Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science
Computer Science and Business
Year 4 Annual Examination

Hilary Term 2017

CS4012: Topics in Functional Programming

Friday, 6th January

Exam Hall

09:30–11:30

Mr. Glenn Strong

Instructions to Candidates

You may not start this examination until you are instructed to do so by the Invigilator.

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

Q 1. In this question you are to write a Haskell Domain Specific Language which implements a simple procedural drawing language based on "turtle graphics". In this system we imagine that there is a small robot (called the turtle) which holds a pen. This pen can either be put down touching the floor, or raised up. The turtle can be ordered to move about on the floor, so that the pen draws patterns.

The turtle implements only four primitive operations:

- `forward` n (which moves the turtle an integer number of steps forward)
- `turn` n (which moves the turtle an integer number of degrees clockwise)
- 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 will be a list of line segments to draw (represented as pairs of points).

The eDSL allows for a monadic description of the turtle graphics program. Here is a possible example program:

```
drawing = do
  down;  forward 100;  right 90; up;  forward 100;  down
  right 90;  forward 100
```

Running the above program in the turtle graphics monad produces a set of line segments; for example a possible output of the above program would be:

```
[ ( (0,0), (0,100) ), ( (100,100), (100,0) ) ]
```

Using the following declarations:

```
data TurtleState = TurtleState { pos :: Point, facing :: Angle,
                                 penDown :: Bool, trail :: [ Line ] }
data Program a = Program { runTurtle :: TurtleState -> (TurtleState, a) }
type Point = (Int, Int)
type Line = (Point,Point)
```

Continued on next page...

(i) Provide implementations of the following primitive operations:

   i. `forward :: Int -> Program ()`

   ii. `right :: Int -> Program ()`

   iii. `up :: Program ()`

   iv. `down :: Program ()`

[35 marks]

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

[25 marks]

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

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

[10 marks]

(iv) 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 100 ; right 90)` .

[15 marks]

(v) State whether your turtle graphics eDSL should be considered a *deep* or a *shallow* embedding, and comment on which embedding would be most suitable.

[15 marks]

Q 2. (i) Explain each of the following terms as it is used in Haskell, and comment on what each can be used for:

i. Phantom type

ii. Existential type

iii. Generalised Algebraic Data Type (GADT)

[30 Marks]

(ii) Given a simple expression language which can represent integer and boolean constants, and which has some arithmetic operators and an equality test, explain the principle by which GADT's could be used to define a "type-safe" interpreter where it is not possible to construct an expression in which arithmetic operators are mistakenly applied to boolean values. Give an example implementation. It is not necessary to provide the entire interpreter - it's sufficient to give only one numeric and one boolean constructor to demonstrate type-safety.

[35 Marks]

(iii) It is sometimes said that GADTs allow Haskell to simulate some features of dependent types. Explain what is meant by this, and include an explanation of what is meant by the term "dependent type".

[35 Marks]

Q 3. (i) Haskell defines three related type classes, `Functor`, `Applicative`, and `Monad`. Explain briefly how these classes relate to each other.

[20 Marks]

(ii) What is meant by saying "Applicative Functors compose, Monads do not", and explain how the notion of Monad Transformers can be used to help solve this problem.

[30 Marks]

(iii) Give an implementation for the Reader Transformer Monad (that is, give the code for a monad transformer that can add an "`ask`" operation, which allows the inspection of a piece of immutable state, to any monad). Provide implementaions for `ask` and for the interpretation function`runReaderT`.

[25 Marks]

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

[10 Marks]

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

[15 Marks]

Q 4. (i) Concurrent Haskell provides Channel-based inter-thread communication via the `Chan` data type. Explain how these channels are constructed from the more primitive `MVars`, and discuss how the implementation attempts to minimise the risk of deadlock.

[25 Marks]

(ii) Software Transactional Memory (STM) represents an attempt to simplify concurrent programming by providing an alternative to lock-based approaches. Explain the STM approach to concurrency, sketch how an implementation might be provided, and discuss briefly the limitations of the STM approach to designing concurrent programs.

[50 Marks]

(iii) What would be the consequences of permitting `IO` actions inside an `STM` atomic block, and what mechanism is used to prevent this from happening?

[25 Marks]