# CS 476 Homework #13 Due 10:45am on 12/1

**Note:** Answers to the exercises listed below should be emailed to `nishant2@illinois.edu` in *typewritten form* (latex formatting preferred) by the deadline mentioned above. You should also email to `nishant2@illinois.edu` your Maude code for the second exercise.

1. Answer the following questions:

   (a) Let $\Pi = \{p, q\}$ be a set of state predicates. Prove — by giving in each case a counterexample trace $\tau \in [\mathbb{N} \to \mathcal{P}(\Pi)]$— that for $\phi$ any of the formulas:

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

   $\phi$ is a *wrong* formalization of the "leads to" property $p \leadsto q$. Your counterexample trace should show that either $\tau \models p \leadsto q$ and $\tau \not\models \phi$ or $\tau \not\models p \leadsto q$ and $\tau \models \phi$.

   (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 a property $\varphi$ holds.

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

   (f) Let $\Pi = \{p, q\}$. Define traces $\tau_1, \tau_2 \in [\mathbb{N} \to \mathcal{P}(\Pi)]$ such that, instantiating $\varphi$ to $p$, and $\phi$ to $q$, $\tau_1$ satisfies property (d) but not property (e), and $\tau_2$ satisfies properties (d) and (e).

2. `QLOCK` is a mutual exclusion protocol proposed by K. Futatsugi, where the number of processes is unbounded. There are several versions of it. One of them was presented in Lecture 18. The version below is an even simpler specification of `QLOCK` in Maude:

```
in model-checker.maude

mod QLOCK is protecting NAT .
 sorts NatMSet NatList QState .
 subsorts Nat < NatMSet NatList .
 op mt : -> NatMSet [ctor] .
 op _ _ : NatMSet NatMSet -> NatMSet [ctor assoc comm id: mt] .
 op nil : -> NatList [ctor] .
 op _;_ : NatList NatList -> NatList [ctor assoc id: nil] .
 op <_|_|_|_> : NatMSet NatMSet NatMSet NatList -> QState [ctor] .

 op [_] : Nat -> NatMSet .    *** set of first n numbers
 op init : Nat -> QState .    *** initial state, parametric on n

 vars n i j : Nat .  vars L W C : NatMSet .  var Q : NatList .

 eq [0] = mt .
 eq [s(n)] = n [n] .
 eq init(n) = < [n] | mt | mt | nil > .

 rl [l2w] : < L i | W | C | Q > => < L | W i | C | Q ; i > .
 rl [w2c] : < L | W i | C | i ; Q > => < L | W | C i | i ; Q > .
 rl [c2l] : < L | W | C i | i ; Q > => < L i | W | C | Q > .
endm
```

Since this problem is about specifying and automatically verifying some LTL properties of `QLOCK` using Maude's LTL theorem prover, the file `model-checker.maude` should first be imported.

Recall that each process is identified by a natural number. The system's state has four components: three "rooms" where processes can be, and a queue. The leftmost room is a "lounge," where processes can stay before moving to the next room, the "waiting room," and from there they may move to the "critical room," which means that they get access to some critical resource. To ensure mutual exclusion, a process in the lounge must register its name at the end of the waiting queue when entering the waiting room (rule `[l2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished using the resources implicitly modeled by being in the critical section, it can go back to the lounge (rule `[c2l]`).

As mentioned above, the number of processes is unbounded. However, the total number of processes in a given state never changes. This means that, instead of considering a single initial state, we can consider a *parametric family* of initial states `init(n)`, which has exactly `n` processes, intially all in the "lounge," with the other "rooms" and the queue intially empty. Verification of LTL properties by using Maude's LTL model checker can be made for different choices of the parameter `n`. You are asked to *formally specify and verify* four important LTL properties of `QLOCK` (in the last two cases, not a single formula, but 6 of them in each case), from the initial state `init(6)`, namely:

- **Mutual Exclusion**, i.e., the critical section is always either empty or has at most one process,
- **Deadlock Freedom**
- **Non-Starvation 1**: each process $i$, with $0 \leq i \leq 5$ that is waiting infinitely often will enter the critical section infinitely often (you should have to verify six formulas, one for each choice of process name $i$). This is an example of a *process fairness* property: the fairness is localized to each process.
- **Non-Starvation 2**: each process $i$, with $0 \leq i \leq 5$ that **is not a loafer** will enter the critical section infinitely often. Of course, being a **loafer** is a metaphor; but it is easy to explain: A process $i$ in the lounge will be a "loafer" if at a certain time it ceases to participate in the protocol: It may have participated a few times before, and even gotten sometimes to the critical room; but, being "lazy," at some point it gets tired of working and remains sitting in the lounge indefinitely, doing nothing. This is another example of a *process fairness* property.

**Hints**: (1) You have already seen several examples of *parametric state predicates* in lectures. For the two non-starvation properties, although not strictly necessary, it can be convenient for you to use parametric state predicates in your formulas. (2) As shown by various examples in the lectures, to define the semantics of state predicates you do not need to give equations for all the cases: some remaining cases (for example, when the given state predicate is `false`) can be handled by an equation that uses Maude's `[owise]` feature.

Just for your convenience, here is a template that you can fill in to solve this problem:

```
in model-checker.maude

mod QLOCK is protecting NAT .
 sorts NatMSet NatList QState .
 subsorts Nat < NatMSet NatList .
 op mt : -> NatMSet [ctor] .
 op _ _ : NatMSet NatMSet -> NatMSet [ctor assoc comm id: mt] .
 op nil : -> NatList [ctor] .
 op _;_ : NatList NatList -> NatList [ctor assoc id: nil] .
 op <_|_|_|_> : NatMSet NatMSet NatMSet NatList -> QState [ctor] .

 op [_] : Nat -> NatMSet .   *** set of first n numbers
 op init : Nat -> QState .   *** initial state, parametric on n

 vars n i j : Nat .  vars L W C : NatMSet .  var Q : NatList .

 eq [0] = mt .
 eq [s(n)] = n [n] .

 eq init(n) = < [n] | mt | mt | nil > .

 rl [l2w] : < L i | W | C | Q > => < L | W i | C | Q ; i > .
 rl [w2c] : < L | W i | C | i ; Q > => < L | W | C i | i ; Q > .
 rl [c2l] : < L | W | C i | i ; Q > => < L i | W | C | Q > .
endm

mod QLOCK-PREDS is protecting QLOCK .
including SATISFACTION .
subsort QState < State .

*** define here your state predicates and define their
*** semantics by means of equations; you can use [owise] in some of them.

endm

mod QLOCK-CHECK is
protecting QLOCK-PREDS .
including MODEL-CHECKER .
endm

*** give here the LTL model checking commands for each of your formulas.
```