Homework 5 — ML module language — assigned Thursday 20 March, due Friday 4 April

Total number of points available on this homework is 200. Full credit is equivalent to 100 points.
Some problems in this homework set are extensions of problems from the first four sets. Therefore you can reuse your code. Try, however, to make the code more elegant, and more in the ML style of programming; use the full wealth of purely functional ML programming constructs.

Reading assignment

Read Chapters 5, 6, and 7 of *ML for the Working Programmer*. In Chapter 7, ignore the obsolete `absttype` construct.

5.1 Specification and implementation: sets of integers (35pts)

The following is a specification for the abstract data type of immutable sets of integers, from an early version of the SML Standard Library.

(* intset-sig.sml
 *
 * COPYRIGHT (c) 1993 by AT&T Bell Laboratories. See COPYRIGHT file for details.
 *
 * Signature for sets of integers.
 *)

signature INTSET =
  sig
    type intset
    exception NotFound
    
    val empty : intset
    (* Create a new set *)

    val singleton : int -> intset
    (* Create a singleton set *)

    val add : intset * int -> intset
    (* Insert an int. *)

    val addList : intset * int list -> intset
    (* Insert a list of ints. *)

    val isEmpty : intset -> bool
(* Return true if and only if the set is empty *)
val equal : (intset * intset) -> bool
(* Return true if and only if the two sets are equal *)
val isSubset : (intset * intset) -> bool
(* Return true if and only if the first set is a subset of the second *)
val member : intset * int -> bool
(* Return true if and only if the int is an element in the set *)
val delete : intset * int -> intset
(* Remove an int. *)
    * Raise NotFound if not found
val numItems : intset -> int
(* Return the number of ints in the table *)
val union : intset * intset -> intset
    (* Union *)
val intersection : intset * intset -> intset
    (* Intersection *)
val difference : intset * intset -> intset
    (* Difference *)
val listItems : intset -> int list
(* Return a list of the ints in the set *)
val app : (int -> 'b) -> intset -> unit
(* Apply a function to the entries of the set *)
    * in decreasing order
val revapp : (int -> 'b) -> intset -> unit
(* Apply a function to the entries of the set *)
    * in increasing order
val fold : (int * 'b -> 'b) -> intset -> 'b -> 'b
(* Apply a folding function to the entries of the set *)
    * in decreasing order
val revfold : (int * 'b -> 'b) -> intset -> 'b -> 'b
(* Apply a folding function to the entries of the set *)
    * in increasing order
val exists : (int -> bool) -> intset -> int option
(* Return an item satisfying the predicate, if any *)

end (* INTSET *)

Write a structure IntSet that matches the signature INTSET and implements sets of integers. Do not worry about efficiency.

5.2 Specification and implementation: directed graphs (35pts)

The following is a partial specification for an abstract data type of immutable directed graphs.

signature DIGRAPH =
  sig
    type ('a, 'b, 'c) digraph
    (* An immutable type of directed graphs,
       with node labels of type 'b,
       edge labels of type 'c,
       and a general graph information label of type 'a.
       Both nodes and edges are indexed. Indices can be arbitrary integers
       (i.e., they need not be
       contiguous; they need not be 1..n) *)
  end

exception Digraph of string

val create: 'a -> ('a, 'b, 'c) digraph
val set_graphinfo: ('a, 'b, 'c) digraph * 'a -> ('a, 'b, 'c) digraph
val get_graphinfo: ('a, 'b, 'c) digraph -> 'a
val num_nodes: ('a, 'b, 'c) digraph -> int
val max_node: ('a, 'b, 'c) digraph -> int
val get_nodes: ('a, 'b, 'c) digraph -> IntSet.intset
val add_node: ('a, 'b, 'c) digraph * 'b -> ('a, 'b, 'c) digraph * int
    (* Returns the new graph, with one more node with the given label, and also
    returns the index assigned to that node *)
val rem_node: ('a, 'b, 'c) digraph * int -> ('a, 'b, 'c) digraph
    (* Returns the new graph, with the node of given index removed.
    Raises exception Digraph if there is no such node in the input graph. *)
val set_nodeinfo: ('a, 'b, 'c) digraph * int * 'b -> ('a, 'b, 'c) digraph
val get_incoming_edges: ('a, 'b, 'c) digraph * int -> IntSet.intset
val get_outgoing_edges: ('a, 'b, 'c) digraph * int -> IntSet.intset
val get_nodeinfo: ('a, 'b, 'c) digraph * int -> 'b
val num_edges: ('a, 'b, 'c) digraph -> int
val max_edge: ('a, 'b, 'c) digraph -> int
val get_edges: ('a, 'b, 'c) digraph -> IntSet.intset
val exists_edge: ('a, 'b, 'c) digraph * (int * int) -> bool
    (* Test whether there is an edge between two nodes. *)
val add_edge: ('a, 'b, 'c) digraph * (int * int) * 'c
    -> ('a, 'b, 'c) digraph * int
val rem_edge: ('a, 'b, 'c) digraph * int -> ('a, 'b, 'c) digraph
val set_edgeinfo: ('a, 'b, 'c) digraph * int * 'c -> ('a, 'b, 'c) digraph
val get_from_node: ('a, 'b, 'c) digraph * int -> int
val get_to_node: ('a, 'b, 'c) digraph * int -> int
val get_edgeinfo: ('a, 'b, 'c) digraph * int -> 'c

val as_string: ('a, 'b, 'c) digraph
    * ('a -> string) * ('b -> string) * ('c -> string)
    -> string
end

Complete the specification; i.e., add appropriate comments giving the requirements and effects of the functions. Write a structure `Digraph` that matches the signature `DIGRAPH` and implements operations on directed graphs.

5.3 Polynomials (30pts)

In the textbook *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*, Barbara Liskov and John Guttag give the following specification for an abstract data type of polynomials:

```java
public class Poly
{
    // OVERVIEW: Polys are immutable polynomials with integer coefficients.
    // A typical Poly is c_0 + c_1 x + ...

    // constructors
    public Poly ()
    {  // EFFECTS: Initializes this to be the zero polynomial

    public Poly (int c, int n) throws NegativeExponentException
    {  // EFFECTS: If n < 0 throws NegativeExponentException else
       // initializes this to be the Poly c x^n

    //methods
    public int degree ()
    {  // EFFECTS: Returns the degree of this, i.e., the largest exponent
       // with a non-zero coefficient. Returns 0 if this is the zero Poly.

    public int coeff (int d)
    {  // EFFECTS: Returns the coefficient of the term of this whose exponent is d.

    public Poly add (Poly q) throws NullPointerException
    {  // EFFECTS: If q is null throws NullPointerException else
       // returns the Poly this + q.

    public Poly mul (Poly q) throws NullPointerException
    {  // EFFECTS: If q is null throws NullPointerException else
       // returns the Poly this * q.
```
public Poly sub (Poly q) throws NullPointerException
   // EFFECTS: If q is null throws NullPointerException else
   // returns the Poly this - q.

public Poly minus ()
   // EFFECTS: Returns the Poly - this.
}

Write an SML signature POLY that captures the same notion. Then write an SML structure Poly that matches the
signature POLY and implements the abstract data type of polynomials.

5.4 Expressions using arbitrary-precision floating-point numbers (60pts)

5.4.1 Floating-point numbers (30pts)

This is a continuation of exercise 4.6.

We extend the requirements as follows:

- You must write a pretty-printer to convert a number to a string.
- You must handle floating-point numbers. The following type is used for floating-point numbers:

  datatype sign = Pos | Neg | Zero
datatype float =
     Float of {radix: int, sign: sign, mantissa: int list, exponent: int}

  For instance,

  Float {radix= 10, sign= Pos, mantissa= [3,1,4], exponent= ~2}

  represents \(314 \times 10^{-2} = 3.14\).

- The numerals in the input expression string can be in any radix. The syntax is indicated by these examples:

  The number twenty can be represented as any of the following strings: 20, 10x20, 16x14, 2x10100, 3x202,
  20x10, 30xK, 30x<20>. (The notation 30xK generalizes the alphanumeric notation that is standard for
  hexadecimal numbers. Clearly, it is only usable if the radix is at most 36.)

  The number one-half can be represented as any of the following strings: 0.5, 5e-1, 2x1e-1, 2x01. Note that
  .5 is not a valid representation! The radix and the exponent are always written in decimal.

5.4.2 Encapsulation (30pts)

Package the code using the SML module system. You should provide an expression-parsing and an expression-
evaluation abstraction that are independent of the particular numbers used in the expressions; then you should
provide three kinds of numbers: ordinary ML ints, longints (from exercise 4.4) and long floats (this exercise).
You should abstract over the different kinds of expression evaluators as well (e.g., simple arithmetic evaluator vs.
evaluator that supports let-binding).
5.5 ★★ Tic-tac-toe (40pts)

This is a continuation of exercise 4.7.

The task is to redo exercise 4.7, subject to certain restrictions. You should consider each of these possible restrictions:

1. The first move of player X is 5, and the first move of player O is restricted to be either 1 or 4.
2. The first move of player X is 5, and the first move of player O is restricted to be 1, 2, 3, or 4.
3. The first move of player X is 5, and the first move of player O is restricted to be 1, 3, 7, or 9.
4. The first move of player X is 5, and the first move of player O is restricted to be 2, 4, 6, or 8.

How to turn in

Turn in your code by running

```
~roshan/handin your-file
```

on a regular UNM CS machine.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.