CSL 102: Introduction to Computer Science

Minor 2 Mon 21 Mar 2005 VI 301(Gps 1-6)& VI 401(Gps 7-8) 9:30-10:30 Max Marks 40

1. Answer in the space provided on the question paper.
2. The answer booklet you have been given is for **rough work only**
3. The answer booklet will not be collected.

Q1	Q2	Q3	Q4	TOTAL
10	9	9	12	40

Note:

1. **You may use ML notation wherever necessary, in addition to the usual algorithmic notations that we use in lectures**
2. **Correctness is of paramount importance**
3. **The more time and space efficient your solution the more the marks you will be given**

1. For the

```
datatype 'a bintree =  Empty | Node of 'a * 'a bintree * 'a bintree
```

we defined the following preorder traversal function.

```
local
fun pre (Empty, L) = L
|   pre (Node (x, left, right), L) =
        let val M = pre (right, L);
            val N = pre (left, M)
        in  x::N
        end
in fun preorder2 (T) = pre (T, [])
end
```

State and prove a correctness theorem for the function `preorder2`.

(10 marks)

Solution.

Since `preorder2` depends upon the function `pre` we need to first prove a lemma for the correctness of `pre` and then a theorem for `preorder2` which will use the lemma.

Let $preorder(T)$ be the function that mathematically describes preorder traversal. It may be inductively defined as follows:

$$\begin{array}{lcl} preorder(\texttt{Empty}) & = & [] \\ preorder(\texttt{Node(x, left, right)}) & = & x :: preorder(\texttt{left})@preorder(\texttt{right}) \end{array}$$

Lemma. $\texttt{pre(T, L)} = preorder(T)@L$ for all binary trees `T` and lists `L`.

Proof. By induction on the structure of the tree `T`.

Basis. (`T = Empty`). Then $\texttt{pre(T, L)} = \texttt{L} = preorder(\texttt{T})@\texttt{L}$.

Induction Hypothesis. For all binary trees, `T`, of depth at most m, for some $m > 0$ and all lists `L`,

$$\boxed{\texttt{pre(T, L)} = preorder(T)@L}$$

Induction step. Let `T = Node (x, TL, TR)`, where the trees `TL` and `TR` are each of depth at most m. Then

$$\begin{array}{lll} & \texttt{pre(T, L)} & \\ = & \texttt{x} :: N & \\ = & \texttt{x} :: \texttt{pre(TL, M)} & \\ = & \texttt{x} :: \texttt{pre(TL, pre(TR, L))} & \\ = & \texttt{x} :: \texttt{pre(TL,} \, preorder(\texttt{TR})@\texttt{L}) & \text{Ind. hypothesis} \\ = & \texttt{x} :: (preorder(\texttt{TL})@(preorder(\texttt{TR})@L)) & \text{Ind. hypothesis} \\ = & \texttt{x} :: (preorder(\texttt{TL})@preorder(\texttt{TR}))@L & \text{Associativity of @} \\ = & (\texttt{x} :: (preorder(\texttt{TL})@preorder(\texttt{TR}))@L & \text{Property of :: and @} \\ = & preorder(\texttt{T})@L & \end{array}$$

QED

The property $(x :: L)@M = x :: (L@M)$ and the associativity of @ can be proven by induction on the lengths of lists. We leave their proofs to the interested reader.

Theorem. $\texttt{preorder2(T)} = preorder(\texttt{T})$ for all binary trees `T`.

Proof. $\texttt{preorder2(T)} = \texttt{pre(T, [])} = preorder(\texttt{T})@[] = preorder(\texttt{T})$.

QED

2. Consider a list of real numbers $[a_n, a_{n-1}, \ldots, a_0]$ with $a_n \neq 0.0$ representing the polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

in one variable x, where the empty list represents the value 0.0. Design a function `deriv L` which computes the derivative of the polynomial as a list of coefficients in the same order.

(9 marks)

Solution.

We directly give the ML version of the algorithm as it is a really simple problem in which the only challenge is to compute the derivative as we traverse through the list of coefficients. Otherwise, one would use the length of the list to find the coefficient.

The function `auxDeriv` here keeps track of the multiplicative factor for each coefficient.

```
local exception EmptyList;
      fun auxDeriv []     = raise EmptyList
        | auxDeriv [h]    = (0.0, 0)
        | auxDeriv (h::L) =
              let val (D, e) = auxDeriv L
               in (h*real(e+1))::D
              end

   in fun deriv [] = 0.0
        | deriv L  = #1(auxDeriv L)
  end
```

3. Consider the datatype

```
datatype poly = COEFF of real | VAR of string | POWER of poly * int |
                PLUS poly * poly | MULT poly * poly
```

which may be used to represent polynomials of several variables and in any form such as products of monomials or sums of monomial terms or even mixtures of them. In the above datatype definition

- `COEFF` stands for a coefficient,
- `VAR` to denote that a certain string such as "x" stands for a variable,
- `POWER` for the exponent of a variable,
- `PLUS` for the addition operation, and
- `MULT` for the multiplication operation.

Consider a list of real numbers (as in Question 2) representing a polynomial of one variable x. Write a function `convert` which converts a list of reals into an element of the datatype `poly`.

(9 marks)

Solution.

Here again we directly code the ML function to convert a polynomial represented as a list of coefficients into the given datatype `poly`. However, rather than give a direct conversion (since that requires knowing the power to which the variable needs to be raised in each case) we follow Arden's rule which allows us to represent the polynomial $\sum_{i=0}^{n} a_i x^i$ in the equivalent form $(\cdots(((a_n x + a_{n-1})x + a_{n-2})x + \cdots) + a_1)x + a_0$, by factoring out x in each case. This form of the polynomial does not require any counting and we can directly write the code of the conversion function through an auxiliary function `auxConvert`.

```
local fun auxConvert ([], p)    = p
        | auxConvert (h::L, p ) =
      auxConvert (L, PLUS (MULT (p, VAR(``x'')), COEFF(h)))
   in
      fun convert []     = COEFF(0.0)
        | convert (h::L) = auxConvert (L, COEFF(h))
  end
```

4. It is often possible to check whether a program works correctly for a given input by writing another checking program. Assume there is a function `sort R L` which sorts a list `L: 'a list` under an ordering relation `R: 'a * 'a -> bool`. In general if

   - `sort` has not been proven to be correct, or
   - `R` is a very complicated relation, or
   - the input list `L` is too long that it is tedious or difficult to check that the result of sorting is correct, or
   - a combination of the above reasons,

   then it is often useful to use a checking program before using the result for other purposes.

   Write a function `sortCheck` which checks whether `sort` works correctly for each input. But before you write it answer the following questions.

   (a) What input arguments does `sortCheck` take?

   (b) What is the value returned by the function `sortCheck`?

   (c) What properties should `sortCheck` check?

   (d) `sortCheck`

   (1 + 1 + 2 + 8 = 12 marks)

   Solution.

   (a) Two possible answers:
   - `R`, `L` and the output list of `sort`, or
   - `R`, `L` and `sort` itself.

   (b) Boolean

   (c) The permutation property and the ordering property.

   (d) We will assume that `sortCheck` takes the relation, the input list `L` and the ouput list `M` produced by `sort` as arguments.

```
local fun ordered (R, [])  = true
        | ordered (R, [h]) = true
        | ordered (R, h1::h2::M) = R(h1, h2) andalso ordered (R, h2::M)

      fun perm ([], []) = true
        | perm (_, []) = false
        | perm ([], _) = false
        | perm (h1::L, h2::M) =
              if h1=h2 then perm (L, M)
              else let fun belongsRemove (h, []) = (false, [])
```

```
      | belongsRemove (h, [h]) = (true, [])
      | belongsRemove (h, h2::M) =
           if h=h2 then (true, M)
           else let val (B, L) = belongsRemove (h, M)
                 in (B, h2::M)
                end
   val (C, N) = belongsRemove (h1, M)
 in C andalso perm (L, N)
end
```

Here the function `belongsRemove` removes a single copy of an element `h` from a list `L` provided it is present in the list. If it does not belong in the list it returns a value `false` and also the list. This function ensures that we do not check for an element separately and then remove it in another traversal.