1 Introduction

You will write some simple programs in the Standard ML language and practice using pattern matching and type inference.

2 Description

- Recall the length and append functions in Scheme and Standard ML:
  - Scheme
    ```scheme
    (define length (xs)
        (if (null? xs)
            0
            (+ 1 (length (cdr xs)))))
    (define append (xs ys)
        (if (null? xs)
            ys
            (cons (car xs) (append (cdr xs) ys))))
    ```
  - Standard ML
    ```ml
    fun length xs =
        case xs of
        [] => 0
        | z::zs => 1 + length zs
    fun append (xs , ys) =
        case xs of
        [] => ys
        | z::zs => z ::( append (zs , ys ))
    or
    fun length [] = 0
        | length (z::zs) = 1 + length zs
    fun append ([], ys) = ys
        | append (z::zs , ys) = z ::( append (zs , ys ))
    ```

Translate the following recursive functions on lists in Scheme to Standard ML.

- sum
  ```scheme
  (define sum (xs)
      (if (null? xs)
          0
          (+ (car xs) (sum (cdr xs)))))
  ```

- revapp and reverse
  ```scheme
  (define revapp (xs ys)
      (if (null? xs)
          ys
          (revapp (cdr xs) (cons (car xs) ys))))
  (define reverse (xs) (revapp xs '()))
  ```
(define all? (p? xs)
  (if (null? xs)
      #t
      (if (p? (car xs))
          (all? p? (cdr xs))
          #f)))

Notes: Name the SML function all (all? is not a valid SML identifier). SML's boolean constants are true
and false and SML's short-circuiting conjunction and disjunction operators are andalso (like C's/Java's
&& operator) and orelse (like C's/Java's || operator).

• Consider the pattern (w, x::y::zs) and the following expressions:
  – ("CSCI", 344), [1, 2, 3])
  – ("CSCI", 344), [0])
  – ([1, 2, 3], ("CSCI", 344))
  – (true, ["CSCI", "344"])
  – (2.718281828, [true, false])

For each expression, determine whether or not the pattern matches the value denoted. If the value does match
the pattern, then give the values that are bound to the four variables w, x, y, zs. If the value does not match
the pattern, then explain why not.

• Write a function evenLength of type 'a list -> bool. The function evenLength should return true if the
input list has an even number of elements and should return false otherwise. Note: evenLength must be
implemented without using integer arithmetic or integer comparisons; this precludes using length to implement
evenLength.

• Consider the following function:
  fun doit f x = f (f x) + 1

Determine the principal (“most general”) type scheme for the function doit.
Consider the following datatype for and operations on binary trees:

```mln
datatype 'a btree = Leaf | Node of 'a btree * 'a * 'a btree
```

```mln
fun btreeLookup cmp (x, btree ) = 
case btree of 
  Leaf => NONE 
| Node (lt, y, rt) =>
    (case cmp (x, y) of
       LESS => btreeLookup cmp (x, lt)
       | EQUAL => SOME y
       | GREATER => btreeLookup cmp (x, rt))
```

```mln
fun btreeInsert cmp (x, btree ) = 
case btree of
  Leaf => Node (Leaf , x, Leaf )
| Node (lt, y, rt) =>
    (case cmp (x, y) of
       LESS => Node ( btreeInsert cmp (x, lt), y, rt)
       | EQUAL => Node (lt , x, rt)
       | GREATER => Node (lt , y, btreeInsert cmp (x, rt))
```

Determine the principal (“most general”) type schemes for the functions `btreeLookup` and `btreeInsert`.

Note: The `NONE` and `SOME` constructors belong to the `'a option` type constructor, which is pre-defined in the Standard ML basis; its definition is equivalent to the following:

```mln
datatype 'a option = NONE | SOME of 'a
```

Note: The `LESS`, `EQUAL`, and `GREATER` constructors belong to the `order` type constructor, which is pre-defined in the Standard ML basis; its definition is equivalent to the following:

```mln
datatype order = LESS | EQUAL | GREATER
```

(From Programming 04: Standard ML Programming) Write functions `unzip` of type `('a * 'b) list -> ('a list * 'b list)` and `zip` of type `('a list * 'b list) -> ('a * 'b) list`. The function `unzip` takes a list of pairs and produces a pair of lists, while the function `zip` takes a pair of lists and produces a list of pairs; in each case, the order of elements is preserved. When `zip` is applied to lists of unequal length, the excess elements from the tail of the longer one are ignored. For example, `unzip [(1,2),(3,4),(5,6)]` should return `[(1,3,5),[2,4,6]]` and `zip ([(1,3,5),[2,4]])` should return `[(1,2),(3,4)].` The `unzip` and `zip` functions satisfy the following algebraic laws:

- `unzip-zip law: zip (unzip xys) = xys`
- `zip-unzip law: unzip (zip (xs, ys)) = (xs, ys) if length xs = length ys`

### 3 Requirements and Submission

You may use the reference interpreter (see Appendix A), but there may only be one active laptop in each group.

At the end of class, submit the group’s solutions either as hard-copy or by e-mail to `mtf@cs.rit.edu` be sure to include the names of all group members in the submission.

### A Interpreters

A reference Standard ML interpreter (MoscowML) is available on the CS Department Linux systems (e.g., `glados.cs.rit.edu` and `queeg.cs.rit.edu` and ICLs 1 and 2) at:

```
/usr/local/pub/mtf/plc/bin/mosml
```