IMPERIAL COLLEGE LONDON

TIMED REMOTE ASSESSMENTS 2020-2021

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant assessments for the Associateship of the City and Guilds of London Institute

PAPER COMP40006

REASONING ABOUT PROGRAMS

Friday 30 April 2021, 10:00
Duration: 95 minutes
Includes 15 minutes for access and submission

Answer ALL TWO questions
Open book assessment

Paper contains 2 questions

1 This question is about proof by induction.

a The following code describes two Haskell functions, `rev` and `revTR`, which reverse lists; the latter is tail-recursive.

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

revTR :: [a] -> [a] -> [a]
revTR [] ys = ys
revTR (x:xs) ys = revTR xs (x:ys)
```

i) Write out the result of

```
revTR (1:3:5:[]) (2:4:[])
```

(You do not need to show any intermediate steps.)

ii) Prove that

(*) $\forall$`xs : [a].` $\forall$`ys : [a].` $\forall$`zs : [a].`

`[ revTR (xs++ys) zs  = revTR  ys ((rev xs)++zs) ]`

Write what is to be shown, what is taken arbitrary and justify each step.

You may want to use some of the following properties of lists, which hold for all `u:a`, all `us : [a]`, all `vs : [a]`, and all `ws : [a]`.

(A) `us++(vs++ws) = (us++vs)++ws`
(B) `u:us = [u]++us`
(C) `us++[] = us`
(D) `[]++us = us`
(E) `rev (us++vs) = (rev vs)++(rev us)`

iii) Prove that

(**) $\forall$`xs:[a].[ revTR  xs  [] = rev  xs ]` b Consider the following three definitions of Haskell data types, where `Exp` describes a simple language of expressions, `TypeT` is meant to represent integer and boolean types, and `Val` is meant to represent values.

```
data Exp = Cond Exp Exp Exp | BoolE Bool | IntE Int
data TypeT = IntT | BoolT
data Val = IntV Int | BoolV Bool
```

The following three relations

$\mathit{EType} \subseteq \texttt{Exp} \times \texttt{TypeT}$	describes the type of an expression,
$\mathit{VType} \subseteq \texttt{Val} \times \texttt{TypeT}$	describes the type of a value, and
$\mathit{EVal} \subseteq \texttt{Exp} \times \texttt{Val}$	describes the value of an expression.

are defined below:

(R1) $\forall \texttt{i}:\texttt{Int}.\, \mathit{EType}(\texttt{IntE i}, \texttt{IntT})$

(R2) $\forall \texttt{b}:\texttt{Bool}.\, \mathit{EType}(\texttt{BoolE b}, \texttt{BoolT})$

(R3) $\forall \texttt{e1}, \texttt{e2}, \texttt{e3} : \texttt{Exp}.\, \forall \texttt{t} : \texttt{TypeT}.$
$$[\ \mathit{EType}(\texttt{e1}, \texttt{BoolT}) \wedge \mathit{EType}(\texttt{e2}, \texttt{t}) \wedge \mathit{EType}(\texttt{e3}, \texttt{t}) \rightarrow \mathit{EType}(\texttt{Cond e1 e2 e3}, \texttt{t})\ ]$$

(R4) $\forall \texttt{i}:\texttt{Int}.\, \mathit{VType}(\texttt{IntV i}, \texttt{IntT})$

(R5) $\forall \texttt{b}:\texttt{Bool}.\, \mathit{VType}(\texttt{BoolV b}, \texttt{BoolT})$

(R6) $\forall \texttt{i}:\texttt{Int}.\, \mathit{EVal}(\texttt{IntE i}, \texttt{IntV i})$

(R7) $\forall \texttt{b}:\texttt{Bool}.\, \mathit{EVal}(\texttt{BoolE b}, \texttt{BoolV b})$

(R8) $\forall \texttt{e1}, \texttt{e2}, \texttt{e3} : \texttt{Exp}.\, \forall \texttt{v2}, \texttt{v3} : \texttt{Val}$
$$[\ \mathit{EVal}(\texttt{e1}, \texttt{BoolV true}) \wedge \mathit{EVal}(\texttt{e2}, \texttt{v2}) \wedge \mathit{EVal}(\texttt{e3}, \texttt{v3}) \rightarrow \mathit{EVal}(\texttt{Cond e1 e2 e3}, \texttt{v2})\ ]$$

(R9) $\forall \texttt{e1}, \texttt{e2}, \texttt{e3} : \texttt{Exp}.\, \forall \texttt{v2}, \texttt{v3} : \texttt{Val}$
$$[\ \mathit{EVal}(\texttt{e1}, \texttt{BoolV false}) \wedge \mathit{EVal}(\texttt{e2}, \texttt{v2}) \wedge \mathit{EVal}(\texttt{e3}, \texttt{v3}) \rightarrow \mathit{EVal}(\texttt{Cond e1 e2 e3}, \texttt{v3})\ ]$$

i) Consider the expression `e` defined as:

$\texttt{e} \triangleq \texttt{Cond (BoolE false) (IntE 3) (Cond (BoolE true) (IntE 4) (IntE 5))}$

Give a value $\texttt{v} \in \texttt{Val}$ such that $\mathit{EVal}(\texttt{e}, \texttt{v})$ holds.
(You do not need to show any intermediate steps or justify your answer.)

ii) Write an expression $\texttt{e}' \in \texttt{Exp}$ for which there exists *no* type $\texttt{t} \in \texttt{TypeT}$ for which $\mathit{EType}(\texttt{e}, \texttt{t})$ holds. (You do not need to justify your answer.)

iii) Based on the definition of *EType*, write the inductive principle that would allow you to prove:

(***) $\forall \texttt{e} : \texttt{Exp}.\, \forall \texttt{t} : \texttt{TypeT}.$
$$[\ \mathit{EType}(\texttt{e}, \texttt{t}) \rightarrow \exists \texttt{v}.[\ \mathit{EVal}(\texttt{e}, \texttt{v}) \wedge \mathit{VType}(\texttt{v}, \texttt{t})\ ]\ ]$$

The two parts carry, respectively, 60% and 40% of the marks.

2 This is a question about loops and method calls.

Consider the Java method `split(char[] in, char c)` defined as:

```
char[][] split( char[] in, char c )
// PRE: in ≠ null                                                              (P)
// POST: ∃k:ℕ.[Occurs( in[..], c ) = k ∧ in[..] ≈ Flatten( r[..], c, k ) : r[k] ] ∧ in ≈ in[..]_pre   (Q)
{
  int start = 0;
  int pos = 0;
  int found = 0;
  char[][] out = new char[in.length+1][];

  // INV: ???                                                                  (I)
  // VAR: ???                                                                  (V)
  while (pos < in.length){
    if ( in[pos] == c ){
      out[found] = slice(in, start, pos);
      found++;
      start = pos + 1;
    }
    pos++;
  }

  // MID: ???                                                                  (M1)
  out[found] = slice(in, start, pos);
  // MID: ???                                                                  (M2)
  return out;
}
```

This method splits up a provided string (treated as a character array) into an array of substrings that were delimited by the provided character `c` in the original string. The method makes use of an auxiliary library method `slice` that creates a partial copy of a provided array. The implementation of the `slice` method is not known, but it is claimed that it satisfies the following specification:

```
char[] slice(char[] str, int start, int finish)
//PRE: str ≠ null ∧ 0 ≤ start ≤ finish ≤ str.length
//POST: r[..] ≈ str[start..finish) ∧ str[..] ≈ str[..]_pre
{ ... }
```

The specification of the `split` method relies on the following functions for array slices:

$$Occurs(\, a[..], v \,) \;\triangleq\; |\{\, k \mid \mathtt{a}[k] = v \,\}|$$

$$Flatten(\, a[..], v, k \,) \;\triangleq\; \begin{cases} [\,] & \text{if } k = 0 \\ Flatten(\, a[..], v, k-1 \,) : a[k-1] : v & \text{otherwise} \end{cases}$$

where $Flatten(\, a[..], v, k \,)$ converts k elements of a two-dimensional array a into a one-dimensional array interleaved with the element v. For example:

$$Flatten(\mathtt{[\,['a','b'],['c']\,]},\mathtt{'-'},2) = \mathtt{['a','b','-','c','-']}$$

a i) Write the result of evaluating
Occurs(`['w','a','a','t','?']`, `'a'`).

ii) Write out the state of the whole array **r** returned from running the code
`split(['w','a','a','t','?'], 'a')`.

[***Note:*** *You may assume that* `new char[x][];` *creates a new two-dimensional character array whose outer length is* `x`*, with all of its contents set to* `null`*.*]

b Unfortunately, the author has not fully specified the `split` method.

i) Give mid-conditions M_1 and M_2 that are strong enough to prove partial correctness of the code. (You do *not* need to prove anything.)

ii) Give an invariant I for the loop that is appropriate to show total correctness. (You do *not* need to prove anything.)

[***Hint:*** *The invariant should have four conjuncts: the first should bound and relate the values of* `start` *and* `pos`*; the second should describe the contents of the array* `in`*; the third should define the value of* `found`*; and the last should relate the contents of the two-dimensional array* `out` *with the array* `in`*.*]

iii) Give a variant V for the loop that is appropriate to show termination. (You do *not* need to prove anything.)

c Prove that the body of the loop in the `split` method re-establishes your invariant from part b.ii) in an iteration where `in[pos]` $=$ `c`.
State clearly what is given and what you need to show.

d On line 8, the `split` method creates a two-dimensional character array `out` whose outer length is one element more than the length of the input array `in`.

Could we save space by creating the two-dimensional character array `out` with a smaller outer length, without compromising the correctness of the method?

Justify your answer and provide a worst case example input to the `split` method that requires the most space in the two-dimensional character array `out`.

The four parts carry, respectively, 10%, 35%, 45%, and 10% of the marks.