Note: Answers to the exercises listed below and the code solution for Exercise 2 should be emailed in typewritten form (latex formatting preferred) by the deadline mentioned above to nishant2@illinois.edu.

1. 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 the result of applying \( R \) to \( a \), let us denote it \( R\{a\} \), as the set of all \( b \)'s such that \( (a, b) \in R \). Unlike for functions, the set \( R\{a\} \) may be empty, or may have more than one element.

Note that the powerset \( \mathcal{P}(B) \) allows us to view the “non-deterministic mapping” \( a \mapsto R\{a\} \) as a normal function from \( A \) to \( \mathcal{P}(B) \). More precisely, we can define\(^1\) \( R\{\cdot\} \) as the function:

\[
R\{\cdot\} : A \ni a \mapsto \{b \in B \mid (a, b) \in R\} \in \mathcal{P}(B).
\]

But since this can be done for any relation \( R \subseteq A \times B \), the mapping \( R \mapsto R\{\cdot\} \) is itself a function:

\[
\{\cdot\} : \mathcal{P}(A \times B) \ni R \mapsto R\{\cdot\} \in [A \mapsto \mathcal{P}(B)].
\]

One can now ask an obvious question: are the notions of a relation \( R \in \mathcal{P}(A \times B) \) and of a function \( f \in [A \mapsto \mathcal{P}(B)] \) 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 \( \{\cdot\} : \mathcal{P}(A \times B) \ni R \mapsto R\{\cdot\} \in [A \mapsto \mathcal{P}(B)] \) is bijective.

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

\[
\text{Declarative Programming = Mathematical Modeling}
\]

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:

\[
\{\cdot\} : \mathcal{P}(A \times B) \ni R \mapsto R\{\cdot\} \in [A \rightarrow \mathcal{P}(B)]
\]

from Problem 1 above. Note that we then also have a function:

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

that applies the function \( R\{\cdot\} \) 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 \( R \subset \mathbb{N} \times \mathbb{Q} \) 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 \( f \subset \mathbb{N} \times \mathbb{Q} \) is a partial function. That is, whether \( f \) satisfies the uniqueness condition:

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

\(^1\)Note that the function \( R\{\cdot\} \) is closely related to the function

\[
R\{\cdot\} : \mathcal{P}(A) \ni A' \mapsto \{b \in B \mid a \in A' \land (a, b) \in R\} \in \mathcal{P}(B)
\]

defined in STACS, namely, by the equation: \( R\{a\} = R\{\{a\}\} \). We are using a different notation (\( R\{\cdot\} \) and \( R[\cdot] \)) to distinguish them.
In Computer Science a finite partial function \( f \subset \mathbb{N} \times \mathbb{Q} \) is called an array of rational numbers, or sometimes a map. Note that when \( f \) is an array, the result \( f\{n\} \) is either a single rational number, or, if \( f \) is not defined for the index \( n \), then \( \text{mt} \). That is, \( f\{n\} \) is exactly array lookup, which usually would be denoted \( f[n] \) instead. 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 \( \text{NAT} \) as a submodule, and has a subsort relation \( \text{Nat} < \text{Rat} \). 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.

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

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

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

\textbf{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 nishant2@illinois.edu.