answer.

[5 marks]

Q2.

(a) Explain how generalised algebraic data types (GADTs) in Haskell allow programmers to express constraints on data types, and suggest why this may be a useful feature for programmers.

[10 Marks]

(b) Give a definition of a list type in Haskell that can be used to store values of differing types (that is, a list where it is not necessary to have each element of the same type).

Explain what problem a programmer will encounter when attempting to operate on elements of such a list and describe *two* mechanisms that could be used to overcome the problem. For each mechanism, indicate what the trade-offs the programmer will encounter might be.
[15 Marks]

(c) Dependent type systems allow for an even greater degree of expression than GADTs. Explain why this is, and suggest why language designers may be reluctant to incorporate dependent types into their programming language despite this.

[8 Marks]

Q3.

(a) Explain what a *monad transformer* is, and indicate why it is necessary to have a special technique to overcome issue with combining arbitrary monads.

[10 Marks]

(b) Give an implementation for the Writer Transformer Monad (that is, give the code for a monad transformer that can add a "tell" operation to any monad; the operation should accumulate a list of values. It could be used, for example, to build up a string containing logging data).

[10 Marks]

(c) Explain why the Haskell IO monad cannot be a monad transformer.

[4 Marks]

(d) What does the "lift" function do in the monad transformer class?

[5 Marks]

(e) Sketch how the Haskell `mtl` libraries use type classes to avoid requiring programmers to make explicit reference to the lift operation.

[4 Marks]

Q4. (a) Explain the basic concept of an `MVar` as it is encountered in Concurrent Haskell, and illustrate how individual `MVar` values can be combined to produce the concept of a `Channel`. You should include a discussion of how the implementation of Channels permits channels to be safely duplicated.

[10 Marks]

(b) Software Transactional Memory (STM) represents an attempt to simplify approaches to concurrent programming by providing an alternative to lock based approaches. Explain the STM approach to concurrency and sketch how an implementation might be provided.

[15 Marks]

(c) Is there a safe way to incorporate either IO actions or `MVars` in STM, and if not why not?

[8 Marks]

Reference

```
class Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a -> m b -> m b
  fail   :: String -> m a

class MonadTrans t where
    lift :: Monad m => m a -> t m a
```