CS 476 Homework #4 Due 10:45am on 9/19

Note: Answers to the exercises listed below, as well as the Maude code for Problem 2, should be emailed by the above deadline to `clarage2@illinois.edu`.

1. In the slides for Lecture 5, given an equational theory (Σ, E) the *joinability relation* $t \downarrow_{\vec{E}} t'$ is defined by the equivalence:

$$t \downarrow_{\vec{E}} t' \;\; \Leftrightarrow \;\; (\exists w)\, (t \rightarrow^*_{\vec{E}} w \wedge t' \rightarrow^*_{\vec{E}} w).$$

   That is, $t \downarrow_{\vec{E}} t'$ holds iff t and t' *can both be rewritten to a common term* w (the modulo case for $t \downarrow_{\vec{E}/B} t'$ is defined in Lecture 5 in the same way, but we will stick to the case without axioms in this exercise).

   Prove the *Church-Rosser Theorem* (also stated in Lecture 5) in the following form:

   If (Σ, E) is a kind-complete and order-sorted equational theory such that the rules $\vec{E}$ are confluent and sort-decreasing, then for any two Σ-terms t, t' whose sorts are in the same connected component we have the equivalence:

$$t =_E t' \;\; \Leftrightarrow \;\; t \downarrow_{\vec{E}} t'.$$

   Hint: Use induction on the length of the proof of $t =_E t'$.

   Note that, under the assumption of confluence and sort-decreasingness, the Church Rosser Theorem *reduces* the very difficult problem of *proving* $t =_E t'$ to the much easier problem of checking that t and t' can both be rewritten to a common term w. Furthermore, if $\vec{E}$ is not only confluent and sort decreasing but also *terminating*, the problem of checking whether $t =_E t'$ holds becomes even easier: we just reduce t and t' to their respective *normal forms* by $\vec{E}$, say, $t!_{\vec{E}}$ and $t'!_{\vec{E}}$, and then compare $t!_{\vec{E}}$ and $t'!_{\vec{E}}$ for syntactic equality. If they are syntactically equal, then $t =_E t'$. Otherwise, $t =_E t'$ is *not provable*. Of course, you can just do this in Maude by typing:

   `red` t `==` t' .

   So, under the assumptions of confluence, sort decreasingness and termination for $\vec{E}$, Maude becomes not just a *theorem prover* for E-equality, but actually a *decision procedure* for E-equality. All this should motivate you to prove the above Church-Rosser Theorem: it is not some theoretical curiosity: it is a *fundamental theorem* reducing equational reasoning to rewriting and easily mechanizable in Maude.

2. This problem is a good example of the motto:

   Declarative Programming = Mathematical Modeling

   Specifically, of how you can define an executable mathematical model of multisets of natural numbers and an algebra of useful functions on multisets which at the same time is an *implementation* of that data structure and that algebra of functions.

   Since your *mathematical model* should be specified by an equational theory without extra-logic features,

   > *No use should be made of the* `owise` *attribute in equations. Likewise, no use should be made of the built-in equality predicate* `==` *in any equations.*

The `owise` attribute is very convenient for programming purposes, since fewer equations are needed, but it is not essentially needed. So is the built-in equality predicate `==`. But by using either of them you are not giving a full mathematical definition. Here you are asked to give a full *mathematical definition* of all the functions involved, which should at the same time be a *correct program*.

Multisets of natural numbers are defined as expected 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 explicit equation(s)*. You are asked to write equations defining the following additional properties and functions:

(a) the property that `mt` is an identity element for _,_

(b) an equality predicate on numbers

(c) multiset difference between two multisets

(d) the containment predicate _ ⊆ _ on multisets

(e) the membership relation _ ∈ _ 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 set 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 their canonical forms are equal as terms modulo associativity and commutativity.

Now that the meaning of all these functions has been clarified, you are asked to define the equality property of `mt` and 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, and you should further test your definitions with other examples.

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
```

You can retrieve this module as a "skeleton" on which to give your answer from the course web page. Also, send a file with your module to `clarage2@illinois.edu`.