CS 476 Homework #11 Due 10:45am on 4/9

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 to `fanyang@illinois.edu` the Maude code for both exercises. For any difficulties using any of the tools, you can send email to `fanyang@illinois.edu`.

1. Consider the definition of binary trees with quoted identifiers on the leaves given in Lecture 13. Complete the module given below (available in the course web page) by defining with confluent and terminating equations the two functions called `leaves` and `inner`, that count, respectively, the number of leaf nodes of a tree, and the number of nodes in a tree that are *not* leaf nodes. For example, for the tree `(('a # 'b) # 'c) # 'd` there are 4 leaf nodes (namely `'a`, `'b`, `'c`, and `'d`), and 3 inner nodes (corresponding to the 3 different occurrences of the `#` operator).

```
fmod TREE is
  protecting QID .
  sorts Natural Tree .
  subsort Qid < Tree .
  op 0 : -> Natural [ctor].
  op s : Natural -> Natural [ctor].
  op _+_ : Natural Natural -> Natural [assoc comm].
  op _#_ : Tree Tree -> Tree [ctor] .
  ops leaves inner : Tree -> Natural .
  var I : Qid .
  vars N M : Natural .
  vars T T' : Tree .
  eq N + 0 = N .
  eq N + s(M) = s(N + M) .
  *** add equations for leaves and inner here
endfm
```

   Once you have defined and tested your definitions for `leaves` and `inner` do the following, *including screenshots for each tool used* in your hardcopy of the homework:

   - check that it is sufficiently complete using the SCC tool
   - state a theorem, in the form of a universally quantified equation, that gives a general law stating, for any tree `T`, the exact relation between the numbers `leaves(T)` and `inner(T)`
   - give a mechanical proof of that theorem using the ITP

2. Consider the following dining philosophers example, that you can retrieve from the course web page:

```
fmod NAT/4 is
   protecting NAT .
   sort Nat/4 .
   op [_] : Nat -> Nat/4 .
   op _+_ : Nat/4 Nat/4 -> Nat/4 .
   op _*_ : Nat/4 Nat/4 -> Nat/4 .
   op p : Nat/4 -> Nat/4 .
   vars N M : Nat .
```

```
   ceq [N] = [N rem 4] if N >= 4 .
   eq [N] + [M] = [N + M] .
   eq [N] * [M] = [N * M] .
   ceq p([0]) = [N] if s(N) := 4 .
   ceq p([s(N)]) = [N] if N < 4 .
endfm

mod DIN-PHIL is
   protecting NAT/4 .
   sorts Oid Cid Attribute AttributeSet Configuration Object Msg .
   sorts Phil Mode .
   subsort Nat/4 < Oid .
   subsort Attribute < AttributeSet .
   subsort Object < Configuration .
   subsort Msg < Configuration .
   subsort Phil < Cid .

   op __ : Configuration Configuration -> Configuration
                                                [ assoc comm id: none ] .
op _`,_ : AttributeSet AttributeSet -> AttributeSet
                                                [ assoc comm id: null ] .
   op null : -> AttributeSet .
   op none : -> Configuration .
   op mode`:_ : Mode -> Attribute [ gather ( & ) ] .
   op holds`:_ : Configuration -> Attribute [ gather ( & ) ] .
   op <_:_|_> : Oid Cid AttributeSet -> Object .
   op Phil : -> Phil .

   ops t h e : -> Mode .
   op chop : Nat/4 Nat/4 -> Msg [comm] .
   op init : -> Configuration .
   op make-init : Nat/4 -> Configuration .

   vars N M K : Nat .
   var C : Configuration .

   ceq init = make-init([N]) if s(N) := 4 .
   ceq make-init([s(N)])
     = < [s(N)] : Phil | mode : t , holds : none >  make-init([N])  (chop([s(N)],[N]))
     if N < 4 .
   ceq make-init([0]) =
      < [0] : Phil | mode : t , holds : none > chop([0],[N]) if  s(N) := 4 .

   rl [t2h] : < [N] : Phil | mode : t , holds : none > =>
      < [N] : Phil | mode : h , holds : none > .
   crl [pickl] :  < [N] : Phil | mode : h , holds : none > chop([N],[M])
       => < [N] : Phil | mode : h , holds :  chop([N],[M]) > if [M] = [s(N)] .
   rl [pickr] :  < [N] : Phil | mode : h , holds : chop([N],[M]) >
      chop([N],[K]) =>
      < [N] : Phil | mode : h , holds :  chop([N],[M]) chop([N],[K]) > .
   rl [h2e] :  < [N] : Phil | mode : h , holds :  chop([N],[M])
      chop([N],[K]) > => < [N] : Phil | mode : e ,
      holds :  chop([N],[M]) chop([N],[K]) > .
   rl [e2t] :  < [N] : Phil | mode : e , holds :  chop([N],[M])
      chop([N],[K]) >  =>   chop([N],[M]) chop([N],[K])
```

```
      < [N] : Phil | mode : t , holds :  none > .
endm
```

There are four philosophers, that you can imagine eating in a circular table. Initially they are all in thinking mode (`t`), but they can go into hungry mode (`h`), and after picking the left and right chopsticks (they eat Chinese food) into eating mode (`e`), and then can return to thinking.

The identities of the philosophers are naturals modulo 4, with contiguous philosophers arranged in increasing order from left to right (but wrapping around to 0 at 4). The chopsticks are numbered, with each chopstick indicating the two philosophers next to it.

Prove, by giving appropriate search commands from the initial state `init`, the following properties:

- (contiguous mutual exclusion): it is never the case that two *contiguous* philosophers are eating simultaneously.
- (mutual non-exclusion): it is however possible for two philosophers to eat simultaneously.
- (three exclusion): it is impossible for three philosophers to eat simultaneously.
- (deadlock) the system can deadlock.