CS 476 Homework #2 Due 10:45am on 9/22

Note: Answers to the exercises listed below should be handed to the instructor *in hardcopy* and in *typewritten form* (latex formatting preferred) by the deadline mentioned above. You should also email the Maude code for the Problems 4 and 5 to `skeirik2@illinois.edu`.

1. Solve Exercise 93 in the August 28, 2013 version of *Set Theory and Algebra in Computer Science* (STAC).

2. Solve Exercise 103 in the August 28, 2013 version of *STAC*.

3. Note that we can think of a relation $R \subseteq A \times B$ as a "nondeterministic function from A to B." That is, given an element $a \in A$, we can think of its results, say aR, as the set of all b's such that $(a, b) \in R$. Unlike for functions, the set aR may be empty, or may have more than one element. For example, for $A = B = \{Peter, Paul, Sean, Meg, Dana\}$, the relation $\mathbf{father.of} = \{(Peter, Sean), (Peter, Meg), (Paul, Dana)\}$ has:

   - $(Peter)\mathbf{father.of} = \{Sean, Meg\}$
   - $(Paul)\mathbf{father.of} = \{Dana\}$, and
   - $(Sean)\mathbf{father.of} = (Meg)\mathbf{father.of} = (Dana)\mathbf{father.of} = \emptyset$.

   Note that the powerset $\mathcal{P}(B)$ allows us to view the "non-deterministic mapping" $a \mapsto aR$ as a *function* from A to $\mathcal{P}(B)$. More precisely, we can define $\_R$ as the function:

   $$\_R : A \ni a \mapsto \{b \in B \mid (a, b) \in R\} \in \mathcal{P}(B).$$

   But since this can be done for any relation $R \subseteq A \times B$, the mapping $R \mapsto \_R$ is then a function:

   $$\_[\_] : \mathcal{P}(A \times B) \ni R \mapsto \_R \in [A \to \mathcal{P}(B)].$$

   One can now ask an obvious question: are the notions of a relation $R \in \mathcal{P}(A \times B)$ and of a function $f \in [A \to \mathcal{P}(B)]$ *essentially the same*? That is, can we go *back and forth* between these two supposedly equivalent representations of a relation? But note that the idea of "going back and forth" between two equivalent representations is precisely the idea of a *bijection*.

   Prove that the function $\_[\_] : \mathcal{P}(A \times B) \ni R \mapsto \_R \in [A \to \mathcal{P}(B)]$ is bijective.

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

   $$\textit{Declarative Programming} = \textit{Mathematical Modeling}$$

   Specifically, of how you can model *discrete mathematics* in a computable way by functional programs in Maude, so that what you get is a *computable mathematical model* of discrete mathematics. Furthermore, it will allow you to obtain a *computable mathematical model of arrays and array lookup* as a special case of your model.

   Recall the function:

   $$\_[\_] : \mathcal{P}(A \times B) \ni R \mapsto \_R \in [A \to \mathcal{P}(B)]$$

   from Problem 3 above. Note that we then also have a function:

   $$\_[\_] : A \times \mathcal{P}(A \times B) \ni (a, R) \mapsto aR \in \mathcal{P}(B)$$

   that applies the function $\_R$ to an element $a \in A$ to get its image set under R.

   Define this latter function in Maude for $A = \mathbf{N}$ the set of natural numbers, and $B = \mathbf{Q}$ the set of rational numbers, and for *finite* relations $R \subset \mathbf{N} \times \mathbf{Q}$ by giving recursive equations for it in the functional module below.

Define also in the same functional module the auxiliary functions: `dom`, which assigns to each finite relation $R \subset \mathbf{N} \times \mathbf{Q}$ the set $dom(R) = \{n \in \mathbf{N} \mid \exists (n,r) \in R\}$, and the predicate `pfun`, which tests wether a relation $f \subset \mathbf{N} \times \mathbf{Q}$ is a *partial function*. That is, whether f satisfies the uniqueness condition:

$$(\forall n \in \mathbf{N})\ \ (\forall p, q \in \mathbf{R})\ \ [(n,p) \in f \wedge (n,q) \in f] \Rightarrow p = q.$$

In Computer Science a *finite* partial function $f \subset \mathbf{N} \times \mathbf{Q}$ is called an *array* of rational numbers, or sometimes a *map*. Note that when f is an array, the result $n[f]$ is either a single rational number, or, if f is not defined for the index n, then `mt`. That is, $n[f]$ is *exactly* array lookup, which usually would be denoted $f[n]$ instead than, as done here in a funkier way, $n\,f$. In summary, the function _[_] that you will define includes as a special case the *array lookup* function for arrays of rational numbers of arbitrary size.

Note: Notice Maude's built-in module `RAT` contains `NAT` as a submodule, and has a subsort relation `Nat < Rat`. You can use the automatically imported module `BOOL` and its built-in equality predicate `==` and if-then-else `if_then_else_fi` as auxiliary functions.

```
fmod RELATION-APPLICATION is protecting RAT .
  sorts Pair NatSet RatSet Rel .
  subsort Pair < Rel .
  subsort Nat < NatSet < RatSet .
  subsort Rat < RatSet .
  op [_,_] : Nat Rat -> Pair [ctor] .       *** Pair is cartesian product Nat x Rat
  op mt : -> NatSet [ctor] .                *** empty set of naturals
  op null : -> Rel [ctor] .                 *** empty relation
  op _,_ : NatSet NatSet -> NatSet [ctor assoc comm id: mt] .  *** union
  op _,_ : RatSet RatSet -> RatSet [ctor assoc comm id: mt] .  *** union
  op _,_ : Rel Rel -> Rel [ctor assoc comm id: null] .         *** union
  op _in_ : Nat NatSet -> Bool .           *** membership
  op _[_] : Nat Rel -> RatSet .            *** relation application to a number
  op dom : Rel -> NatSet .                 *** domain of a relation
  op pfun : Rel -> Bool .                  *** partial function predicate
  vars n m : Nat .  var r : Rat .  var P : Pair . var S : NatSet . var R : Rel .
  eq n,n = n .                             *** idempotency
  eq P,P = P .                             *** idempotency
  eq n in mt = false .                     *** membership


  eq n in m,S = (n == m) or n in S .       *** membership

  *** your equations defining the functions  _[_], dom, and pfun here
  *** if you need to declare any other variables or auxiliary
  *** functions besides those above, you can also do so

endfm
```

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 `skeirik2@illinois.edu`.

5. Consider the following module, that defines the usual strict order _<_ relation on natural numbers, and a `min` function for computing the smallest element of a multiset of numbers:

```
fmod NAT-MSET-MIN is
  sorts Nat NatMSet .
  subsort Nat < NatMSet .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _ _ : NatMSet NatMSet -> NatMSet [assoc comm ctor] .
```

```
  op _<_ : Nat Nat -> Bool .
  op min : NatMSet -> Nat .
  vars N M : Nat .
  var  S : NatMSet .
  eq 0 < s(N) = true .
  eq s(N) < 0 = false .
  eq s(N) < s(M) = N < M .
  eq min(N N S) = min(N S) .
  ceq min(N M S) = min(N S) if N < M .
  ceq min(N M) = N if N < M .
  eq min(N) = N .
endfm
```

This module has *two* functions that are not completely defined, that is, such that when invoked on some ground terms do not reduce to constructors. (**Note**: for the sort `Bool`, the constructors are of course `true` and `false`). Do the following:

- identify the two functions that fail to be fully defined by evaluating them on suitable ground term arguments.
- correct the specification by adding some extra equations (without modifying the present ones) so that all the functions in the module become completely defined.

Include both a screenshot of your evaluation of problematic expressions in the original module as well as the correct specification in your answer. Also, send a file with the correct module to `skeirik2@illinois.edu`. You can retrieve the module itself from the course web page.