XCS40121

UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics, and Science

School of Computer Science and Statistics

Integrated Computer Science Programme
Senior Sophister Examination

Hilary Term 2013

CS4012: Topics in Functional Programming

Friday, 4th January   Regent House   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 (pp6–6).

Materials permitted for this examination:

none

Q1. Domain Specific Languages

(a) A domain specific language "RPN" based on a reverse-polish notation calculator has these operations (which only work on integers):

- Stack operations: push, pop, clear
- Arithmetic operations: Addition, subtraction, multiplication, (integer) division (each works on topmost two stack elements, and places the result on the stack).

Define a datatype `RPN a` and make it an instance of the Monad class. Include functions which implement the RPN calculator language. Calculations can produce a numeric answer *or* they may fail with an error message (for example if an attempt is made to pop from an empty stack, or to divide by zero).

20 Marks

(b) Explain the terms *deep* or *shallow* embedding with respect to Domain Specific Languages and explain which applies to your implementation of the RPN calculator.

5 Marks

(c) In a DSL shallow embedding "allows transformations like optimizations before translating to the target language. ". Explain what is meant by this and how we might take advantage of it in an implementation (you may refer to your RPN monad if it is appropriate but do not be limited to discussing only that).

8 marks

Q2.

(a) What does the term "Generalised Algebraic Data Type" refer to in Haskell. Explain in your answer in what sense these dec-
larations are "generalised" and what features they offer that are useful to software developers.

8 Marks

(b) An AVL tree is a kind of balanced binary search tree with an extra invariant:

- The heights of the subtrees of any node differ by no more than one

Give a declaration for a Haskell data type `AVL a` which uses GADTs to ensure that the AVL tree balance condition cannot be violated. Hint: you may find the declarations of `Zero` and `Succ` in the reference useful. There is no need to include any operations on the tree.

16 Marks

(c) Comment *briefly* on the relationship between GADTs and Existentially quantified types.

4 marks

(d) 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".

5 Marks

Q3.

(a) What problem do Monad Transformers solve?

5 Marks

(b) Give an implementation for the State Transformer Monad (that is, give the code for a monad transformer that can add stateful 'get' and 'set' operations to any monad).

15 Marks

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

5 Marks

(d) What is the Identity monad, and explain why it is useful in the context of monad transformers?

4 Marks

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

4 Marks

Q4.

(a) The standard Haskell libraries offer both Parallel (through 'par' and 'pseq') and Concurrent (through 'forkIO') approaches to programming. Explain how these approaches differ semantically.

10 Marks

(b) Explain how `MVar` values are used in Haskell to enable communication and locking between threads.

5 Marks

(c) Software Transactional Memory (STM) provides an alternative to lock-based communication.

(i) What does it mean in STM for a block to be atomic?

5 Marks

(ii) What is the effect of the `retry` function in the STM monad?

8 Marks

(iii) Why is it not permitted to perform an IO action inside an STM "atomically" block? What mechanism is used in Haskell to prevent this?

5 Marks

Reference (I)

Q2(a):

```
data Zero = Zero
data Succ a = Succ a
```

Reference (II)

General:

```
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
```