CS 476 Homework #7 Due 10:45am on Friday 12/6

Note: Answers to the exercises listed below should be emailed to `meseguer@illinois.edu`, including the Maude code and screenshots of commands for Exercise 3.

1. Answer the following questions:

   (a) Let $\Pi = \{p, q\}$. Explain — by giving in each case a counterexample trace $\tau \in [\mathbb{N} \to \mathcal{P}(\Pi)]$ such that, for ϕ the chosen wrong formalization, either $\tau \models p \rightsquigarrow q$ and $\tau \not\models \phi$ or $\tau \not\models p \rightsquigarrow q$ and $\tau \models \phi$ — why each of the following LTL formalizations ϕ of the "leads to" property $p \rightsquigarrow q$ is wrong:

   - $p \to q$
   - $\Box(p \to q)$

   (b) Write an LTL formula stating that for any path from the given initial state at only a finite number (possibly zero) of positions (i.e., steps) in such a path the property φ holds.

   (c) Write an LTL formula stating that for any path from the given initial state whenever φ holds at some step n, then it will also hold at all steps $n + 2k$ for any $k \in \mathbb{N}$.

   (d) Write an LTL formula stating that for any path from the given initial state, whenever φ holds at some step n, property ϕ will then hold at a strictly later step after that.

   (e) Write an LTL formula stating that for any path from the given initial state, whenever φ holds at some step n then ϕ does not hold at step n but will hold at a strictly later step after that.

2. Give a proof of the **Pattern Decomposition Lemma** in Lecture 24.

   Hint. Keep in mind that in any generator set $\{v_1, \ldots, v_m\}$ for sort s, the variables in the terms v_i are always *fresh*; that is, they are always disjoints from the variables of the pattern u to which the Pattern Decomposition Lemma is applied.

3. Consider the following system module, which specifies a 2-process version of Lamport's *Bakery* protocol:

```
fmod NAT-ACU is
  sort Nat .
  ops 0 1 : -> Nat [ctor] .
  op _+'_ : Nat Nat -> Nat [ctor assoc comm id: 0] .
endfm

mod BAKERY is
  protecting NAT-ACU .

  sorts Mode BState .

  ops sleep wait crit : -> Mode [ctor] .
  op <_,_,_,_> : Mode Nat Mode Nat -> BState [ctor] .
  op initial : -> BState .

  vars P Q : Mode .
  vars X Y Z : Nat .

  eq initial = < sleep, 0, sleep, 0 > .
```

```
  rl [p1_sleep] : < sleep, X, Q, Y > => < wait, Y +' 1, Q, Y > [narrowing] .
  rl [p1_wait] : < wait, X, Q, 0 > => < crit, X, Q, 0 > [narrowing] .
  rl [p1_wait] : < wait, X, Q, X +' Y > => < crit, X, Q, Y +' X > [narrowing] .
  rl [p1_crit] : < crit, X, Q, Y > => < sleep, 0, Q, Y > [narrowing] .

  rl [p2_sleep] : < P, X, sleep, Y > => < P, X, wait, X +' 1  > [narrowing] .
  rl [p2_wait] : < P, 0, wait, Y > => < P, 0, crit, Y > [narrowing] .
  rl [p2_wait] : < P, X +' Y +' 1, wait, Y > => < P, X +' Y +' 1, crit, Y > [narrowing] .
  rl [p2_crit] : < P, X, crit, Y > => < P, X, sleep, 0 > [narrowing] .
endm
```

where in the 4-tuple `< P, X, Q, Y >`, the state of process 1 is represented by `P, X` and that of process 2 by `Q, Y`. Use Maude and the folding narrowing and pattern subsumption methods explained in Lecture 24 and illustrated with examples in Lecture 25 to prove that the following two invariants hold for the above `BAKERY` module from the symbolic initial state `< sleep , X , sleep, X >`

- mutual exclusion, and
- deadlock freedom.