ECM2418

(with Answers)

UNIVERSITY OF EXETER

COLLEGE OF ENGINEERING, MATHEMATICS AND PHYSICAL SCIENCES

COMPUTER SCIENCE

Examination, January 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.

The marks for this module are calculated from 70% of the percentage mark for this paper plus 30% of the percentage mark for associated coursework.

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) Explain what is wrong with this Haskell function definition, which is supposed to append two lists together, and say how you would put it right.

```
append :: [t] -> [u] -> [t,u]
append (x:xs) ys = x : append ys xs
append []     ys = ys
```

BOOKWORK

One problem with this Haskell function definition is that the type is illegal because the type `[t,u]` is illegal **[1 mark]**. This could be put right by using the type `[t] -> [t] -> [t]` **[1 mark]**. Another problem with this Haskell function definition is that the arguments to the recursive `append` call are in the wrong order **[1 mark]**. This could be put right by writing `append xs ys` **[1 mark]**.

(4 marks)

(b) Assuming that its argument `n` is of type `Int`, write the type of the Haskell function `mystery`, defined as

```
mystery n = foldr (*) 1 [1..n]
```

and say what it computes.

BOOKWORK

Assuming that the argument `n` is of type `Int`, the type of the function is `Int -> Int` **[2 marks]**. It computes factorials **[2 marks]**.

(4 marks)

(c) State four benefits of the Haskell type system.

BOOKWORK

Among the benefits of the Haskell type system are that:

- types suggest errors during development **[1 mark]**;
- types prevent errors during execution **[1 mark]**;
- types guide the development of functions **[1 mark]**;

- types provide useful documentation **[1 mark]**.

(4 marks)

(d) Consider the following Prolog clauses:

```
f([],A,[]):-number(A).

f([X|Y],A,[U|V]):-
      U is X*A,
      f(Y,A,V).
```

where `number(A)` evaluates to true if and only if the variable `A` is a number.

(i) What is the result of the query

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

BOOKWORK

```
?- f([1,2,3],2,X).
X = [2,4,6].
```

[2 marks]

(ii) Give a general description of the meaning of the predicate `f`.

BOOKWORK

`f(X,A,Y)` is true if `X` is a list of numbers and `Y` is the list containing each element of `X` multiplied by `A` (the function computes the scalar product of a vector and a scalar) **[2 marks]**.

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

BOOKWORK

A query which returns `false` is `?- f([],a,[]).` since `a` is not a number (other correct answers are fine) **[1 mark]**.

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

(6 marks)

(e) Explain the meaning of a *green cut* and a *red cut* in Prolog.

BOOKWORK

- **Green cut**. The use of the cut operator in Prolog is said to be a green cut if when removed the program remains correct **[2 marks]**.
- **Red cut**. The use of the cut operator in Prolog is said to be a red cut if when removed the program looses its correctness **[2 marks]**.

(4 marks)

(f) Given the Prolog program:

```
p(X) :- \+q(X).
q(a).
```

describe the behaviour of the query

```
?- p(b).
```

BOOKWORK

```
?- p(b)
  |
  |  p(X) :- \+q(X)
  |  [X/b]
  ↓
?- \+q(b)
  |
  |  +----------------------+
  |  | TRY UNNEGATED GOAL:  |
  |  | ?- q(b)              |
  |  | no                   |
  |  +----------------------+
  ↓
 yes
```

For steps in execution **[4 marks]**.

(4 marks)

Describe the behaviour of the same query when the clause `q(b)` is added to the program.

(4 marks)

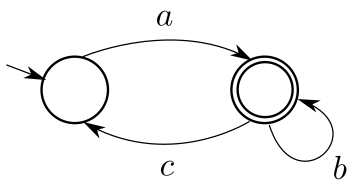
BOOKWORK

When the clause `q(b)` is added to the program we have

```
?- p(b)
  |
  |  p(X) :- \+q(X)
  |  [X/b]
  ↓
?- \+q(b)
  |
  |  +----------------------+
  |  | TRY UNNEGATED GOAL:  |
  |  | ?- q(b)              |
  |  |     |                |
  |  |     | q(b)    yes    |
  |  |     | []             |
  |  |     ↓                |
  |  +----------------------+
  ↓
 no
```

For steps in execution **[4 marks]**.

(g) Consider the following automaton.



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

$$\Lambda, ac, aca, abc, abcab, abbca.$$

BOOKWORK

Accepted strings: $aca, abcab, abbca$ **[1 mark]**.
Rejected string: Λ, ac, abc **[1 mark]**.

(2 marks)

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

BOOKWORK

The regular expression representing the language recognised by the automaton is $a(b^*ca)^*b^*$ **[2 marks]**.

(2 marks)

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

BOOKWORK

L_1 is a subset of L **[1 mark]**. In particular, if w is a string of L_1, then w has to start with a sequence of 'ac', which is compatible with the edges going from the initial state to the accepting state, and vice-versa. The last 'a' imposed by L_1 is compatible with the edge from the initial state to the accepting state of the automaton **[2 marks]**

(3 marks)

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

BOOKWORK

L_2 is not a subset of L **[1 mark]**, because strings belonging to L_2 end with a ‘c’, while strings belonging to L end either with an ‘a’ or a ‘b’ **[2 marks]**

(3 marks)

(Total 40 marks)

Question 2

(a) A playing card suit is one of `Clubs`, `Diamonds`, `Hearts` or `Spades`. Write a definition of a Haskell data type `Suit` for representing playing card suits that allows `Suit` values to be printed.

```
    data Suit = Clubs | Diamonds | Hearts | Spades
                deriving Show
```

For `data Suit` **[1 mark]**, for `Clubs`, `Diamonds`, `Hearts` and `Spades` **[1 mark]**, for `deriving Show` **[1 mark]**.

(3 marks)

(b) A Haskell programmer proposes that playing cards be represented by the type `(Int, Suit)`. For example, `(4, Clubs)`. Why might you advise them against doing so?

One would advise the programmer against doing so because the `Int` type allows many obviously illegal card values (eg. the pair `(2000, Hearts)`) **[1 mark]**. Moreover, the pair type does not identify values as cards **[1 mark]**, allowing confusion with values that are not cards **[1 mark]**, and so for errors at runtime that could be caught by the typechecker **[1 mark]**.

(4 marks)

(c) The Haskell platform comes with a function for producing random numbers in an (inclusive) range with the type

```
  randomRIO ::  (Int,Int) -> IO Int
```

What does the `IO` type signify, and how does it constrain the use of this function?

The `IO` type signifies that this function engages in Input/Output **[1 mark]** with the non-functional world **[1 mark]**, and so cannot be treated as a pure function **[1 mark]**. This constrains it to be used only from other functions with `IO` types **[1 mark]**.

(4 marks)

(d) Write a definition and the polymorphic type of a Haskell function that returns a pair of the nth item of a list, counting from 0, and a list of all the other items, so that

```
pick 1 ['a','b','c','d'] ===> ('b',['a','c','d'])
```

```
pick ::  Int -> [a] -> (a, [a])
pick n [x]    = (x, [])
pick 0 (x:xs) = (x, xs)
pick n (x:xs) = (y, x:ys)
                where (y, ys) = pick (n-1) xs
```

For each of three equations **[2 marks]**.

(6 marks)

(e) Write a definition of a Haskell function that shuffles a list by moving a randomly picked item to the front of the list several times, and has the type

```
shuffle ::  [a] -> IO [a]
```

```
shuffle ::  [a] -> IO [a]
shuffle xs = loop (length xs) xs

loop ::  Int -> [a] -> IO [a]
loop 0 xs = return xs
loop n xs =
   do rand <- randomRIO (1, length xs)
      let (y, ys) = pick rand xs
      loop (n-1) (y :  ys)
```

For loop **[4 marks]**, for random number generation **[2 marks]**, for pick at random point **[2 marks]**.

(8 marks)

(f) An programmer claims that, in practice, functional programs are no easier to understand that imperative ones. Do you agree? Argue one way or the other.

Either argue yes, the functional style encourages terse definitions and short variable names (which are sometimes omitted entirely). Often, general type signatures (which again are sometimes omitted entirely) provide the only commentary. The result is largely undocumented code that might once have be understood by its original author, but is much harder for anyone else to understand or extend, leading to a maintenance nightmare.

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

Or ague no, the functional style encourages terse definitions and short variable names that make it far easier to see all of the code for a particular task, and because of the absence of side effects, to understand what it does. All-of-a-sudden, one can see the "big picture". Type signatures provide the only commentary that one needs, and they are always up-to-date because they are checked by the compiler.

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

(5 marks)

(Total 30 marks)

Question 3

(a) Given the Prolog program:

```
sameperson(X,X).
sisters(X,Y) :- daughter(X,M,F),
                daughter(Y,M,F),
                \+sameperson(X,Y).
daughter(ann,bill,carol).
daughter(alice,bill,carol).
```

(i) Trace the behaviour of the program when run with the query

```
?- sisters(ann,alice).
```

```
?- sisters(ann,alice).
  |
  |  sisters(X,Y) :- daughter(X,M,F), daughter(Y,M,F), \+sameperson(X,Y).
  |  [X/ann,Y/alice]
  ↓
?- daughter(ann,M,F), daughter(alice,M,F), \+sameperson(ann,alice)
  |
  |  daughter(ann,bill,carol)
  |  [M/bill,F/carol]
  ↓
?- daughter(alice,bill,carol), \+sameperson(ann,alice)
  |
  |  daughter(alice,bill,carol)
  |  []
  ↓
?- \+sameperson(ann,alice)
  |
  |  +--------------------------------+
  |  | TRY UNNEGATED GOAL:            |
  |  | ?- sameperson(ann,alice)       |
  |  | no                             |
  |  +--------------------------------+
  ↓
 yes
```

For steps in execution **(5 marks)**.

(5 marks)

(ii) Trace the behaviour of the program when run with the query

```
?- sisters(ann,ann).
?- sisters(ann,ann).
 │
 │  sisters(X,Y) :- daughter(X,M,F), daughter(Y,M,F), \+sameperson(X,Y).
 │  [X/ann,Y/ann]
 ↓
?- daughter(ann,M,F), daughter(ann,M,F), \+sameperson(ann,ann)
 │
 │  daughter(ann,bill,carol)
 │  [M/bill,F/carol]
 ↓
?- daughter(ann,bill,carol), \+sameperson(ann,ann)
 │
 │  daughter(ann,bill,carol)
 │  []
 ↓
?- \+sameperson(ann,ann)
 │
 │  TRY UNNEGATED GOAL:
 │  ?- sameperson(ann,ann)
 │    │
 │    │  sameperson(X,X)
 │    │  [X/ann]
 │    ↓
 │   yes
 ↓
no
```

For steps in execution **(5 marks)**.

(5 marks)

(b) Consider the language L whose regular expression is $(ba)^+|(bab)^+$.

(i) Describe in words the language L.

(2 marks)

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

(4 marks)

(iii) Define a non-deterministic finite-state automaton recognising L. State where the non-determinism occurs. (**Hint:** the machine can non-deterministically decide on the first character whether the input will be either $(ba)^+$ or $(bab)^+$)

(6 marks)

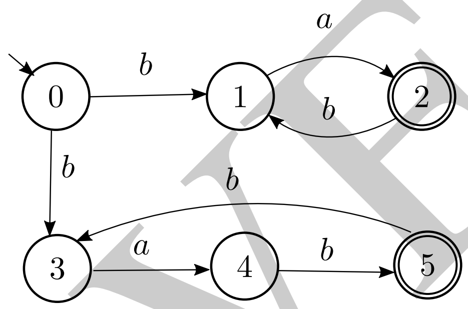
(iv) Transform the non-deterministic finite-state automaton recognising L above into a deterministic one.

(8 marks)

(i) L is the language whose strings are formed by sequences of ba or bab, and there is always at least one of them. **(2 marks)**

(ii) Strings belonging to L: ab, bab (**(1 mark)** each). Strings not belonging to L: Λ, aba (**(1 mark)** each).

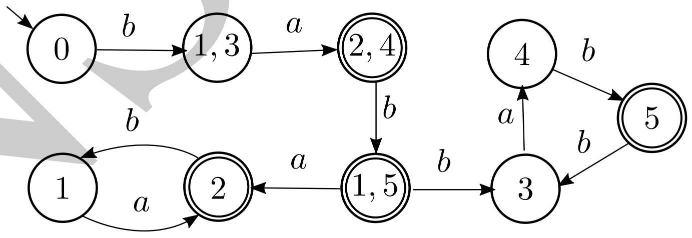
(iii) A non-deterministic automaton recognising L is:



The non-determinism occurs in the state 0 where the machine, when reading b, may move to state 1 or 3.

For the automaton states and transitions, upto **(6 marks)**, one of which is for stating correctly where the non-determinism is.

(iv) The deterministic automaton derived from the automaton above is:



For the automaton states and transitions, upto **[8 marks]**.

(Total 30 marks)