IMPERIAL COLLEGE LONDON

TIMED REMOTE ASSESSMENTS 2021-2022

BEng Honours Degree in Mathematics and Computer Science Part I
MEng Honours Degree in Mathematics and Computer Science 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 COMP40012

LOGIC AND REASONING

Friday 6 May 2022, 10:00
Writing time: 80 minutes
Upload time: 25 minutes

Answer ALL TWO questions
Open book assessment

This time-limited remote assessment has been designed to be open book. You may use resources which have been identified by the examiner to complete the assessment and are included in the instructions for the examination. You must not use any additional resources when completing this assessment.

The use of the work of another student, past or present, constitutes plagiarism. Giving your work to another student to use constitutes an offence. Collusion is a form of plagiarism and will be treated in a similar manner. This is an individual assessment and thus should be completed solely by you. The College will investigate all instances where an examination or assessment offence is reported or suspected, using plagiarism software, vivas and other tools, and apply appropriate penalties to students. In all examinations we will analyse exam performance against previous performance and against data from previous years and use an evidence-based approach to maintain a fair and robust examination. As with all exams, the best strategy is to read the question carefully and answer as fully as possible, taking account of the time and number of marks available.

Paper contains 2 questions

1 a Let ϕ, ψ and ρ be arbitrary propositional formula. Rewrite the following formula into DNF:

$$(\phi \to \psi) \wedge \neg(\neg\rho \vee \psi)$$

Specify at each step the transformation rule applied.

b Consider a variant $\vdash^*$ of the natural deduction system in which $\vee$-introduction is removed and the following rule is added instead:

$$\begin{array}{lll} 1 & A & \\ 2 & \neg(\neg A \wedge \neg B) & \neg\wedge I(1) \end{array}$$

Is $\vdash^*$ sound? Justify your answer.

c Let L be the 2-sorted signature with sorts $\texttt{Nat}$ and $[\texttt{Nat}]$, containing constants $\underline{0}, \underline{1}, \underline{2}, \ldots : \texttt{Nat}$ and $[\,] : [\texttt{Nat}]$, function symbols $+$, $-$, $\times$, $:$, $++$, $!!$, $\sharp$, and relation symbols $<, \leq$ and $=$ of the appropriate sorts. The intended semantics is a structure whose domain consists of the natural numbers $0, 1, 2, \ldots$ (sort $\texttt{Nat}$) and all lists of natural numbers (sort $[\texttt{Nat}]$). The symbols have the usual meanings (as in lectures). For example, $ys = (x : xs)$ holds iff ys is the list including x as first element, followed by all the elements of xs in the same order. Variables x, y, z, m, n, etc. have sort $\texttt{Nat}$, and xs, ys, etc. have sort $[\texttt{Nat}]$.

Complete in logic the definitions (i) - (iv) below, the underlined parts of which are informally described in English.

i) $\forall x{:}\texttt{Nat}.\forall xs{:}[\texttt{Nat}][\texttt{smallest}(x, xs) \leftrightarrow \underline{x\text{ is the smallest value in }xs}]$

ii) $\forall x{:}\texttt{Nat}.\forall xs{:}[\texttt{Nat}][\texttt{twice}(x, xs) \leftrightarrow \underline{\text{the value }x\text{ occurs at least twice in }xs}]$

iii) $\forall xs, ys{:}[\texttt{Nat}][\texttt{dups}(xs, ys) \leftrightarrow \underline{\text{all values in }xs\text{ occur at least twice in }ys}]$

(**Hint:** make use of the relation `twice` given in part (ii))

iv) $\forall xs{:}[\texttt{Nat}][\texttt{twoReps}(xs) \leftrightarrow$
$\underline{\text{there are exactly two values that are repeated in }xs}]$

(**Hint:** again, make use of the relation `twice` given in part (ii))

Consider data structures `Tree` (for trees), and `TShape` (for tree shapes):

```
data Tree = Leaf Char | Node Tree Tree
data TShape = LShape | NShape TShape TShape
```

The function `split` splits a tree into a tree shape and a list of characters:

```
split :: Tree -> ( TShape x [Char] )

split Leaf c  = ( LShape, [c] )
split (Node t1 t2) = ( (NShape ht1 ht2), cs1++cs2 )
    where
    split t1 = ( ht1, cs1 )
    split t2 = ( ht2, cs2 )
```

The function `zip` reconstructs a tree out of a tree-shape and a list of characters:

```
zip :: TShape -> [Char] -> ( Tree x [Char] )

zip LShape c:cs = ( Leaf c, cs )
zip (NShape ht1 ht2) cs = ( (Node t1 t2), cs2 )
    where
    zip ht1 cs  = t1 cs1
    zip ht2 cs1 = t2 cs2
```

d Write the result of executing:

i) `split (Node (Node (Leaf 'c') (Leaf 'r')) Leaf 'y')`

and the result of executing:

ii) `zip (NShape LShape (NShape LShape LShape)) ['b','y','e']`

e We would like to prove:

(D) $\forall \mathtt{st} : \mathtt{TShape}. \forall \mathtt{cs} : \mathtt{[Char]}. \forall \mathtt{t} : \mathtt{Tree}.$

$$[\, \mathtt{zip\ st\ cs} = (\mathtt{t}, \mathtt{[]}) \;\longrightarrow\; \mathtt{split\ t} = (\mathtt{st}, \mathtt{cs}) \,]$$

However, (D) cannot be proven by induction. Write down a stronger assertion, (D'), which implies (D), and which can be proven by induction.
You do not need to prove anything.

f Prove:

(E) $\forall \mathtt{st} : \mathtt{TShape}. \forall \mathtt{cs} : \mathtt{[Char]}. \forall \mathtt{t} : \mathtt{Tree}.$

$$[\, \mathtt{split\ t} = (\mathtt{st}, \mathtt{cs}) \;\longrightarrow\; \mathtt{zip\ st\ cs} = (\mathtt{t}, \mathtt{[]}) \,]$$

State what is given, what is to be shown, what is taken arbitrary, justify your proof steps and state where you instantiate universally quantified variables.

The six parts carry, respectively, 15%, 10%, 25%, 5%, 10%, and 35% of the marks.

2 This is a question about loops and method calls.

Consider the Java method `abbrvts(char[] str)` defined as:

```
int abbrvts( char[] str )
// PRE: str ≠ null ∧ ???                            (P)
// POST: Abbreviates( str[..)pre, str[..r) )        (Q)
{
  int cnt = 1;
  int pos = 1;
  // INV: ???                                        (I)
  // VAR: ???                                        (V)
  while (cnt < str.length){
    if ( !isVowel(str[cnt]) ){
      str[pos] = str[cnt];
      pos++;
    }
    cnt++;
  }
  // MID: ???                                        (M)
  return pos;
}
```

This method abbreviates a provided string `str` (treated as a character array) in-place removing all of the vowels from the array, except for the first element. The method makes use of an auxiliary library method `isVowel` that returns `true` if the provided character is a vowel and `false` otherwise. The implementation of the `isVowel` method is not known, but it is claimed that it satisfies the following specification:

```
char[] isVowel(char c)
//PRE: true
//POST: r ⟷ Vowel(c)
{ ... }
```

The specifications of the `abbrvts` and `isVowel` methods rely on the following predicates for characters and array-slices:

$$Vowel(c) \triangleq c \in \{\mathtt{a}, \mathtt{A}, \mathtt{e}, \mathtt{E}, \mathtt{i}, \mathtt{I}, \mathtt{o}, \mathtt{O}, \mathtt{u}, \mathtt{U}\}$$

$$\begin{aligned} Abbreviates(\, a[\,..\,y_1), b[\,..\,y_2)\,) \triangleq\ & y_1 \leq 0 \ \wedge\ y_2 \leq 0 \\ & \vee\ a\,[0] = b\,[0] \ \wedge\ Abbrv(\, a[1\,..\,y_1), b[1\,..\,y_2)\,) \end{aligned}$$

$$\begin{aligned} Abbrv(\, a[x_1\,..\,y_1), b[x_2\,..\,y_2)\,) \triangleq\ & y_1 \leq x_1 \ \wedge\ y_2 \leq x_2 \\ & \vee\ Vowel(a\,[x_1]\,) \ \wedge\ Abbrv(\, a[x_1{+}1\,..\,y_1), b[x_2\,..\,y_2)\,) \\ & \vee\ \neg Vowel(a\,[x_1]\,) \ \wedge\ a\,[x_1] = b\,[x_2] \\ & \qquad \wedge\ Abbrv(\, a[x_1{+}1\,..\,y_1), b[x_2{+}1\,..\,y_2)\,) \end{aligned}$$

where $Abbreviates(\, a[\,..\,y_1), b[\,..\,y_2)\,)$ states that the array-slice $b[\,..\,y_2)$ is an abbreviation of the array-slice $a[\,..\,y_1)$. For example:

- $Abbreviates(\,[\mathtt{h,e,l,l,o,t,h,e,r,e}], [\mathtt{h,l,l,t,h,r}]\,)$ is true.
- $Abbreviates(\,[\mathtt{e,l,l,o,g,u,v}], [\mathtt{l,l,g,v}]\,)$ is false.

a Write out the value of `len` and state of the array-slice `str`$[..$`len`$)$ after running the code `len = abbrvts(str)` for the following initial values of `str`.

i) `str = [H,a,s,k,e,l,l]`

ii) `str = [A,-,l,e,v,e,l,s]`

b In their rush to finish the `abbrvts` method in time for this exam, the author forgot to complete its pre-condition P.

i) Give an example `char[]` input `str` where running `abbrvts(str)` will **not** satisfy the post-condition Q and briefly explain your choice.

ii) Complete the precondition P for `abbrvts` so that it rules out your example from part b.i) and ensures the expected behaviour of the method.

c Unfortunately, the author has also not fully specified the `abbrvts` method.

i) Write a mid-condition M which holds immediately after the loop has terminated and is strong enough to prove partial correctness of the code. (You do *not* need to prove anything.)

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

[***Hint:*** *The invariant should have three conjuncts: the first should bound and relate the values of* `cnt` *and* `pos`*; the second should describe the modified part of the array* `str`*; and the last should describe the unmodified part of the array.*]

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

d Prove that the body of the loop in the `abbrvts` method re-establishes your invariant from part c.ii) in an iteration where $Vowel(\texttt{str[cnt]}) = false$. State clearly what is given and what you need to show.

You may use the following Lemma without proof:

$$(\mathbf{Ab+}) : \begin{array}{l} \forall a, b, y_1, y_2. \\ \left[\begin{array}{l} 1 \leq y_1 < a\texttt{.length} \land 1 \leq y_2 < b\texttt{.length} \land a\texttt{[}y_1\texttt{]} = b\texttt{[}y_2\texttt{]} \\ \land\ Abbreviates(\, a[\,..\,y_1),\, b[\,..\,y_2)\,) \land \neg Vowel(a\texttt{[}y_1\texttt{]}) \\ \quad \longleftrightarrow \\ Abbreviates(\, a[\,..\,y_1{+}1),\, b[\,..\,y_2{+}1)\,) \end{array} \right] \end{array}$$

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