ECM2418

(with Answers)

Referred / Deferred

UNIVERSITY OF EXETER

COLLEGE OF ENGINEERING, MATHEMATICS AND PHYSICAL SCIENCES

COMPUTER SCIENCE

Referred/Deferred Examination, August 2019

Computer Languages and Representations

Module Leader: Dr David Wakeling

Duration: TWO HOURS

Answer all questions.

Question 1 is worth 40 marks. Other questions are worth 30 marks each.

No electronic calculators of any sort are to be used during the course of this examination.

This is a CLOSED BOOK examination.

Question 1

(a) What is wrong with this Haskell function definition, which is supposed to find the last element of a list?

```
last :: [a] -> b
last (x:xs) = last xs
last []     = []
```

> **BOOKWORK**
>
> One problem with this Haskell function definition is that the type should be `[a] -> a` **[2 marks]**. Another problem is that the base case for the recursion is "too late" — the pattern that should be matched is `[x]` **[2 marks]**.

(4 marks)

(b) What is meant by referential transparency in functional programming, and why is it important?

> **BOOKWORK**
>
> Referential transparency in functional programming means that variables always have the same value **[1 mark]**, and that equal terms can always be substituted for one another **[1 mark]**. It is important because it makes it possible to transform and reason about program behaviour **[1 mark]** using algebraic means **[1 mark]**.

(4 marks)

(c) Write the polymorphic type of the Haskell function

```
mystery (x:xs) e
   | x < e        =  [x] ++ mystery xs e
   | otherwise    =  [e,x] ++ xs
mystery []     e  =  [e]
```

Say what it does, and say how its efficiency could be improved.

> **BOOKWORK**
>
> The polymorphic type is `Ord a => [a] -> a -> [a]` **[1 mark]**. It inserts an element into an ordered list **[1 mark]**. The efficiency could

be improved by replacing `[`p`]` `++` q by $p : q$ throughout **[2 marks]**.

(4 marks)

(d) (i) What is meant by the *most general unifier* of two Prolog literals?

BOOKWORK

The most general unifier of literals x and y is the smallest substitution μ such that $x\mu = y\mu$, if any such substitution exists, otherwise it is undefined **[2 marks]**.

(2 marks)

(ii) Write down the most general unifier of the literals

```
married(father(X),mother(X))
```

and

```
married(father(john),Y),
```

BOOKWORK

The most general unifier of the given literals is `[X/john, Y/mother(john)]` **[2 marks]**.

(2 marks)

(iii) State the result of applying this unifier to the two literals.

BOOKWORK

When applied to the given literals the resulting literal is `married(father(john),mother(john))` **[2 marks]**.

(2 marks)

(e) What is the difference between imperative programming and declarative programming, and logic programming in particular?

BOOKWORK

In imperative programming, the user tells the computer via a program

the exact sequence of steps to perform in order to achieve/compute a particular result. Essentially the user describes the algorithm to achieve the result **[2 marks]**.

In declarative programming, the user specifies the characteristics of the result, and in particular in logic programming the characteristics of the results are described via logic. The computer uses an engine, in particular for logic programming a reasoning engine is used, in order to compute the result according to the characteristics specified the user **[2 marks]**.

(4 marks)

(f) Consider the following Prolog clauses:

```
f([],0).

f([X|Y],R):-
      f(Y,R1),
      R is X + R1.
```

(i) Write down the result of making the query

```
?- f([1,2,3],X).
```

and give the meaning of the predicate `f`.

BOOKWORK

```
?- f([1,2,3],X).
X = 6.
```

[2 marks]

`f(X,Y)` is true if `X` is a list of numbers and `Y` is a number which is the sum of the numbers in X **[2 marks]**.

(4 marks)

(ii) Give an example of a query involving `f` which returns `false`, and (separately) a query involving `f` which generates an error.

A query which returns `false` is `?- f([],1).` because the sum of the members of an empty list is 0 (other correct answers are fine) **[1 mark]**.

A query which generates an error is `?- f([a,2,3],X).` because `a` is not a number (other correct answers are fine) **[1 mark]**.

(2 marks)

(iii) Would the definition of `f` be correct if it were defined as:

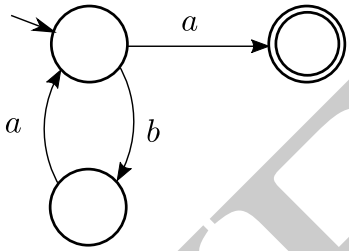
```
f([],0).

f([X|Y],R):-
      R is X + R1,
      f(Y,R1).
```

Explain why.

> The second definition of the predicate `f` would not be correct because `R1` is not instantiated when it is added to `X` **[2 marks]**.

(2 marks)

(g) Consider the following finite-state automaton



(i) State which of the following strings are accepted, and which are rejected, by this automaton

$$\Lambda, a, b, baa, bab, ab.$$

> **BOOKWORK**
>
> Strings accepted: a, baa, **[1 mark]**.
> Strings rejected: Λ, b, bab, ab **[1 mark]**.

(2 marks)

(ii) Write down a regular expression for the language accepted by this automaton.

> **BOOKWORK**
>
> The regular expression of the language recognised by the automaton is: $(ba)^*a$ **[2 marks]**.

(2 marks)

(iii) Consider the language L_1 defined by the regular expression $(baa)^*$. State whether L_1 is a subset of the language L of the automaton above, giving your reasons.

BOOKWORK

L_1 is not a subset of L **[1 mark]**, because $baabaa$ is a string belonging to L_1 but not to L **[2 marks]**.

(3 marks)

(iv) Consider the language L_2 defined by the regular expression $(baba)^+a$. State whether L_2 is a subset of the language L of the automaton above, giving your reasons.

BOOKWORK

L_2 is a subset of L **[1 mark]**. In particular, if w is a string of L_2, then w has to start with a sequence of 'baba', which can be repeated multiple times, which is compatible with the part $(ba)^*$ of the language L. The last 'a' imposed by L_2 is compatible with the edge from the initial state to the accepting state of the automaton **[2 marks]**.

(3 marks)

(Total 40 marks)

Question 2

(a) Write down a value of the Haskell rose tree data type

```
data Rose = Rose Char [Rose]
```

What would have to be changed to make this rose data tree type polymorphic? What would have to be added to this data type to allow values to be printed?

An example value is

```
Rose 'w' [Rose 'x' [Rose 'y' []]]
```

[2 marks].

To make the type polymorphic, use a type variable

```
data Rose a = Rose a [Rose a]
```

[2 marks].

Add `deriving Show` to allow values to be printed **[1 mark]**.

(5 marks)

(b) Write a definition of a Haskell function that returns `True` when a particular character occurs in a particular rose tree, and has the type

```
containsRose :: Char -> Rose -> Bool
```

What would have to be be changed to make the function work with polymorphic rose trees?

The basic contains function

```
containsRose ::  Char -> Rose -> Bool
containsRose x (Rose c rs)
  = if x == c then True
    else foldr (||) False (map (containsRose e) rs)
```

For condition **[1 mark]**, for `foldr` **[2 marks]**, for `map` **[2 marks]**.

To make this work with polymorphic rose trees, a type variable and an equality constraint would be required

```
containsRose ::  Eq k => k -> Rose k -> Bool
```

[1 marks].

(6 marks)

(c) Write a definition of a Haskell function suitable for performing a *parallel* map using the standard `par` and `seq` annotations.

BOOKWORK

```
parmap ::  (a -> b) -> [ a ] > [ b ]
parmap f []     = []
parmap f (x:xs) = par fx (seq fxs (fx :  fxs))
                  where
                  fx = f x
                  fxs = parmap f xs
```

For type **[1 mark]**, for first equation **[1 mark]**, for second equation — for use of annotations **[2 marks]**, for local definitions **[2 marks]**.

(6 marks)

(d) How could a Haskell function that says whether a rose tree contains a value use a parallel map to improve performance, and why might this not work as well as expected?

On the face of it, all that is required is to replace the sequential `map` with the parallel `parmap`

```
containsRose ::  Char -> Rose -> Bool
containsRose x (Rose c rs)
 = if x == c then True
   else foldr (||) False (parmap (contains e) rs)
```

[2 marks]

This might not work as well as expected because for some large trees **[1 mark]**, more tasks might be spawned that there are processors, swamping the machine with work that can only be queued **[1 mark]**, and at the leaves of these trees **[1 mark]**, many tasks might be spawned to do small amounts of computation, failing to repay their set-up cost **[1 mark]**. Moreover, for some trees with a large branching factor **[1 mark]**, the sequential `foldr` will be a bottleneck **[1 mark]**.

(8 marks)

(e) An experienced programmer claims that functional programming will never

be used much in industry. Do you agree? Give your reasons.

Either argue yes, using functional programming means throwing away years of programming experience, existing code, libraries, and algorithms to be replaced by a technology that is not widely used or supported. No significant industrial player would be prepared to take such a high risk. Therefore functional programming will remain the stuff of academic study rather than industrial use.

For an argument such as this **[5 marks]**.

Or argue no, functional programming is already being used in industry, either directly by financial institutions using Haskell, or indirectly by more traditional programming languages borrowing ideas form it. For example, higher-order functions, polymorphic types, pattern-matching, and lambda functions all feature in Apple's Swift. Many other languages have map/filter/reduce as library functions. Therefore, functional programming is already being used in industry, and that usage will only increase in future.

For an argument such as this **[5 marks]**.

(5 marks)

(Total 30 marks)

Question 3

(a) Define a Prolog predicate `multTable(X,Y)` which prints the multiplication table for X from 0 to Y, for example:

```
?- multTable(7,9).
7 x 0 = 0
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
```

(10 marks)

```
linePrint(X,Y) :-
        write(X),
        write(' x '),
        write(Y),
        write(' = '),
        R is X * Y,
        write(R),
        nl.
```

Upto **[5 marks]**.

```
multTable(X,0) :- linePrint(X,0).
multTable(X,Y) :-
        Y > 0,
        Y1 is Y - 1,
        multTable(X,Y1),
        linePrint(X,Y).
```

Upto **[5 marks]**.

(b) Consider the language L defined via the regular expression $(0|1)^*001(0|1)^*$.

(i) Describe the language L in words.

L is the language defined over the alphabet $\{0, 1\}$, and whose strings contain the sequence 001.

(2 marks)

(2 marks)

(ii) Write two strings belonging to L and two strings not belonging to L, respectively.

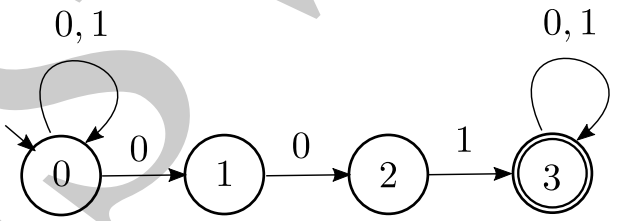
Strings belonging to L: $001, 0001$. **[1 mark]** per string, other correct proposals are fine.

String not belonging to L: $01, 101$. **[1 mark]** per string, other correct proposals are fine.

(4 marks)

(iii) Define a non-deterministic finite-state automaton recognising L. (**Hint:** devise a machine that non-deterministically decides when the string 001 starts in the input). State where the non-determinism occurs.

A non-deterministic automaton recognising L is:

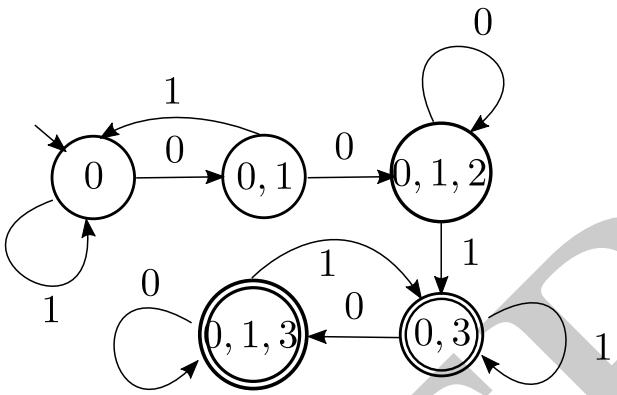


The non-determinism occurs in the state 0 where the machine, when reading 0, may move to state 0 or 1. Upto **[7 marks]** for machine states and transitions, of which **[1 mark]** for stating correctly where the non-determinism is.

(7 marks)

(iv) Transform the automaton for L defined above into a deterministic one.

The deterministic automaton derived from the automaton of the point above is:



Upto **[7 marks]** for machine states and transitions.

(7 marks)

(Total 30 marks)