PAPER C141=MC141

REASONING ABOUT PROGRAMS

Wednesday 29 April 2020, 11:00
Duration: 80 minutes
Post-processing time: 30 minutes
Answer TWO questions

While this time-limited remote assessment has not been designed to be open book, in the present circumstances it is being run as an open-book examination. We have worked hard to create exams that assesses synthesis of knowledge rather than factual recall. Thus, access to the internet, notes or other sources of factual information in the time provided will not be helpful and may well limit your time to successfully synthesise the answers required.
Where individual questions rely more on factual recall and may therefore be less discriminatory in an open book context, we may compare the performance on these questions to similar style questions in previous years and we may scale or ignore the marks associated with such questions or parts of the questions. 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 This question is about proof by induction.

Consider the following definition of an infix operator <> which concatenates pairs of lists:

```
(<>)  ::  ([a],[b])  ->  ([a],[b]) ->   ([a],[b])
(xs,ys) <> (xs',ys') = (xs++xs', ys++ys')
```

For example, the term
`([1,2],['a','b']) <> ([3,4,5],['c','d','e'])`
is equal to
`([1,2,3,4,5],['a','b','c','d','e'])` .

For all `xs` : `[a]`, `ys` : `[b]`, all `x` : `a`, `y` : `b`, and all `zs`, `zs'`, `zs''` : `[(a,b)]`, the following properties hold:

(A) `zs<>([],[]) = zs = ([],[])<>zs`
(B) `(zs<>zs')<>zs'' = zs<>(zs'<>zs'')`

You do not need to prove the properties from above, but may want to use some of these properties in parts c and d below.

a Consider the following definition of the function `split` which splits a list of pairs into a pair of lists:

```
split :: [(a,b)] -> ([a],[b])
split []       = ([],[])
split (x,y):zs = ([x],[y]) <> (split zs)
```

Write out the result of

```
split [(1,'a'),(2,'b')]
```

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

b Consider the following definition of the tail-recursive function `splitTR` which also splits a list of pairs into a pair of lists:

```
splitTR ::  ([a],[b])  -> [a] -> [b] ->  ([a],[b])
splitTR []       xs ys = (xs,ys)
splitTR (x,y):zs xs ys = splitTR zs (xs++[x]) (ys++[y])
```

Write out the result of

```
splitTR [(5,'e'),(2,'b')] [3,6] ['c','f']
```

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

c Prove that

$\forall$zs : [(a,b)].$\forall$xs : [a].$\forall$ys : [b].
[splitTR zs xs ys = (xs,ys)<>(split zs)]

Write what is to be shown, state which variables are taken arbitrary, and justify each step.

d Prove that

$\forall$zs : [(a,b)].
[splitTR zs [] [] = split zs]

e Consider the following definition of function F.

```
F :: (Int, Int, Int, Int) -> Int
F (m,n,i,j)
      | m+i==n     = i
      | n+j==m     = j
      | otherwise  = 5 * ( F (m,n,i+2,j+3)  )
```

Assume some predicate $Q \subseteq$ Int $\times$ Int $\times$ Int $\times$ Int $\times$ Int.
Based on the definition of F, write the inductive principle which allows us to prove that

$$\forall \mathtt{m},\mathtt{n},\mathtt{i},\mathtt{j},\mathtt{r} : \mathtt{Int}.\, [\, \mathtt{r} = \mathtt{F}(\mathtt{m},\mathtt{n},\mathtt{i},\mathtt{j}) \longrightarrow Q(\mathtt{m},\mathtt{n},\mathtt{i},\mathtt{j},\mathtt{r})\,]$$

Note: Just in case you are temporarily rusty with Haskell syntax: F returns its third argument if the second argument and the sum of first and third argument are equal. It returns its fourth argument if the first argument and the sum of second and last argument are equal. And otherwise, it increments the third argument by 2, increments the fourth argument by 3, recurses, and multiplies the result by 5.

The five parts carry, respectively, 5%, 5%, 60%, 5%, and 25% of the marks.

2 This is a question about loops.

Consider the predicate $Wb(\,\texttt{a}[..]\,)$ which defines well-bracketed strings via array slices as follows:

$$
\begin{array}{rcl}
Wb(\,\texttt{a}[..]\,) & \triangleq & \forall c \in \texttt{a}[..].[\,Nb(c)\,] \\
& & \quad \vee\ \exists \texttt{a}_1, \texttt{a}_2, \texttt{a}_3 \in \texttt{char}[\,].\, \exists c_1, c_2 \in \texttt{char}. \\
& & \qquad \left[\begin{array}{l} \texttt{a}[..] \approx \texttt{a}_1[..] : c_1 : \texttt{a}_2[..] : c_2 : \texttt{a}_3[..] \ \wedge\ BPair(c_1, c_2) \\ \ \wedge\ Wb(\,\texttt{a}_1[..]\,) \wedge Wb(\,\texttt{a}_2[..]\,) \wedge Wb(\,\texttt{a}_3[..]\,) \end{array}\right] \\
Ob(c) & \triangleq & c \in \{\,\texttt{'('}, \texttt{'['}\,\} \\
Cb(c) & \triangleq & c \in \{\,\texttt{')'}, \texttt{']'}\,\} \\
Nb(c) & \triangleq & \neg(\,Ob(c) \vee Cb(c)\,) \\
BPair(c_1, c_2) & \triangleq & (\,c_1 = \texttt{'('} \wedge c_2 = \texttt{')'}\,) \vee (\,c_1 = \texttt{'['} \wedge c_2 = \texttt{']'}\,)
\end{array}
$$

The strings "`a[2(x+y)]`" and "`a[0]+f()`" and "`abc`" are all well-bracketed, but the strings "`(b[]`" and "`a[])`" and "`(a)x]+1`" are not well-bracketed.

The following Java method `wellBr(char[] str)` determines if an input character array represents a well-bracketed string:

```
1  public static boolean wellBr(char[] str)
2  // PRE: str ≠ null                                                    (P)
3  // POST: str[..] ≈ str[..]pre  ∧  r ⟷ Wb(str[..])                    (Q)
4  {
5    int i = 0;
6    char[] stack = new char[str.length];
7    int j = stack.length;
8    // INV: ???                                                         (I)
9    // VAR: ???                                                         (V)
10   while (i < str.length){
11     char c = str[i];
12     // MID: I ∧ ???                                                  (M1)
13     if ( c == '(' || c == '[' ){
14       // MID: I ∧ ???                                                (M2)
15       if ( c == '(' ){ stack[--j] = ')'; }
16       if ( c == '[' ){ stack[--j] = ']'; }
17     } else if ( c == ')' || c == ']' ){
18       // MID: I ∧ ???                                                (M3)
19       if ( j == stack.length || c != stack[j] ){
20         return false;
21       }
22       j++;
23     }
24     i++;
25   }
26   // MID: str[..] ≈ str[..]pre ∧ ( j = stack.length ⟷ Wb(str[..]) )   (M4)
27   return j == stack.length;
28 }
```

The `wellBr` method uses an array called `stack` to track the expected brackets that will close each unmatched open-bracket character seen so far.

You may use the following Lemmas without proof throughout this question:

Lemma 1:

$$\forall \mathtt{a}_1, \mathtt{a}_2 \in \mathtt{char[]}.\, \forall c \in \mathtt{char}.\\ [\, Wb(\, \mathtt{a}[..]\,) \;\wedge\; Cb(c) \;\longrightarrow\; \neg Wb(\, \mathtt{a}_1[..] : c : \mathtt{a}_2[..]\,)\,]$$

Lemma 2:

$$\forall \mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3 \in \mathtt{char[]}.\, \forall c_1, c_2 \in \mathtt{char}.\\ \left[\begin{array}{l} Wb(\, \mathtt{a}_1[..] : c_1 : \mathtt{a}_2[..]\,) \;\wedge\; Cb(c_1) \;\wedge\; Cb(c_2) \;\wedge\; c_1 \neq c_2 \\ \qquad\qquad\qquad\qquad\qquad\qquad \longrightarrow\; \neg Wb(\, \mathtt{a}_1[..] : c_2 : \mathtt{a}_3[..]\,) \end{array}\right]$$

You may also assume that `char[] stack = new char[str.length];` creates a new character array that is the same length as the input `str` array.

a Unfortunately, the author has not fully specified the loop of the method.

i) Complete the invariant I for the loop so that it is appropriate to show total correctness. (You do *not* need to prove anything.)

[**Hint:** *We suggest you add six further conjuncts: the first should describe the contents of the array* `str`*; the next three should bound and relate* `i` *and* `j`*; the fifth should describe the well-bracketedness of the array seen so far and the last should describe the known contents of the array* `stack`*.*
Use mid-condition M_4 to guide you and when relating `i` *and* `j` *consider the safety of accessing* `stack` *at position* `j`*.*]

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

iii) Complete the mid-conditions M_1, M_2 and M_3 so that they are strong enough to prove partial correctness of the code. (You do *not* need to prove anything.) (You may also refer to the entire loop invariant as I.)

b Show that if `wellBr` terminates early on line 20, then its post-condition Q is established. State clearly what is given and what you need to show.

[**Hint:** *When considering the* `if` *statement condition on line 19, remember that Java evaluates boolean expressions lazily and hence the second disjunct will only be considered if the first evaluates to* `false`*.*]

c On line 7 of the `wellBr` method we create an array `stack` that is the same length as the input array `str`. Could we save space by creating a smaller array without compromising the correctness of the method? Justify your answer and provide a worst case example input that requires the most space in `stack`.

The three parts carry, respectively, 50%, 40%, and 10% of the marks.