CS 476 Homework #4 Due 10:45am on 2/23

Note: Answers to the exercises listed below (in typewritten form, preferably using Latex) as well as the Maude code for Problem 2, should be emailed by the above deadline to `reedoei2@illinois.edu`.

1. The addition function on the natural numbers:

$$+_{\mathbb{N}} : \mathbb{N}^2 \to \mathbb{N}$$

is a relation, and therefore a subset $+_{\mathbb{N}} \subset \mathbb{N}^2 \times \mathbb{N}$. In decimal notation this subset *cannot be explicitly described*: we need to invoke the addition algorithm to specify the set defined by this function. However, a nice feature of the Peano representation of the naturals is that $+_{\mathbb{N}}$ *can* be explicitly described as a set. It is the set:

$$+_{\mathbb{N}} = \{((n, 0), n) \in \mathbb{N}^2 \times \mathbb{N} \mid n \in \mathbb{N}\} \cup \{((n, s(m)), s^m(n)) \in \mathbb{N}^2 \times \mathbb{N} \mid n, m \in \mathbb{N}\}.$$

Consider now the following Maude functional module (in prefix notation):

```
fmod NATURAL is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op + : Nat Nat -> Nat .
  vars N M : Nat .
  eq +(N,0) = N .
  eq +(N,s(M)) = s(+(N,M)) .
endfm
```

Adopting the Peano notation, any natural number $n \in \mathbb{N}$ is *exactly* a constructor term in the above module, i.e., n is either 0, or has the form $s^k(0)$ for some $k \geq 1$.

You are asked to do two things:

(A). Prove the following theorem:

Theorem. For any $n, m \in \mathbb{N}$ in Peano notation, the term $+(n, m)$ has a unique *terminating* sequence of equality steps:

$$+(n, m) = t_1 = t_2 = \ldots = u$$

such that each step in the sequence is obtained by applying one of the two equations in `NATURAL` from left to right as a simplification rule.[1] Furthermore, the term u in which the sequence terminates is precisely the constructor term $+_{\mathbb{N}}(n, m) \in \mathbb{N}$.

Hint. Use induction!

(B). Use the above theorem to show that for (Σ, E) the equational theory specified by the above module `NATURAL`, its canonical term algebra $\mathbb{C}_{\Sigma/E}$ *is exactly* what one would expect: the algebra of the natural numbers $\mathbb{N}$ in Peano notation, with the *standard* interpretation for the symbols $\{0, s, +\}$.

2. This exercise is about using lists of naturals to represent sets of naturals [in an obviously *non-unique* way, since lists can have repeated elements, and, even if they do not, list elements may appear in different orders]. Specifically, you are asked to define functions:

[1] What this means is intuitively obvious: we have seen various examples. But, in any case, this process of left-to-right simplification has been formally defined as *term rewriting* with the rules $+(N, 0) \to N$ and $+(N, s(M)) \to s(+(N, M))$ in Lecture 5.

- `insert` to insert a number into a set.
- a predicate `_in_` to test whether a number belongs to a set.
- `_U_` to compute the union of two sets.
- `simplify` to obtain a list representation of a set that has no repeated elements.
- `_/\_` to compute the intersection of two sets.
- `_-_` to compute the difference of two sets.
- `equal-sets` to test whether or not two lists represent the same set.

You can do so by adding the needed equations defining such functions [plus those for any other auxiliary functions that you may need] to the module below, which imports `NAT`, the built-in naturals. This ensures that you have various functions, such as `if_then_else_fi`, order comparison between numbers, Boolean operations, and the built-in equality predicate `_==_` (for both numbers and lists) already available to you.

```
fmod LIST-REPRESENTATION-OF-SETS is
  protecting NAT .
  sort List .
  op nil : -> List [ctor] .
  op _;_ : Nat List -> List [ctor] .

  vars N M : Nat .  vars L L1 L2 : List .

  op insert : Nat List -> List .   *** inserts a number into a "set"

  *** add your equations here

  op _in_ : Nat List -> Bool .     *** "set" membership predicate

  *** add your equations here

   op _U_ : List List -> List .   *** "set" union

  *** add your equations here

   op simplify : List -> List . *** returns "set" with no repetitions

  *** add your equations here

  op _/\_ : List List -> List .        *** "set" intersection

  *** add your equations here

  op _-_ : List List -> List .        *** "set" difference

  *** add your equations here

  op equal-sets : List List -> Bool .  *** two lists represent the same set

  *** add your equations here

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 and tests cases to `reedoei2@illinois.edu`.