ECM2418

(with Answers)

Referred / Deferred

UNIVERSITY OF EXETER

COLLEGE OF ENGINEERING, MATHEMATICS AND PHYSICAL SCIENCES

COMPUTER SCIENCE

Referred/Deferred Examination, August 2021

Computer Languages and Representations

Module Leader: Dr David Wakeling

Duration: TWO HOURS + 30 MINUTES UPLOAD TIME

Answer all questions.

Questions 1 and 2 are worth 40 marks each; question 3 is worth 20 marks.

This is an OPEN BOOK examination.

Question 1

(a) Simplify this λ-expression, showing your working

$$(\lambda x.\ \lambda y.\ \lambda z.\ y\ y\ x)\ 6\ (\lambda x.\ x)\ (\lambda x.\ \lambda y.\ x\ y)$$

$$\begin{aligned}
& (\lambda x.\ \lambda y.\ \lambda z.\ y\ y\ x)\ 6\ (\lambda x.\ x)\ (\lambda x.\ \lambda y.\ x\ y) \\
\Rightarrow\ & (\lambda y.\ \lambda z.\ y\ y\ 6)\ (\lambda x.\ x)\ (\lambda x.\ \lambda y.\ x\ y) \\
\Rightarrow\ & (\lambda z.\ (\lambda x.\ x)\ (\lambda x.\ x)\ 6)\ (\lambda x.\ \lambda y.\ x\ y) \\
\Rightarrow\ & (\lambda x.\ \lambda y.\ x\ y)\ (\lambda x.\ x)\ (\lambda x.\ x)\ 6 \\
\Rightarrow\ & (\lambda x.\ x)\ (\lambda x.\ x)\ 6 \\
\Rightarrow\ & (\lambda x.\ x)\ 6 \\
\Rightarrow\ & 6
\end{aligned}$$

One mark for each simplification (except the last trivial one). Partial marks awarded for partially-correct simplifications.

(5 marks)

(b) Rewrite the function "`addition`" below using pattern-matching.

```
addition as bs
   = if null as then bs
     else head as :  addition (tail as) bs
```

The "`addition`" function may be written using pattern-matching as follows.

```
addition [] bs
   = bs
addition (a:as) bs
   = a :  addition as bs
```

For first clause **[2 marks]**. For second clause **[3 marks]**.

(5 marks)

(c) In Morse Code, messages consist of sequences of dots, dashes and gaps. Write one Haskell data type suitable for representing dots, dashes and gaps, and another suitable for representing messages.

```
data Morse
   = Dot | Dash | Gap
```

[3 marks]

```
data Message
   = Message [ Morse ]
```

[2 marks]

(5 marks)

(d) Rewrite the function "`strange`" without using a higher-order function, add its polymorphic type, and say what it computes.

```
strange
   = foldr (++) []
```

The "`strange`" function may be written without using a higher-order function as follows.

```
strange ::  [ [ a ] ] -> [ a ]
strange []
   = []
strange (x:xs)
   = x ++ strange xs
```

For the type **[1 mark]**. For the definition, **[3 marks]**. This function concatenates a list of lists to give a single list **[1 mark]**.

(5 marks)

(e) Why is it claimed the functional programming languages might make it easier to program the computers of the future?

The computers of the future are likely to have large numbers of processors or cores **[1 mark]**. It is claimed that functional programming languages might make it easier to program these computers because they have no notion of sequence **[1 mark]** or state **[1 mark]**. This means that the cores can all run simultaneously without having to explicitly synchronise **[1 mark]** or communicate **[1 mark]**.

(5 marks)

(f) Rewrite the definition of the function "`both`" below using a single application of "`filter`", and say how you would use it to pick out numbers

that are multiples of both 7 and 9 from a list.

```
both p q xs
   = (filter p . filter q) xs
```

A definition of the function "`both`" below using a single application of "`filter`" is

```
both2 p q
   = filter (\x -> p x && q x)
```

[3 marks]

One way to use it to pick out numbers that are multiples of both 7 and 9 from a list.

```
both2 (\x -> x `mod` 7 == 0) (\x -> x `mod` 9 == 0)
```

[2 marks]

(5 marks)

(g) The function "`toList`" below converts values of the type "`Mystery`" to lists.

```
toList :: Mystery a -> [ a ]
toList (Mystery x xs)
   = x : concatMap toList xs
```

Show a possible definition of the "`Mystery`" data type.

A possible definition of the "`Mystery`" data type is

```
data Mystery a
   = Mystery a [ Mystery a ]
```

Partial marks for partial solutions, although this may be all-or-nothing **[5 marks]**.

(5 marks)

(h) Rewrite the Haskell function "`mish`" below, making the minimum changes to the right-hand sides of the clauses required for it to have the type given.

```
mish ::  [a] -> [a]
mish
   = aux []
      where
      aux z []
         = [z]
      aux z (a:as)
         = aux (a ++ z) as
```

First clause of "`aux`" should be:

```
aux z []
   = z
```

[2 marks]

Second clause should be:

```
aux z (a:as)
   = aux (a :  z) as
```

[3 marks]

(5 marks)

(Total 40 marks)

Question 2

(a) Table 1 shows three musicians and their musical genres.

Musician	Musical Genre
Jake Bugg	Indie Folk, Indie Rock
Kylie Minogue	Pop, Dance, Disco
Snoop Dogg	Hip Hop, Funk, Gangsta Rap

Table 1: Musicians and their musical genres.

Write Prolog predicates to express the information in Table 1, and show an example query for the musical genre of a given musician.

```
genre( jake_bugg, [ indie_folk, indie_rock ]).
genre( kylie_minogue, [ pop, dance, disco ]).
genre( snoop_dogg, [ hip_hop, funk, gangsta_rap ]).
```

[3 marks]

```
main :- genre( snoop_dogg, X ), write( X ).
```

[2 marks]

(5 marks)

(b) The autonomous driving system of a car is written in Prolog. The code for avoiding obstructions relies on the "closed world assumption". With the aid of examples, explain how this might effect the operation of the system.

The "closed world" assumption means that anything not described as an obstruction will not be considered as one **[1 mark]**. Thus, the system will avoid things that the programmer has classified as obstructions **[1 mark]**, such as cars, lorries and busses **[1 mark]**. But the system will not avoid things that the programmer has not classified as obstructions, such as wheelie bins, letter boxes, and horses **[1 mark]**. Note that further things could be classified as obstructions by assertions in the program, but maybe only as the result of expensive experience **[1 mark]**.

(5 marks)

(c) A rushed Prolog programmer has written the following code. Briefly explain what it computes, and the programming technique that it uses.

```
p( B, E, R )
   :- q( B, E, 1, R ).

q( B, 0, A, A ).
q( B, E, A, R )
   :- E > 0,
      E1 is E - 1,
      A1 is B * A,
      q( B, E1, A1, R ).
```

The call `p(` X, N, R `)` sets $R = X^N$.

[3 marks]

The programming technique that it uses is that of an accumulating parameter, where the result is built up in a parameter before being returned.

[2 marks]

(5 marks)

(d) Rewrite the predicate "`best`" below using the Prolog conditional (if-else) notation, and say what it computes.

```
best( [R], R ).
best( [H|T], H )
   :- best( T, V ), H > V.
best( [H|T], V )
   :- best( T, V ), H =< V.
```

The "`best`" predicate may be written using Prolog conditional (if-else) notation as follows.

```
best( [R], R ).
best( [H|T], R )
   :- best( T, U ),
      ( H > U -> R = H; R = U ).
```

For the use of the conditional **[4 marks]**. This predicate computes the maximum in a list of integers **[1 mark]**.

(5 marks)

(e) The Japanese Fifth Generation Project aimed to encode program specifications as Prolog programs. Why?

The Japanese Fifth Generation Project aimed to encode program specifications as Prolog programs because

- specifications usually ambiguously written in natural language **[1 mark]** can be unambiguously written as logical descriptions in Prolog **[1 mark]**.
- specifications written as logical descriptions can be systematically **[1 mark]**, and indeed automatically **[1 mark]**, refined from inefficient ones, to efficient ones in Prolog, that may then be distributed **[1 mark]**.

(5 marks)

(f) A Prolog programmer concerned with efficiency has defined a "`factorial`" predicate as follows.

```
factorial( N, R )
   :- number( N ),
      factorial2( N, 1, R ).

factorial2( 0, R, R )
   :- !.
factorial2( N, A, R )
   :- N1 is N - 1,
      A1 is A * N,
      factorial2( N1, A1, R ).
```

Describe the attempts to improve efficiency in the this predicate, and say how effective they are likely to be.

The use of the "`number`" predicate restricts the "`factorial` predicate to numbers **[1 mark]**, but it is rather unlikely that the predicate will be used with them, as this would be a type error **[1 mark]**. The "!" in "`factorial2`" prevents unwanted backtracking once the result is known **[1 mark]**. The use of an accumulating parameter in "`factorial2`" makes the definition tail-recursive **[1 mark]**. Both are likely to be effective **[1 mark]**.

(5 marks)

(g) If it is possible to unify the two Prolog terms:

- `p( f( X ), Y, g( Y ), b )`
- `p( f( W ), a, g( b ), X )`

show the substitution that unifies them; otherwise say why such a substitution cannot be produced.

It is not possible to unify the two Prolog terms **[1 mark]**. This is because the atom `b` must unify with the variable `Y` **[1 mark]** in terms `g( Y )` and `g( b )` **[1 mark]**, but elsewhere the variable `Y` must also unify with the atom `a` **[1 mark]** in the terms `Y` and `a` **[1 mark]**.

(5 marks)

(h) Write a Prolog predicate that may be used to split a list at a particular point. For example,

```
splitAt( [ 'A', 'B', 'C' ], 2, X, Y ).
X = [ 'A', 'B' ]
Y = [ 'C' ]
```

```
splitAt( [], N, [], [] ).
splitAt( X,  0, [], X  ).
splitAt( [H|T], N, [H|V], W )
   :- M is N - 1, splitAt( T, M, V, W ).
```

For first clause **[1 mark]**. For second clause **[1 mark]**. Fof final clause **[3 marks]**.

(5 marks)

(Total 40 marks)

Question 3

(a) Consider the Deterministic Finite State Automaton (DFA) shown in Figure 1.

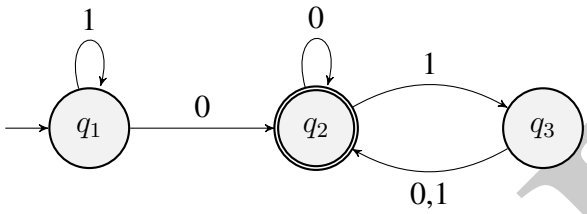


Figure 1: A DFA.

Show the transition table for this DFA and describe (in words) the strings that it accepts.

The transition table is:

	0	1
q_1	q_2	q_1
q_2	q_2	q_3
q_3	q_2	q_2

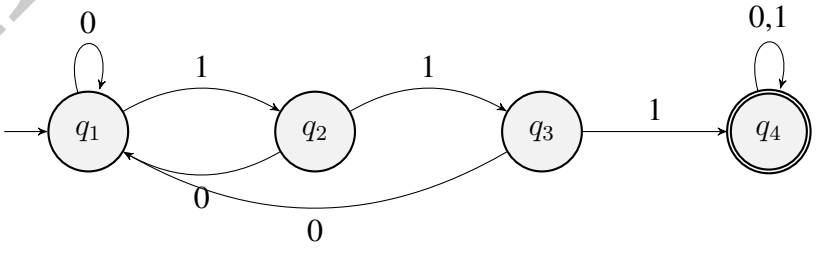
[3 marks]

This DFA accepts strings that contain at least one '0', and an even number of '1's following the last '0'.

[2 marks]

(5 marks)

(b) Show a Deterministic Finite State Automaton (DFA) that accepts any string from the alphabet { '0', '1' } containing at least three successive '1's.



For states **[2 marks]**; for transitions **[3 marks]**.

(5 marks)

(c) Consider the Nondeterministic Finite State Automaton (NFA) shown in Figure 2.

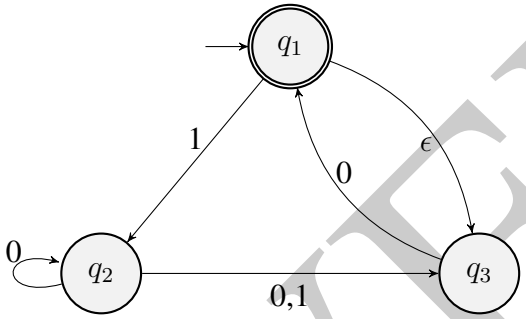


Figure 2: A NFA.

Show the transition table for this NFA say whether it accepts the strings "`1010`" and "`10110`".

The transition table is:

	`0`	`1`	ϵ
q_1	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	q_3	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

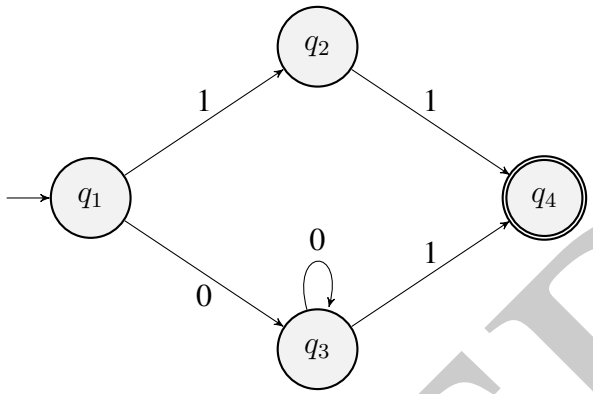
[3 marks]

This NFA accepts accepts the string "`1010`", but rejects "`10110`".

[2 marks]

(5 marks)

(d) Show a Deterministic Finite State Automation (DFA) that accepts strings described by the regular expression "`11` $\cup$ `0`+ `1`".



For states **[2 marks]**; for transitions **[3 marks]**.

(5 marks)

(Total 20 marks)