

**Possible solutions for  
Written examination  
9 June 2004**

Version 1.0 of 2004-06-04

The suggested solutions below are not the only possible ones.

**Question 1 (25 %): Standard ML**

**Question 1.1**

```
val add = fn : int -> int list -> int list
val mul = fn : int -> int list -> int list
val fromTo = fn : int -> int -> int list
val 'a tabulate = fn : int * (int -> 'a) -> 'a list
val res1 = [2, 3, 4, 5] : int list
val res2 = [6, 9, 12, 15] : int list
val res3 = [] : int list
val res4 = [] : int list
val res5 = true : bool
val res6 = [25, 16, 9, 4, 1, 0] : int list
```

The application `add k xs` returns the list resulting from adding `k` to every member of `xs`.

The application `mul k xs` returns the list resulting from multiplying every member of `xs` by `k`.

The application `fromTo m n` returns the list `[m, m+1, ..., n]` when `m <= n`, and the empty list if `m > n`.

The application `tabulate(m, f)` returns the list `[f m, f(m-1), ..., f 1, f 0]` when `m >= 0`, and the empty list if `m < 0`.

**Question 1.2**

```
fun add1 k xs = map (fn x => k+x) xs
fun mul1 k xs = map (fn x => k*x) xs
fun fromTo1 m n = tabulate(n-m, fn x => n-x)
```

**Question 1.3**

```
fun sequence (a, i, k) = add a (mul i (fromTo 0 (k-1)))
```

**Question 1.4**

```
fun isRegSeq [] = true
  | isRegSeq [_] = true
  | isRegSeq (x0 :: x1 :: xr) =
    let val i = x1-x0
        fun check last [] = true
          | check last (y::yr) = (y-last) = i andalso check y yr
        in check x1 xr end
```

**Question 1.5**

```
fun getRegSeq [] = (1,1,0)
  | getRegSeq [x] = (x,1,1)
  | getRegSeq (x0 :: x1 :: xr) =
    let val i = x1-x0
        fun get k last [] = (x0, i, k)
          | get k last (y::yr) =
            if (y-last) = i then get (k+1) y yr else raise Fail "not regular"
        in get 2 x1 xr end
```

**Question 2 (25 %): Grammar and abstract syntax**

**Question 2.1**

```
Expr ::= m
      | [ ]
      | [ m : n ]
      | Expr + Expr
      | Expr * Expr
      | Expr ++ Expr
      | max Expr
      | min Expr
      | rev Expr
      | ( Expr )
```

**Question 2.2**

```
%left PLUSPLUS          /* lowest precedence */
%left PLUS              /* highest precedence */
%left TIMES
```

**Question 2.3 and 2.4**

```
Main:
    Expr EOF           { $1 }
;
Expr:
    AtExpr            { $1 }
    | Expr PLUS Expr  { Add($1, $3) }
    | Expr TIMES Expr { Mul($1, $3) }
    | Expr PLUSPLUS Expr { Append($1, $3) }
    | MAX AtExpr      { Max $2 }
    | MIN AtExpr      { Min $2 }
    | REV AtExpr      { Rev $2 }
;
AtExpr:
    Int               { CstI $1 }
    | LBRACKET Int COLON Int RBRACKET { FromTo($2, $4) }
    | LBRACKET RBRACKET { Empty }
    | LPAR Expr RPAR  { $2 }
;
Int:
    INT              { $1 }
    | MINUS INT      { ~ $2 }
;
```

## Question 3

## Question 3.1

```

fun eval (e : expr) : value =
  case e of
    CstI i          => Int i
  | FromTo(m, n) => Seq (fromto m n)
  | Empty         => Seq []
  | Add(e1, e2) =>
      (case (eval e1, eval e2) of
        (Int i1, Int i2) => Int (i1+i2)
        | (Int i1, Seq s2) => Seq (map (fn x => i1+x) s2)
        | (Seq s1, Int i2) => Seq (map (fn x => x+i2) s1)
        | (Seq _, Seq _) => raise Eval "cannot add two sequences")
      )
  | Mul(e1, e2) =>
      (case (eval e1, eval e2) of
        (Int i1, Int i2) => Int (i1*i2)
        | (Int i1, Seq s2) => Seq (map (fn x => i1*x) s2)
        | (Seq s1, Int i2) => Seq (map (fn x => x*i2) s1)
        | (Seq _, Seq _) => raise Eval "cannot multiply two sequences")
      )
  | Append(e1, e2) =>
      (case (eval e1, eval e2) of
        (Seq s1, Seq s2) => Seq (s1 @ s2)
        | (_, _) => raise Eval "append works only on sequences")
      )
  | Min e1 =>
      (case eval e1 of
        Seq s1 => Int (foldl Int.min 1073741823 s1)
        | _ => raise Eval "min works on sequences only")
      )
  | Max e1 =>
      (case eval e1 of
        Seq s1 => Int (foldl Int.max ~1073741824 s1)
        | _ => raise Eval "max works on sequences only")
      )
  | Rev e1 =>
      (case eval e1 of
        Seq s1 => Seq (List.rev s1)
        | _ => raise Eval "rev works on sequences only")
      )

```

## Question 3.2

```
fun check (e : expr) : typ =
  case e of
    CstI _    => TypI
  | FromTo _ => TypS
  | Empty    => TypS
  | Add(e1, e2) =>
      (case (check e1, check e2) of
        (TypI, TypI) => TypI
        | (TypI, TypS) => TypS
        | (TypS, TypI) => TypS
        | (TypS, TypS) => raise Type "cannot add two sequences")
  | Mul(e1, e2) =>
      (case (check e1, check e2) of
        (TypI, TypI) => TypI
        | (TypI, TypS) => TypS
        | (TypS, TypI) => TypS
        | (TypS, TypS) => raise Type "cannot multiply two sequences")
  | Append(e1, e2) =>
      (case (check e1, check e2) of
        (TypS, TypS) => TypS
        | (_, _ ) => raise Type "append works only on sequences")
  | Min e1 =>
      (case check e1 of
        TypS => TypI
        | _    => raise Type "min works on sequences only")
  | Max e1 =>
      (case check e1 of
        TypS => TypI
        | _    => raise Type "max works on sequences only")
  | Rev e1 =>
      (case check e1 of
        TypS => TypS
        | _    => raise Type "rev works on sequences only")
```

Question 4

Question 4.1

Here are some additional simplifications, but even this list is not exhaustive. The question asked only for four additional simplifications.

---



---

|                               |                      |                               |   |
|-------------------------------|----------------------|-------------------------------|---|
| <code>[] ++ ys</code>         | can be simplified to | <code>ys</code>               |   |
| <code>xs ++ []</code>         | can be simplified to | <code>xs</code>               |   |
| <code>[k,m] ++ [m+1,n]</code> | can be simplified to | <code>[k,n]</code>            | when $k \leq m \leq n$                      |
| <code>k1 * (e * k2)</code>    | can be simplified to | <code>(k1 * k2) * e</code>    |   |
| <code>k1 + k2</code>          | can be simplified to | <code>k3</code>               | where $k3 = k1 + k2$ and $k1, k2$ constants |
| <code>[m:n] + k</code>        | can be simplified to | <code>[m+k:n+k]</code>        |   |
| <code>rev(-1 * [m:n])</code>  | can be simplified to | <code>[-n:-m]</code>          |   |
| <code>max(rev e)</code>       | can be simplified to | <code>max e</code>            |   |
| <code>min(rev e)</code>       | can be simplified to | <code>min e</code>            |   |
| <code>rev(e1 ++ e2)</code>    | can be simplified to | <code>rev e2 ++ rev e1</code> |   |
| <code>k * (e1 ++ e2)</code>   | can be simplified to | <code>k * e1 ++ k * e2</code> |   |
| <code>k + (e1 ++ e2)</code>   | can be simplified to | <code>k + e1 ++ k + e2</code> |   |
| <code>k * rev e</code>        | can be simplified to | <code>rev (k * e)</code>      |   |
| <code>k + rev e</code>        | can be simplified to | <code>rev (k + e)</code>      |   |

---



---

Question 4.2

Function `simp` implements the basic reductions shown in the exam questions, and their symmetric variants, and a few additional simplifications. Function `simplify` repeatedly applies function `simp` as long as more simplifications are possible.

```

fun simp (e : expr) : expr =
  case e of
    CstI _      => e
  | FromTo(m, n) => if m>n then Empty else e
  | Empty       => e
  | Add(e1, e2) =>
    (case (simp e1, simp e2) of
      (CstI k1, CstI k2)      => CstI (k1+k2)
    | (CstI k1, FromTo(m, n)) => FromTo(k1+m, k1+n)
    | (CstI 0, e2')          => e2'
    | (e1', CstI 0)          => e1'
    | (FromTo(m, n), CstI k2) => FromTo(k2+m, k2+n)
    | (e1', e2')             => Add(e1', e2'))
  | Mul(e1, e2) =>
    (case (simp e1, simp e2) of
      (CstI k1, CstI k2)      => CstI (k1*k2)
    | (CstI 1, e2')          => e2'
    | (e1', CstI 1)          => e1'
    | (e1', e2')             => Mul(e1', e2'))
  | Append(e1, e2) =>
    (case (simp e1, simp e2) of
      (e1', e2')              => Append(e1', e2'))
  | Min e1 =>
    (case simp e1 of
      Empty => CstI 1073741823
    | FromTo(m, n) => CstI m
    | Mul(CstI k, e12') =>
      if k > 0 then
        Mul(CstI k, simp(Min e12'))
      else if k=0 then
        CstI 0
      else
        Mul(CstI k, simp(Max e12'))
    | e1' => Min e1')
  | Max e1 =>
    (case simp e1 of
      Empty => CstI ~1073741824
    | FromTo(m, n) => CstI n
    | Mul(CstI k, e12') =>
      if k > 0 then
        Mul(CstI k, simp(Max e12'))
      else if k=0 then
        CstI 0
      else
        Mul(CstI k, simp(Min e12'))
    | e1' => Max e1')
  | Rev e1 =>
    (case simp e1 of
      Empty => Empty
    | Rev e11 => e11
    | e11 => Rev e11)

fun simplify e =
  let val e' = simp e
  in if e = e' then e else simplify e' end

```

## Question 4.3

1. The first 'simplification' is wrong because `2 * [1:3]` evaluates to `[2, 4, 6]` whereas `[2:6]` evaluates to `[2, 3, 4, 5, 6]`.
2. The second 'simplification' is wrong because `min [3:2]` evaluates to `1073741823` whereas `3` evaluates to `3`.
3. The third 'simplification' is wrong because `max [3:2]` evaluates to `-1073741824` whereas `2` evaluates to `2`.
4. The fourth 'simplification' is wrong because `rev [3:2]` evaluates to `[]` whereas `[2:3]` evaluates to `[2, 3]`.