INFR08013 INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Tuesday 15th December 2015
09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

1. Note that ALL QUESTIONS ARE COMPULSORY.

2. DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.

3. This is an OPEN BOOK examination: notes and printed material are allowed, and USB sticks (read only), but no electronic devices.

4. CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Convener: D. K. Arvind
External Examiner: C. Johnson

THIS EXAMINATION WILL BE MARKED ANONYSOMOUSLY
1. (a) Write a function \( p : \text{[Int]} \rightarrow \text{Int} \) that takes a list of time durations in minutes and calculates what hour it is after all those periods of time have passed, \textit{ignoring negative durations}, starting at 1 o’clock and using a 12-hour clock. For example:

\[
\begin{align*}
p \; [\; ] & \; = \; 1 \\
p \; [-30,-20] & \; = \; 1 \\
p \; [20,-30,30,14,-20] & \; = \; 2 \\
p \; [200,45] & \; = \; 5 \\
p \; [60,-100,360,-20,240,59] & \; = \; 12 \\
p \; [60,-100,360,-20,240,60] & \; = \; 1 
\end{align*}
\]

Use basic functions, list comprehension, and library functions, but \textit{not recursion}. Credit may be given for indicating how you have tested your function. \[12 \text{ marks}\]

(b) Write a second function \( q : \text{[Int]} \rightarrow \text{Int} \) that behaves like \( p \), this time using basic functions and \textit{recursion}, but \textit{not list comprehension} or library functions. Credit may be given for indicating how you have tested your function. \[12 \text{ marks}\]

(c) Write a third function \( r : \text{[Int]} \rightarrow \text{Int} \) that also behaves like \( p \), this time using one or more of the following higher-order library functions:

\[
\begin{align*}
\text{map} & \; : \; (a \rightarrow b) \rightarrow [a] \rightarrow [b] \\
\text{filter} & \; : \; (a \rightarrow \text{Bool}) \rightarrow [a] \rightarrow [a] \\
\text{foldr} & \; : \; (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b
\end{align*}
\]

Do \textit{not} use recursion or list comprehension. Credit may be given for indicating how you have tested your function. \[12 \text{ marks}\]
2. (a) Write a function \( f :: \text{String} \to \text{String} \) that removes all but one occurrence of consecutive repeated characters. For example:

\[
\begin{align*}
  f \ "\text{Tennessee}" & = \ "\text{Tenese}" \\
  f \ "\text{llama}" & = \ "\text{lama}" \\
  f \ "\text{oooh}" & = \ "\text{oh}" \\
  f \ "\text{none here}" & = \ "\text{none here}" \\
  f \ "\text{nNnor hEere}" & = \ "\text{nNnor hEere}" \\
  f \ "\text{A}" & = \ "\text{A}" \\
  f \ "" & = \ ""
\end{align*}
\]

Upper/lower case should be taken into account when comparing characters, as these examples show.

Use basic functions, list comprehension, and library functions, but not recursion. Credit may be given for indicating how you have tested your function. \([16 \text{ marks}]\)

(b) Write a second function \( g :: \text{String} \to \text{String} \) that behaves like \( f \), this time using basic functions and recursion, but not list comprehension or library functions. Credit may be given for indicating how you have tested your function. \([16 \text{ marks}]\)
3. The following data type represents a simplified form of regular expressions which omits the “star” (repetition) operator:

\[
\text{data Regexp} = \begin{cases} 
  \text{Epsilon} & \text{-- empty} \\
  \text{Lit Char} & \text{-- character literal} \\
  \text{Seq Regexp Regexp} & \text{-- sequence: } r \ s \\
  \text{Or Regexp Regexp} & \text{-- choice: } r \ | \ s 
\end{cases}
\]

Recall that every regular expression describes a set of strings (its “language”), where:

- the regular expression \( \varepsilon \) describes only the empty string;
- for any character \( A \), the regular expression \( A \) describes only the string containing the single character \( A \);
- the regular expression \( r s \) describes all strings consisting of a first part that is described by \( r \) followed by a second part that is described by \( s \); and
- the regular expression \( r|s \) describes all strings that are either described by \( r \) or by \( s \).

The template file includes a function \( \text{showRegexp :: Regexp} \rightarrow \text{String} \) which converts regular expressions into a readable format, and code that enables QuickCheck to generate arbitrary values of type \( \text{Regexp} \), to aid testing.

The template file also contains the following regular expressions for you to use in testing:

\[
\begin{align*}
  r1 &= \text{Seq} (\text{Lit 'A'}) (\text{Or} (\text{Lit 'A'}) (\text{Lit 'A'})) \quad -- A(A|A) \\
  r2 &= \text{Seq} (\text{Or} (\text{Lit 'A'}) \text{Epsilon}) \\
       &\quad (\text{Or} (\text{Lit 'B'})) \quad -- (A|e)(A|B) \\
  r3 &= \text{Seq} (\text{Or} (\text{Lit 'A'}) (\text{Seq Epsilon (Lit 'A')}) \\
       &\quad (\text{Or} (\text{Lit 'B'})) \quad -- (A(eA))(A|B) \\
  r4 &= \text{Seq} (\text{Or} (\text{Lit 'A'}) (\text{Seq Epsilon (Lit 'A')}) \\
       &\quad (\text{Seq (Or (Lit 'A') (Lit 'B')) Epsilon}) \quad -- (A(eA))(A|B)e \\
  r5 &= \text{Seq} (\text{Seq Epsilon Epsilon}) \\
       &\quad (\text{Or Epsilon (Lit 'B')}) \quad -- ((A(eA))(e|B))(A|B)e \\
  r6 &= \text{Seq (Seq Epsilon Epsilon)} \\
       &\quad (\text{Or Epsilon Epsilon}) \quad -- (ee)(e|e)
\end{align*}
\]
(a) Write a function `language :: Regexp -> [String]` which, given a regular expression, returns its language in the form of a list without duplicates. For example, referring to the test examples above:

```
language r1 = ["AA"]  -- A(A|A)
language r2 = ["AA","AB","A","B"]  -- (A|e)(A|B)
language r3 = ["AA","AB"]  -- (A|(eA))(A|B)
language r4 = ["AA","AB"]  -- (A|(eA))((A|B)e)
language r5 = ["AA","AB","ABA","ABB"]  -- ((A|(eA))(e|B))((A|B)e)
language r6 = ["""]  -- (ee)(e|e)
```

Credit may be given for indicating how you have tested your function. (Hint: you will need to test using an equality on lists that disregards order but not repetitions. An appropriate function `equal` is provided in the template file.)

(b) Write a function `simplify :: Regexp -> Regexp` that converts a regular expression to an equivalent simpler regular expression by use of the following laws:

\[
\begin{align*}
\varepsilon & r = r \\
r \varepsilon & = r \\
r | r & = r 
\end{align*}
\]

For example:

```
simplify r1 = Seq (Lit 'A') (Lit 'A')  
    -- A(A|A) = AA
simplify r2 = r2  
    -- (A|e)(A|B) is already simplified
simplify r3 = Seq (Lit 'A') (Or (Lit 'A') (Lit 'B'))  
    -- (A|(eA))(A|B) = A(A|B)
simplify r4 = Seq (Lit 'A') (Or (Lit 'A') (Lit 'B'))  
    -- (A|(eA))((A|B)e) = A(A|B)
simplify r5 = Seq (Seq (Lit 'A') 
    (Or Epsilon (Lit 'B'))) 
    (Or (Lit 'A') (Lit 'B'))  
    -- ((A|(eA))(e|B))((A|B)e) = (A(e|B))(A|B)
simplify r6 = Epsilon  
    -- (ee)(e|e) = e
```

Credit may be given for indicating how you have tested your function.  

[16 marks]