CS 476 Homework #9 Due 10:45am on 10/24

Note: Answers to the exercises listed below in *typewritten form* (latex formatting preferred) as well as code solutions should be emailed by the above deadline to `clarage2@illinois.edu`.

1. Consider the following three Maude functional modules for reversing lists: one whose reverse function inducts on the *left* of a string, a similar module that inducts on the *right* of the string and a third module that actually inducts on *both sides* of the string:

```
set include BOOL off .

fmod LIST-REV-L is
  sorts Nat NeList List .
  subsorts Nat < NeList < List .
  op 0 : -> Nat [ctor metadata "0"] .
  op s : Nat -> Nat [ctor metadata "1"] .
  op nil : -> List [ctor metadata "2"] .
  op _ _ : List List -> List [assoc metadata "3"] .
  op _ _ : NeList NeList -> NeList [ctor assoc metadata "3"] .
  op rev : List -> List [metadata "4"] .

   var n : Nat .  var L : List .

   eq L nil = L .
   eq nil L = L .

   eq rev(nil) = nil .
   eq rev(n) = n .
   eq rev(n L) = rev(L) n .
endfm

set include BOOL off .

fmod LIST-REV-R is
  sorts Nat NeList List .
  subsorts Nat < NeList < List .
  op 0 : -> Nat [ctor metadata "0"] .
  op s : Nat -> Nat [ctor metadata "1"] .
  op nil : -> List [ctor metadata "2"] .
  op _ _ : List List -> List [assoc metadata "3"] .
  op _ _ : NeList NeList -> NeList [ctor assoc metadata "3"] .
  op rev : List -> List [metadata "4"] .

   var n : Nat .  var L : List .

   eq L nil = L .
   eq nil L = L .

   eq rev(nil) = nil .
   eq rev(n) = n .
   eq rev(L n) = n rev(L) .
endfm
```

```
set include BOOL off .

fmod LIST-REV-G is
  sorts Nat NeList List .
  subsorts Nat < NeList < List .
  op 0 : -> Nat [ctor metadata "0"] .
  op s : Nat -> Nat [ctor metadata "1"] .
  op nil : -> List [ctor metadata "2"] .
  op _ _ : List List -> List [assoc metadata "3"] .
  op _ _ : NeList NeList -> NeList [ctor assoc metadata "3"] .
  op rev : List -> List [metadata "4"] .

   var n : Nat .  vars L L' : List .

   eq L nil = L .
   eq nil L = L .

   eq rev(nil) = nil .
   eq rev(n) = n .
   eq rev(L L') = rev(L') rev(L) .
endfm
```

Do the following:

(a) Assuming in the terminology of Lecture 14, pg. 13 that these three functional modules are *admissible*, prove that they are each *comparable* to the other two. No hand-waving blather will count as a "proof."

Note. Since admissiblity is assumed, you do not have to check it for each module, although this is an easy automatic check with the Maude tools. In particular, an RPO termination order has already been defined for each module using the `metadata` attribute. All you are asked to do is to prove *comparability* between each module and the other two.

(b) Prove that these three modules are *semantically equivalent* with the help of the NuITP. Explain why exactly your proof ensures semantic program equivalence. That is, on what theorem are you basing your claim that your proof using the NuITP does indeed prove semantic equivalence.

Warning. You should use the latest version of the NuITP, **Alpha 23** of October 13, 2023 available in the NuITP web page. This version corrects several bugs in the previous version and adds some new features (but such new features are not needed for this homework: the Manual for the NuITP Alpha 21 documents and illustrates with examples all the features that you may need). Be aware that caching issues in your browser may make it appear as if the **Alpha 23** isn't there: just refresh your browser a couple of times to see the link for it.

(c) Prove with the help of the NuITP **Alpha 23** that the above three modules (that is, their canonical term algebras) satisfy the inductive theorem:

$$rev(rev(L)) = L.$$

Explain why (based on what theorem(s)) your proof shows that the three modules satisfy this property.

Make sure to send to `clarage2@illinois.edu` *screenshots* of all your NuITP interactions for parts (b) and (c) of this problem, as well as a *proof script* for them (i.e., a list of the commands that you have given to the NuITP: see an example of a proof script in Problem 2 below)

Warning. Note that one of the typings for `_ _` is *not* a constructor. Therefore, do not be surprised if the NuITP will rejects a generator set that you have defined if it contains a term of the form $u\ v$ whose least sort is `LIst`.

Extra Credit. Parts (b) and (c) of this problem can be proved *together* by giving *only three commands* (besides `set` commands to enter goals and `genset` commands defining generator sets, which do not count as

inference steps in the proof) to the NuITP, of which only one of them is a `gsi!` command. You can get 50% extra credit on this problem if you can prove parts (b) and (c) with just three inference step commands of which only one is a `gsi!` command.

2. Recall the following `NATURAL-ARITH` functional module from Lecture 16:

```
set include BOOL off .

fmod NATURAL-ARITH is
  sorts Nat NzNat .
  subsort NzNat < Nat .
  op 0 : -> Nat [ctor metadata "1"] .
  op s : Nat -> NzNat [ctor metadata "2"] .
  op _+_ : Nat Nat -> Nat [metadata "3"] .
  op _*_ : Nat Nat -> Nat [metadata "4"] .
  op _*_ : NzNat NzNat -> NzNat [metadata "5"] .
  op _^_ : NzNat Nat -> NzNat [metadata "6"] .   *** exponentiation

  vars n m k : Nat .  vars n' k' m' : NzNat .

  eq n + 0 = n .
  eq n + s(m) = s(n + m) .
  eq n * 0 = 0 .
  eq n * s(m) = n + (n * m) .
  eq n' ^ 0 = s(0) .
  eq n' ^ s(m) = n' * (n' ^ m) .
endfm
```

In Lecture 16 this module was used to illustrate the *internalize and conquer* proof heuristic. To make your life easier, a *proof script* allowing you to reproduce the proofs of all the results proved for this module is included below:

```
set module NATURAL-ARITH .

genset SIND for Nat is 0 ;; s(N:Nat) .

set goal ((0 + Y:Nat = Y:Nat) /\ (s(X:Nat) + Y:Nat) = s(X:Nat + Y:Nat)) .

apply gsi! to 0 on $2:Nat .

internalize .

set goal X:Nat + (Y:Nat + Z:Nat) = (X:Nat + Y:Nat) + Z:Nat .

apply gsi! to 0 on $3:Nat .

internalize as assoc .

set goal (X:Nat + Y:Nat = Y:Nat + X:Nat) .

apply gsi! to 0 on $1:Nat .

internalize as comm .

set goal X:Nat * (Y:Nat + Z:Nat) = (X:Nat * Y:Nat) + (X:Nat * Z:Nat) .
```

```
apply gsi! to 0 on $2:Nat .

internalize .

set goal ((0 * Y:Nat = 0) /\ (s(X:Nat) * Y:Nat) = (Y:Nat + (X:Nat * Y:Nat))) .

apply gsi! to 0 on $2:Nat .

internalize .

set goal (Y:Nat + Z:Nat) * X:Nat = (Y:Nat * X:Nat) + (Z:Nat * X:Nat) .

apply gsi! to 0 on $2:Nat .

internalize .

set goal X:Nat * (Y:Nat * Z:Nat) = (X:Nat * Y:Nat) * Z:Nat .

apply gsi! to 0 on $3:Nat .

internalize as assoc .

set goal (X:Nat * Y:Nat = Y:Nat * X:Nat) .

apply gsi! to 0 on $2:Nat .

internalize as comm .

set goal (X:NzNat ^ (Y:Nat + Z:Nat) = (X:NzNat ^ Y:Nat) * (X:NzNat ^ Z:Nat)) .

apply gsi! to 0 on $2:Nat .
```

The goal of this problem is to give you the chance to have fun and experience the power of the *internalize and conquer* heuristic by doing the following:

(a) After having entered `NATURAL-ARITH` into Maude and having loaded the NuITP **Alpha 23** into Maude, run the above proof script on the NuITP, so that you get exactly to the point after the proof of the last goal for the property for the exponentiation function.

(b) Continue from that point and prove the following two additional properties of exponentiation, taking advantage as much as possible of the *internalize and conquer* proof heuristic:

```
(X:NzNat ^ (Y:Nat * Z:Nat) = (X:NzNat ^ Y:Nat) ^ Z:Nat) .

set goal (X:NzNat ^ Y:Nat)  ^ Z:Nat = (X:NzNat ^ Z:Nat) ^ Y:Nat .
```

Make sure to send to `clarage2@illinois.edu` a *screenshot* of all your NuITP interactions for part (b) of this problem as well as a *proof script* for it (that is, the, hopefully short, proof script that is the continuation of the above proof script).

Extra Credit. Part (b) of this problem can be accomplished by giving *only four commands* (besides `set` commands to enter goals, which do not count as inference steps) to the NuITP, of which only one of them is a `gsi!` comand. You can get 50% extra credit on this problem if you can prove part (b) with just four commands, of which only one of them is a `gsi!` command.