Problem Set 2

1. In this first problem you are asked to define both insert sort and bubble sort functions on lists of natural numbers defined with a list concatenation operator `_;_` which is associative and has `nil` as its unit element. Since you already became familiar with how to define functions by equations in Problem Set 1, you can feel free to use other Maude features such as the built-in (but also equationally defined) `if_then_else_fi` operator (included together with the built-in `BOOL` module, which is imported by default into any other module), and also the `[owise]` feature. Another convenience in this example is the importation of the built-in module `NAT`, so that Boolean predicates such as `_>_` and `_>=_` on natural numbers are already available to you. A last new possibility that may be useful to you is the use of *conditional* equations, with syntax:

   ```
   ceq u = v if cond .
   ```

   Conditions can generally be a conjunction of equalities. Here, when declaring a conditional equation, it may be quite enough for you to give a Boolean condition of the form `bexp`, where `bexp` is a Boolean expression, which is just syntactic sugar for the equation: `bexp = true`. A few test cases are included for your convenience.

   ```
   fmod INSERT&BUBBLE-SORT is protecting NAT .
     sort List .
     subsort Nat < List .
     op nil : -> List [ctor] .
     op _;_ : List List -> List [ctor assoc id: nil] .
     ops isort bsort : List -> List .  *** insert, resp. bubble, sort

     vars N M : Nat .   vars L Q : List .

     *** include here your equations for isort and bsort,
     *** as well as definitions for any auxiliary operators
     *** such as, e.g., an insert function, and the corresponding
     *** equations defining such auxiliary operators.

   endfm

   red isort(6 ; 5 ; 4 ; 3 ; 1 ; 0) .  *** should be: 0 ; 1 ; 3 ; 4 ; 5 ; 6
   red isort(6 ; 3 ; 5 ; 3 ; 1 ; 1) .  *** should be: 1 ; 1 ; 3 ; 3 ; 5 ; 6
   red isort(6 ; 3 ; 5 ; 3 ; 1 ; 3) .  *** should be: 1 ; 3 ; 3 ; 3 ; 5 ; 6

   red bsort(6 ; 5 ; 4 ; 3 ; 1 ; 0) .  *** same results as for isort
   red bsort(6 ; 3 ; 5 ; 3 ; 1 ; 1) .
   red bsort(6 ; 3 ; 5 ; 3 ; 1 ; 3) .
   ```

2. Multisets of natural numbers can be defined using a binary associative and commutative multiset union constructor _, _. Although _, _ has `mt` as its identity element, the operator _, _ will only be declared associative and commutative, so that the identity property of the empty multiset `mt` *has to be defined by an explicit equation.* The reason for not using the `id: mt` attribute here is to save you from potential non-termination problems you might run into, since you are still becoming familiar with the use of axioms and combinations of the `id:` axioms with A or AC can be tricky. Although you can use Maude features like the `[owise]` or Maude's `==` built-in equality predicate, we would recommend that you do not use them in this example, since this can give you a more complete experience on how to define equations modulo axioms without the help of extra props. You are asked to write equations defining the following additional properties and functions:

(a) an equation stating that `mt` is an identity element for `_,_`

(b) an equality predicate on numbers

(c) multiset difference between two multisets

(d) the containment predicate `_` $\subseteq$ `_` on multisets

(e) the membership relation `_` $\in$ `_` of a number in a multiset

(f) an equality predicate between multisets

(g) intersection of multisets

(h) a function removing all occurrences of a number in a multiset

(i) cardinality of a multisets (counting repetitions)

(j) a function computing how many *different* naturals appear in a multiset.

Given a number n and a multiset U, define the *multiplicity* of n in U, denoted $mult(n, U)$, as the number of occurrences of n in U. For example, $mult(3, (1, 2, 2, 3, 3, 3, 3, 7)) = 4$.

Since notions of multiset difference, containment, membership, intersection, and removing a number must take account of multiplicities, we can specify precisely what these functions should do in terms of multiplicities:

- the multiplicity of any number n in the multiset difference U minus V should be $mult(n, U) - mult(n, V)$,
- we should have $U \subseteq V$ true iff for each n we have $mult(n, U) \leq mult(n, V)$,
- $n \in U$ should be true iff $mult(n, U) \neq 0$,
- the multiplicity of any number n in the multiset intersection $U \cap V$ should be $min(mult(n, U), mult(n, V))$,
- the multiplicity of n in $rem(m, U)$ should be 0 if $n = m$ and $mult(n, U)$ otherwise.

Multiset cardinality counting repetitions is the obvious function, e.g., $|3, 3, 4, 4, 4, 5, 5, 5, 5, 5| = 10$. Instead, the number of distinct elements is $[3, 3, 4, 4, 4, 5, 5, 5, 5, 5] = 3$. Finally, a multiset equality predicate has the obvious meaning: two multisets are equal iff they are equal as terms modulo associativity and commutativity.

Now that the meaning of all these functions has been clarified, you are asked to give equations making `mt` the identity element for multiset union and defining all the functions listed in the module below by writing their appropriate equational definitions modulo the associativity and commutativity axioms of multiset union. Example tests are included for your convenience.

Hints:

- The built-in module `NAT` is included for your convenience because: (i) it supports decimal notation and also Peano notation: 3 can be written both as `3` and as `s(s(s(0)))`, which is very convenient: you can for example define the equality predicate between naturals just using the Peano notation; (ii) it imports the `BOOL` module, so you have at your disposal all the Boolean operations, which can be useful when defining some of the predicates; and (iii) `BOOL` itself imports the if-then-else-fi operator, which again can be helpful when defining some functions.
- The order in which the functions are introduced gives you a hint that some functions earlier in the list may be useful as auxiliary functions for defining other functions later down the list.
- Programming modulo axioms of associativity and commutativity is very powerful and allows writing very short programs. For example, the identity property of `mt` and the nine functions in this example can be defined with just 30 equations. However, with this power comes also the risk of losing sufficient completeness: you may forget some cases in your equations if you are not careful.

```
fmod MULTISET-ALGEBRA is
  protecting NAT .
  sort Mult .
  subsort Nat < Mult .
  op mt : -> Mult [ctor] .                              *** empty multiset
  op _,_ : Mult Mult -> Mult [ctor assoc comm] .        *** multiset union
  op _.=._ : Nat Nat -> Bool [comm] .     *** equality predicate on naturals
```

```
  op _\_  : Mult Mult -> Mult .           *** multiset difference
  op _C=_ : Mult Mult -> Bool .           *** multiset containment
  op _in_ : Nat Mult -> Bool .            *** multiset membership
  op _.=._  : Mult Mult -> Bool [comm] .  *** equality predicate on multisets
  op _/\_ : Mult Mult -> Mult .           *** multiset intersection
  op rem : Nat Mult -> Mult .             *** removes N everywhere in U
  op |_| : Mult -> Nat .                  *** cardinality with repetitions
  op [_] : Mult -> Nat .                  *** number of distinct elements

  vars N M : Nat .  vars U V W : Mult .

  *** write here your equations for the identity of mt and all the functions above

endfm

red 5 .=. 12 .                  *** should be false
red 15 .=. 15 .                 *** should be true

red (3,3,4,4,4,2,2,9) \ (3,3,3,4,2,7) .  *** should be 2,4,4,9

red (3,3,4,4,4,2,2,9) C= (3,3,3,4,2,7) .  *** should be false

red (3,3,4,4,2,2,9) C= (3,3,3,4,4,2,2,2,7,9) .  *** should be true

red 3 in (3,3,4,4,7) .                     *** should be true

red 9 in (3,3,4,4,7) .                     *** should be false

red (3,3,4,4,4,2,2,7) .=. (3,3,3,4,2,7) .   *** should be false

red (3,3,3,4,2,2,7) .=. (3,3,3,4,2,2,7) .   *** should be true

red (3,3,3,4,4,4,2,2,7,9) /\ (3,3,3,3,4,4,2,7,7) . *** should be 2,3,3,3,4,4,7

red rem(2,(3,3,2,2,2,4,4,4)) .  *** should be 3,3,4,4,4

red | 3,3,4,4,4,2,2,9 | .       *** should be 8

red [ 3,3,4,4,4,2,2,9 ] .       *** should be 4
```