CS 476 Homework #6 Due 10:45pm on 4/29

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 (`meseguer@cs.uiuc.edu`) the Maude code for the exercises requiring that.

1. 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:

(a) (contiguous mutual exclusion): it is never the case that two *contiguous* philosophers are eating simultaneously.

(b) (mutual non-exclusion): it is however possible for two philosophers to eat simultaneously.

(c) (three exclusion): it is impossible for three philosophers to eat simultaneously.

(d) (deadlock) the system can deadlock.

2. Consider the same dining philosophers example of Exercise 1, and the same properties (a)–(d) above that you verified by using search commands. You are now asked to define appropriate state predicates and to write four LTL formulas such that you verify properties (a)–(d) *exclusively by LTL model checking* those four formulas from the initial state.

   However, "verification" in some cases (for example for properties (b) and (d)) may not mean that the answer of the model checker is `true`. Instead, it may mean that the model checker gives you a *counterexample*, which precisely *proves* the desired property. In such cases, the LTL formula must in some way negate that the property you want is true, and then the counterexample is precisely the proof.

   Note that LTL formulas assert properties that are true for *all paths* from an intial state. Therefore, a counterexample can always be viewewed as a proof that *there exists a path* satisfying the *negation* of the property that is model checked. See for example Section 13.7 of the Maude book (Crossing the River (Revisited)) for an illustration of how a counterexample to a model checking problem can be used as a proof that there is a path with the desired property.