CS 476 Homework #12 Due 10:45am on 4/28

Note: Answers to the exercises listed below, and the Maude code for exercises requiring it, should be emailed to `abir2@illinois.edu`.

1. The purpose of this exercise is to help you explore some *theorems* of LTL, so that you can get a better feeling for the logic and for what it means for an LTL formula to always be true. The theorems in question are very general: they do not depend on the choice of atomic propositions AP, and hold for any Kripke structures and choices of initial states in such Kripke structures.

   This poses a small technical problem. How are we going to *write* an LTL formula if we do not even *know* what its atomic propositions AP are? The answer is: instead of writing a *concrete* formula, we can write a *formula schema*. For example, $\bigcirc\varphi$ and $\varphi\,\mathcal{U}\,\psi$ *are* formula schemas, where φ and ψ are not LTL formulas, but *formula meta-variables* that can be instantiated to any LTL formula in the algebra of formulas $LTL(AP)$ for any choice of AP. So, how can we formalize LTL formula schemas? We just see LTL as a parameterized functional module (in Maude, parameterized by the `TRIV` theory), so that when the `TRIV` parameter is instantiated by a Maude *view* to a specific set AP of atomic propositions, then the instance is precisely the algebra $LTL(AP)$. So, what are LTL formula schemas? They are exactly elements of the *free algebra* $LTL(X)$, where X is an infinite set of variables X, except that, instead of using normal letters for the variables in X like $x, y, z, \ldots$, we use instead *Greek* letters like $\varphi, \phi, \psi, \mu, \ldots$ for the variables. Let us denote by F a formula schema $F \in LTL(X)$. Then, for each choice of atomic propositions AP, what F describes is the family of LTL formulas $F\theta$ for each ground substitution $\theta \in [Y \to LTL(AP)]$, where $Y = \mathit{vars}(F)$. Therefore, the LTL theorems F that we will discuss are very general: they apply to all the instances $F\theta$ of theirs meta-variables, for all choices of AP and of θ.

   So, what does it mean to say that F is a (meta-)*theorem* of LTL? This can be captured as follows:

   Definition. An LTL formula schema $F \in LTL(X)$ is a (meta-)*theorem* of LTL, denoted $\models F$, iff for each set AP of atomic propositions, Kripke structure $\mathcal{K} = (A, \to_{\mathcal{K}}, L)$ on AP, choice of initial state $a \in A$, and ground substitution $\theta \in [Y \to LTL(AP)]$, where $Y = \mathit{vars}(F)$, we have

   $$\mathcal{K}, a \models F\theta.$$

   Instead, $F \in LTL(X)$ is *not* a (meta-)*theorem* of LTL, denoted $\not\models F$, iff there exist set AP of atomic propositions, Kripke structure $\mathcal{K} = (A, \to_{\mathcal{K}}, L)$ on AP, choice of initial state $a \in A$, and ground substitution $\theta \in [Y \to LTL(AP)]$ such that $\mathcal{K}, a \not\models F\theta$.

   You are asked to prove that the following formula schemas are theorems ($\models$) or fail to be theorems ($\not\models$) of LTL:

   - $\models\ \varphi\,\mathcal{U}\,\psi \leftrightarrow \psi \vee (\varphi \wedge \bigcirc(\varphi\,\mathcal{U}\,\psi))$
   - $\models \diamond\psi \leftrightarrow \psi \vee (\bigcirc(\diamond\psi))$
   - $\models \Box\psi \leftrightarrow \psi \wedge (\bigcirc(\Box\psi))$
   - $\models\ \varphi\,\mathcal{U}\,\psi \to \diamond\psi$
   - $\not\models\ \diamond\psi \to (\varphi\,\mathcal{U}\,\psi)$.

   Note that the first three theorems are both quite interesting and very practical, because they amount to *recursive definitions* of the connectives $\mathcal{U}$, $\diamond$ and $\Box$, so that we can "unroll" those connectives after taking one-step transitions.

Hints: Theorems of the form $\models F \rightarrow G$ can be proved by assuming any path π starting at $a \in A$ in an arbitrary Kripke structure $\mathcal{K}$ on arbitrary AP, and θ, and, assuming $\pi \models F\theta$, then showing that we then must have $\pi \models G\theta$. Obviously, theorems of the form $F \leftrightarrow G$ can be proved by proving $F \rightarrow G$ *and* $G \rightarrow F$. Non-theorems of the form $\not\models F \rightarrow G$ can be proved by choosing concrete AP, $\mathcal{K}$ on AP, $a \in A$, and θ, and a path π starting at a such that $\pi \models F\theta$, but $\pi \not\models G\theta$. Another way to prove a theorem $\models F$ if we have already proved another theorem $\models G$ is to show that there is a (typically non-ground) substitution $\gamma \in [Y \rightarrow LTL(X)]$ where $Y = vars(G)$ such that $F = G\theta$, or at least we already know by the definition of the LTL and Boolean connectives that $F \leftrightarrow G\theta$. That is, we just need to know that F is a less general *special instance* of G up to some known formula equivalence. More generally, since theorems of the form $\models F \leftrightarrow G$ are formula equivalences that can be used as *equations* $F = G$, we can use previously proved or known equivalences as equalities to prove other equivalences. This second method can of course save substantial amounts of work by reusing previous proved or known results.

2. The unordered communication protocol between sender and receiver objects in the `COMM` module below is a slight variant of the example sketched out in Lecture 15. The only differences are: (1) the buffers now are not separate objects: they are attributes of sender and receiver objects; and (2) the sender, before it sends the next item in its buffer, awaits until after receiving an `ack` from the receiver for the previous item. You may want to run a few tests cases. For example, those given in the rewrite commands after the `COMM` module, to get a better feeling for how this protocol works. If you wish to see a detailed trace of the executions, you can type in Maude:

```
Maude> set trace on .
```

The point about this exercise is to give you an opportunity to put into practice the ideas we have been discussing about model checking LTL properties, and also to take advantage of the usefulness of parametric state predicates, by model checking in the Maude LTL model checker some useful properties. Specifically, you should express in LTL and model check two properties, each of which uses one of the two parametric states predicates whose operator declarations are given in the `COMM-PREDS` module. The properties are the following:

- Any initial state of the form `init(A,B,L)` has the state predicate `Inv(A,B,L)` (which you are asked to define) as an *invariant*, where `Inv(A,B,L)` essentially states that in `init(A,B,L)` and in any state reachable from it, if `L1` is the list in the buffer of sender `A`, and `L2` is the list in that of receiver `B`, then we always have the equality:

  ```
  L = L2 ; in-trans(MS) ; L1
  ```

  where `MS` is the current list of messages in the configuration (which could be `none`), and `in-trans(MS)` is a function that you are asked to define. That is, `Inv(A,B,L)` being an invariant means that the protocol achieves *in-order-communication* in spite of being asynchronous.

- Any initial state of the form `init(A,B,L)` always terminates in a state characterized by a state predicate `Term(A,B,L)` (which you are asked to define) where: (1) `L` is in the buffer of `B`, (2) the buffer of `A` is `nil`, (3) the attribute `ack-w:` of `A` is `false`, (4) the value of the `cnt:` attibute in both `A` and `B` is exactly the length of `L` , and (5) there are no pending messages in the final configuration. This is the best possible *progress/liveness* property of the type "something good happens" that one might wish: the protocol always terminates, and terminates as expected.

In summary, you are asked to: (a) equationally define the `Inv(A,B,L)` and `Term(A,B,L)` parametric state predicates, as well as the auxiliary function `in-trans(MS)`; (b) express the above invariant as well as the termination property as LTL formulas based on those parametric predicates; and (c) use the LTL model checker to prove that these properties are true for three initial states of the form `init(A,B,L)` where `A` is `'a`, `B` is `'b`, and `L` is, respectively: `(1 ; 2 ; 3)`, `(1 ; 2 ; 3 ; 4)`, and `(1 ; 2 ; 3 ; 4 ; 5)`.

```
in model-checker

fmod NAT-LIST is protecting NAT .
 sort List .
```

```
 subsorts Nat < List .
 op nil : -> List .
 op _;_ : List List -> List [assoc id: nil] .
 op length : List -> Nat .
 var L : List .
 var N : Nat .
 eq length(nil) = 0 .
 eq length(N ; L) = s(length(L)) .
endfm

mod COMM is protecting NAT-LIST . protecting  QID .
 sorts Oid Class Object Msg Msgs Att Atts Configuration .
 subsort Qid < Oid .
 subsort Att < Atts .     *** Atts is set of attribute-value pairs
 subsort Object < Configuration .
 subsorts Msg < Msgs < Configuration .
 op none : -> Msgs [ctor] .
 op __ : Configuration Configuration -> Configuration
                       [ctor config assoc comm id: none] .
 op __ : Msgs Msgs -> Msgs
                       [ctor config assoc comm id: none] .
 op null : -> Atts .
 op _,_ : Atts Atts -> Atts [ctor assoc comm id: null] .
 op buff:_ : List -> Att [ctor] .
 op snd:_ : Oid -> Att [ctor] .
 op rec:_ : Oid -> Att [ctor] .
 op cnt:_ : Nat -> Att [ctor] .
 op ack-w:_ : Bool -> Att [ctor] .
 ops Sender Receiver : -> Class [ctor] .
 op <_:_|_> : Oid Class Atts -> Object [ctor] .
 msg to_from_val_cnt_ : Oid Oid Nat Nat -> Msg [ctor] .
 msg to_from_ack_ : Oid Oid Nat -> Msg [ctor] .
 op init : Oid Oid List -> Configuration .
 vars N M : Nat .
 var L : List .
 vars A B : Oid .

 rl [snd] : < A : Sender | buff: (N ; L), rec: B, cnt: M, ack-w: false >
   =>
    (to B from A val N cnt M)
    < A : Sender | buff: L, rec: B, cnt: M, ack-w: true > .

 rl [rec] : < B : Receiver | buff: L, snd: A, cnt: M >
      (to B from A val N cnt M)
    =>
      < B : Receiver | buff: (L ; N), snd: A, cnt: s(M) >
      (to A from B ack M) .

 rl [ack-rec] : < A : Sender | buff: L, rec: B, cnt: M, ack-w: true >
    (to A from B ack M)
    =>
    < A : Sender | buff: L, rec: B, cnt: s(M), ack-w: false > .

 eq init(A,B,L) = < A : Sender | buff: L, rec: B, cnt: 0, ack-w: false >
                  < B : Receiver | buff: nil, snd: A, cnt: 0 > .
```

```
endm

rew init('a,'b,(1 ; 2 ; 3)) .

rew init('a,'b,(1 ; 2 ; 3 ; 4)) .

rew init('a,'b,(1 ; 2 ; 3 ; 4 ; 5)) .

mod COMM-PREDS is
 protecting COMM .
 including SATISFACTION .
 subsort Configuration < State .
 ops Inv Term : Oid Oid List -> Prop .
 op in-trans : Msgs -> List .

 var MS : Msgs . var C : Configuration .  vars L L1 L2 : List .
 vars A B : Oid .  vars N M : Nat .  var T : Bool .

*** include here your equational definition of in-trans(MS)

*** include here your equational definition of Inv(A,B,L)

*** include here your equational definition of Term(A,B,L)

endm


mod COMM-CHECK is
protecting COMM-PREDS .
including MODEL-CHECKER .
including LTL-SIMPLIFIER .
endm

*** give here your LTL model checking commands for the invariant
*** and the termination property for the three initial states mentioned above.
```