CS 476 Homework #7 Due 10:45am on 12/1

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 Stephen Skeirik (`skeirik2@illinois.edu`) the Maude code for the exercises requiring that. Also, for any difficulties using any of the tools, you can send email to Stephen Skeirik and ask for his help.

Also note that the LTL model checker tool is included by default with any recent Maude distribution in the file `model-checker.maude`. Finally, note the Maude Reachability Logic Prover is not yet available on the course webpage; it will be posted sometime next week. In the meantime, to avoid wasting time, please work through the first homework problem.

1. Recall Exercise 4 in Homework 6, where the notion of *parametric invariant* was explored for the `COMM-CHANN` example. This exercise extends these ideas to the LTL case to make you familiar with the use of *parametric state predicates* in LTL. In particular, proving a parametric invariant $I(S, x_1, \ldots, x_n)$ from an initial state $init(x_1 : A_1, \ldots, x_n : A_n)$ can, when the set of reachable states is finite, also be done by model checking an LTL formula of the form $\Box I(x_1, \ldots, x_n)$, were now we have defined I as a parametric atomic proposition of the form $I : A_1 \ldots A_n \rightarrow Prop$ and have defined its meaning by giving equations using the `SATISFACTION` module.

   In the slightly modified version of `COMM-CHANN` below you are asked to do two things:

   - define again and prove by LTL model checking the parametric invariant `in-order-reception` for the initial states `init1`, `init2`, and `init3` below, and
   - define a new parametric atomic proposition called `received` that holds when the protocol has finished if the exact same list originally held by the sender is now in the buffer of the receiver, and state and model check the instance of the LTL parametric property expressing that such a state is indeed always reached for initial states `init1`, `init2`, and `init3` below.

   To make things easier for you, we provide a template indicating what you need to define. This template is available on the course webpage in the file `comm-chan-ltl.maude`.

```
mod COMM-CHANN is protecting NAT .  protecting QID .
sorts Msg MsgMSet QidList CHState .
subsort Msg <  MsgMSet .
subsort Qid < QidList .

op nil : -> QidList [ctor] .
op _;_ : QidList QidList -> QidList [ctor assoc id: nil] .
op [_,_] : Qid Nat -> Msg [ctor] .
op null : -> MsgMSet [ctor] .
op _ _ :  MsgMSet MsgMSet -> MsgMSet [ctor assoc comm id: null] .
op [_:_|_|_:_] : QidList Nat MsgMSet Nat QidList -> CHState [ctor] .

vars N M I J : Nat . var MS : MsgMSet . vars L L' : QidList .
vars A B : Qid .

rl [send] : [A ; L : I | MS | J : L'] => [L : s(I) | [A,I] MS | J : L'] .
rl [receive] : [L : I | MS [A,J] | J : L'] => [L : I | MS | s(J) : L' ; A] .
endm
```

```
load model-checker.maude

mod COMM-CHANN-PREDS is protecting COMM-CHANN . including SATISFACTION .
subsort CHState < State .
op in-order-reception : QidList -> Prop .
op received : QidList -> Prop .

vars N M I J : Nat . var MS : MsgMSet . vars L L' L'' : QidList .

*** add your defining equations here

endm

mod COMM-CHANN-CHECK is
protecting COMM-CHANN-PREDS .
including MODEL-CHECKER .
ops init1 init2 init3  : -> CHState .
eq init1 = ['a ; 'b ; 'c : 0 | null | 0 : nil ] .
eq init2 = ['a ; 'b ; 'c ; 'd : 0 | null | 0 : nil ] .
eq init3 = ['a ; 'b ; 'c ; 'd ; 'e : 0 | null | 0 : nil ] .
endm

*** add your LTL commands here
```

2. In the course we have discussed various mutual exclusion algorithms, verified through both model checking and also deductive methods. In this exercise, you will have the chance to use reachability logic to verify a simple token-based mutual exclusion algorithm for yourself. Consider the module `2TOKEN` below:

```
mod 2TOKEN is
  sorts Name Proc Token Conf State .
  subsorts Proc Token < Conf .
  op {_}      : Conf      -> State [ctor] .
  op none     :           -> Conf  [ctor] .
  ops * $     :           -> Token [ctor] .
  ops a b     :           -> Name  [ctor] .
  op __       : Conf Conf -> Conf  [assoc comm id: none] .
  op [_,wait] : Name      -> Proc  [ctor] .
  op [_,crit] : Name      -> Proc  [ctor] .
  var C : Conf .
  rl [a-enter] : { $ [a,wait] C } => { [a,crit]   C } .
  rl [b-enter] : { * [b,wait] C } => { [b,crit]   C } .
  rl [a-exit]  : {   [a,crit] C } => { [a,wait] * C } .
  rl [b-exit]  : {   [b,crit] C } => { [b,wait] $ C } .
endm
```

The specification defines two processes and two two tokens. Each process must consume its respective token to enter the critical section and emit a different token when it exits the critical section.

Of course, we would like to verify that this mutual exclusion algorithm is correct, i.e. that it actually satisfies an invariant *Mutex* where *Mutex* specifies that the critical section is mutually exclusive for each process.

However, note that for any reasonable initial state, `2TOKEN` is a *never-terminating* system. Thus, according to the semantics of reachability logic, *all* formulas will be valid! Thus, to verify the invariant *Mutex*, we will need to apply the corollary discussed in lectures 23 and 24, i.e.

Corollary 1.
If $[\![S_0]\!] \subseteq [\![B]\!]$ *and* $B \longrightarrow^{\circledast} [B]$ *holds in* $\mathcal{R}_{stop}$*, then* B *is an* invariant *for* $\mathcal{R}$ *from initial states* S_0*.*

Let us assume initial state S_0 = `{$ [a,wait] [b,wait]}`. In this exercise, using the stub file `token.maude` provided on the course webpage, you are asked to:

(a) Construct the module `2TOKEN-stop` with an appropriate "stopwatch" as illustrated in lecture 23

(b) Construct a pattern formula T which specifies the terminating states in `2TOKEN-stop`

(c) Construct a pattern formula *Mutex* which specifies the mutual exclusion property for `2TOKEN`

(d) Using the Maude Reachability Logic Prover as shown in lecture 24, apply the corollary to prove *Mutex* is an invariant from S_0 in the module `2TOKEN-stop` with terminating state set T, i.e.

   i. Prove $[\![S_0]\!] \subseteq [\![Mutex]\!]$

   ii. Prove $Mutex \longrightarrow_T^{\circledast} [Mutex]$

HINTS:

- Reference the command grammar frequently as you use the tool; unfortunately, the parser is quite picky.
- After editing the file `token.maude` appropriately, be sure to load your edited module *before* you load `rltool.maude`. Otherwise, you may not be able to load `token.maude`.
- When entering reachability formulas into the tool, pay careful attention to which topmost bracketing operator (e.g. `{}` or `[]`) you are using.