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.


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 \( A = B = \{\text{Peter}, \text{Paul}, \text{Sean}, \text{Meg}, \text{Dana}\} \), the relation \( \text{father.of} = \{(\text{Peter}, \text{Sean}), (\text{Peter}, \text{Meg}), (\text{Paul}, \text{Dana})\} \) has:
   - \( (\text{Peter})\text{father.of} = \{\text{Sean}, \text{Meg}\} \)
   - \( (\text{Paul})\text{father.of} = \{\text{Dana}\} \)
   - \( (\text{Sean})\text{father.of} = (\text{Meg})\text{father.of} = (\text{Dana})\text{father.of} = \emptyset \).

   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 = 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 \( A = \mathbb{N} \) the set of natural numbers, and \( B = \mathbb{Q} \) the set of rational numbers, and for finite relations \( R \subset \mathbb{N} \times \mathbb{Q} \) by giving recursive equations for it in the functional module below.
Define also in the same functional module the auxiliary functions: \( \text{dom} \), which assigns to each finite relation \( R \subseteq \mathbb{N} \times \mathbb{Q} \) the set \( \text{dom}(R) = \{ n \in \mathbb{N} \mid \exists (n, r) \in R \} \), and the predicate \( \text{pfun} \), which tests whether a relation \( f \subseteq \mathbb{N} \times \mathbb{Q} \) is a partial function. That is, whether \( f \) satisfies the uniqueness condition:

\[
(\forall n \in \mathbb{N}) \ (\forall p, q \in R) \ [(n, p) \in f \land (n, q) \in f] \Rightarrow p = q.
\]

In Computer Science a finite partial function \( f \subseteq \mathbb{N} \times \mathbb{Q} \) 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 \( \text{mt} \). That is, \( n[f] \) is exactly array lookup, which usually would be denoted \( f[n] \) instead than, as done here in a funkier way, \( n \cdot f \). In summary, the function \( n[f] \) 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 \( \text{RAT} \) contains \( \text{NAT} \) as a submodule, and has a subsort relation \( \text{Nat} \prec \text{Rat} \). You can use the automatically imported module \( \text{BOOL} \) and its built-in equality predicate \( == \) and if-then-else \( \text{if_then_else_fi} \) as auxiliary functions.

fmod RELATION-APPLICATION is protecting \( \text{RAT} \).

sorts \( \text{Pair} \) \( \text{NatSet} \) \( \text{RatSet} \) \( \text{Rel} \).
subsort \( \text{Pair} \prec \text{Rel} \).
subsort \( \text{Nat} \prec \text{NatSet} \prec \text{RatSet} \).
subsort \( \text{Rat} \prec \text{RatSet} \).

op \( [_,_] \) : \text{Nat} \ \text{Rat} -> \text{Pair} [\text{ctor}] . \quad *** \text{Pair is cartesian product Nat x Rat}

op \( \text{mt} \) : -> \text{NatSet} [\text{ctor}] . \quad *** \text{empty set of naturals}

op \( \text{null} \) : -> \text{Rel} [\text{ctor}] . \quad *** \text{empty relation}

op \( _,_ \) : \text{NatSet} \text{NatSet} -> \text{NatSet} [\text{ctor assoc comm id: mt}] . \quad *** \text{union}

op \( _,_ \) : \text{RatSet} \text{RatSet} -> \text{RatSet} [\text{ctor assoc comm id: mt}] . \quad *** \text{union}

op \( _,_ \) : \text{Rel} \text{Rel} -> \text{Rel} [\text{ctor assoc comm id: null}] . \quad *** \text{union}

op \( \text{in}_\text{in} \) : \text{Nat} \text{NatSet} -> \text{Bool} . \quad *** \text{membership}

op \( \text{in}_\text{[]} \) : \text{Nat} \text{Rel} -> \text{RatSet} . \quad *** \text{relation application to a number}

op \( \text{dom} \) : \text{Rel} -> \text{NatSet} . \quad *** \text{domain of a relation}

op \( \text{pfun} \) : \text{Rel} -> \text{Bool} . \quad *** \text{partial function predicate}

vars \( n \quad m \) : \text{Nat} . \quad \text{var} \quad r \quad P \quad : \text{Rat} . \quad \text{var} \quad P \quad : \text{Pair} . \quad \text{var} \quad S \quad : \text{NatSet} . \quad \text{var} \quad R \quad : \text{Rel} .

eq \( n,n = n \) . \quad *** \text{idempotency}

eq \( P,P = P \) . \quad *** \text{idempotency}

eq \( n \text{ in } \text{mt} = \text{false} \) . \quad *** \text{membership}

eq \( n \text{ in } m,S = (n == m) \text{ or } n \text{ in } S \) . \quad *** \text{membership}

*** your equations defining the functions \( \text{in}_\text{[]} \), \( \text{dom} \), and \( \text{pfun} \) 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 \text{skeirik2@illinois.edu}.

5. Consider the following module, that defines the usual strict order \( \prec \) relation on natural numbers, and a \( \text{min} \) function for computing the smallest element of a multiset of numbers:

fmod NAT-MSET-MIN is

sorts \( \text{Nat} \) \( \text{NatMSet} \).
subsort \( \text{Nat} \prec \text{NatMSet} \).

op \( 0 \) : -> \text{Nat} [\text{ctor}] .

op \( s \) : \text{Nat} -> \text{Nat} [\text{ctor}] .

op \( _,_ \) : \text{NatMSet} \text{NatMSet} -> \text{NatMSet} [\text{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.