Assignment 4

Out: Thursday, 05 October 2006
Due: Tuesday, 17 October 2006

Total: 150 points

Exercise 1 (20 points) Implement a procedure that takes two matrices and returns their product. Note that two matrices can be multiplied only if they are of dimensions \( p \times q \) and \( q \times r \) for some natural numbers \( p, q, r \).

type matrix = (real list) list (* a list of rows *)
val matrix_mul : matrix * matrix -> matrix

Exercise 2 (40 points) The following declaration introduces a datatype 'a mylist for representing lists.

datatype 'a mylist = Nil
| Cons of 'a * 'a mylist
| Append of 'a mylist * 'a mylist
| Reverse of 'a mylist

The list append and reverse functions can be implemented as follows:

fun append (xs: 'a mylist, ys: 'a mylist) = Append (xs, ys)
fun reverse (xs: 'a mylist) = Reverse xs

In other words, both append and reverse are \( O(1) \)-time functions, that is, they take constant time to finish. Please implement the following functions:

val is_empty : 'a mylist -> bool
val length : 'a mylist -> int
val nth : 'a mylist * int -> 'a
val to_list : 'a mylist -> 'a list

Note that \( \text{nth}(xs, i) \) returns the \( i^{th} \) element in the list represented by \( xs \). The meaning of all other functions should be obvious.

Exercise 3 (90 points) Given a sequence \( s \) consisting of \( n \) elements \( a_1, \ldots, a_n \), we represent \( s \) as follows.

- If \( n = 0 \), that is, \( s \) is empty, we use Empty to represent \( s \);
- If \( n = 2k \) for some \( k > 0 \), we use Even \((xs)\) to represent \( s \), where \( xs \) represents the following sequence of pairs:
  \[(a_1, a_2), \ldots, (a_{n-1}, a_n)\]
• If \( n = 2k + 1 \) for some \( k \geq 0 \), we use \( \text{Odd} \ (x, xs) \) to represent \( s \), where \( x \) represents \( a_1 \) and \( xs \) represents the following sequence of pairs.

\[(a_2, a_3), \ldots, (a_{n-1}, a_n)\]

We use the name random-access list for such a representation of \( l \). The following datatype \( 'a \ ralist \) is declared for this kind of representation of list:

```ml
datatype 'a pair = S of 'a | P of 'a pair * 'a pair
datatype 'a ralist = Empty
| Even of 'a ralist
| Odd of 'a pair * 'a ralist
```

Please implement the following operations on random-access lists. All the implementations should be of \( O(\log n) \)-time complexity in order to receive full credit.

1. (20 points) Given an element \( x \) and a random-access list \( xs \), \( \text{racons} \ (x, xs) \) generates a random-access list whose head and tail are \( x \) and \( xs \), respectively.

2. (20 points) Given a nonempty random-access list \( xs \), \( \text{rauncons} \ (xs) \) return a pair consisting of the head and tail of \( xs \).

3. (20 points) Given a number \( n \) and a random-access list \( xs \), \( \text{ralookup} \ (n, xs) \) returns the \( n \)th element (counting starts with 0) in \( xs \) or issues an error if the length of the random-access list is less than or equal to \( n \).

4. (30 points) Given a number \( n \) and a random-access list \( xs \), \( \text{raupdate} \ (xs, n, x) \) updates the \( n \)th element (counting starts with 0) in \( xs \) with \( x \) or issues an error if the length of the random-access list is less than or equal to \( n \). This one is a bit challenging, and the solution may involve the feature of higher-order function.

The types of these functions are given below:

```ml
val racons : 'a pair * 'a ralist -> 'a ralist
val rauncons : 'a ralist -> 'a pair * 'a ralist
val ralookup : 'a ralist * int -> 'a pair
val raupdate : 'a ralist * int * 'a pair -> 'a ralist
```