CS 476 Homework #11 Due 10:45am on 4/21

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

1. Consider the following system module, whose purpose is to generate all permutations of a list `L` as the final states reachable by rewriting with the rules in the module the initial state `perm(L)`. Note that all functions in the module, except for the `l2mset` function, are *constructors*. In particular, `perm` is also a constructor term. This is because the permutations of `L` are not computed by "evaluating" `perm(L)` with some *equations*, but by changing instead the initial state `perm(L)` to other states by *rewrite rules*.

   You are asked to specify the rewrite rules (three rules are actually enough) that will make it the case that the final states reachable from `perm(L)` are exactly the permutations of `L`. Some sample search computations and the number of solutions you should get in each case are included for your convenience. Note that if a list has length n and all its elements are different, then there are $n!$ permutations of it.

```
*** if perm(L) is the initial state, then each final state is a permutations of L

mod PERMUTATIONS is protecting QID .
  sorts List State MSet .
  subsort Qid < List < State .
  subsort Qid < MSet .
  op nil : -> List [ctor] .
  op _:_ : List List -> List [ctor assoc id: nil] .
  op mt : -> MSet [ctor] .
  op __ : MSet MSet -> MSet [ctor assoc comm id: mt] .
  op l2mset : List -> MSet .            *** converts a list to a multiset
  op perm : List -> State [ctor] .      *** perm(L) initial state, final states all L permutations
  op [_,_] : List MSet -> State [ctor] . *** list-multiset pairs

  var I : Qid .  var L : List .  var S : MSet .

  eq l2mset(nil) = mt .
  eq l2mset(I : L) = I l2mset(L) .

*** define here the transitions from perm(L) by some rules, so that the final
*** states reachable from perm(L) are exactly the permutations of L

endm

search perm(nil) =>! L .                    *** 1 solution
search perm('a) =>! L .                     *** 1 solution
search perm('a : 'b) =>! L .                *** 2 solutions
search perm('a : 'b : 'c) =>! L .           *** 6 solutions
search perm('a : 'b : 'c : 'd) =>! L .      *** 24 solutions
search perm('a : 'b : 'c : 'd : 'd) =>! L . *** 60 solutions
search perm('a : 'b : 'c : 'd : 'e) =>! L . *** 120 solutions
```

2. Suppose that $\mathcal{K} = (A, \rightarrow_{\mathcal{K}}, L)$ is a Kripke structure on a set AP of atomic propositions. Then, any state predicate/atomic proposition $p \in AP$ defines the set of states $[\![p]\!]$ where p holds as the set $[\![p]\!] = \{a \in A \mid p \in$

$L(a)\}$. Since $p \in L(a)$ iff $\mathcal{K}, a \models p$, the notion of the set of states where p holds can be generalized to any LTL formula φ on AP, so that we can define the set of states $[\![\varphi]\!]$ where φ holds as the set $[\![\varphi]\!] = \{a \in A \mid \mathcal{K}, a \models \varphi\}$. The goal of this exercise is to help you become familiar with specifying LTL properties of a concurrent system. Given a Kripke structure $\mathcal{K} = (A, \rightarrow_{\mathcal{K}}, L)$ on a set AP of atomic propositions, and an initial state $a \in A$, write a temporal logic formula ψ such that:

- $\mathcal{K}, a \models \psi$ holds iff, for any path π starting at a, a given LTL formula φ holds only on a *finite (and non-zero) number of states* of π. **Hint**: just to help you think about finding such a formula ψ (which depends of course on the given φ) you may begin by assuming that φ is just an atomic proposition p. The general case is totally similar, but considering first the case when φ is p may be helpful.
- $\mathcal{K}, a \models \psi$ holds iff, for any path π starting at a, a given LTL formula φ holds on an *infinite number of states* of π.
- $\mathcal{K}, a \models \psi$ holds iff, for any path π starting at a, a given LTL formula φ_1 holds in all states of the form $\pi(2n)$ for any $n \in \mathbb{N}$ *and* another given LTL formulas φ_2 holds on all states of the form $\pi(1 + 3n)$ for any $n \in \mathbb{N}$.
- $\mathcal{K}, a \models \psi$ holds iff, for any path π starting at a, for given LTL formulas φ_1 and φ_2, there is a *non-zero* $j \in \mathbb{N}$ such that φ_1 holds on $\pi(i)$ for $0 \leqslant i < j$, and φ_2 holds for any $\pi(n)$ such that $n \geqslant j$. **Warning**: To get this right, you might wish to think carefully about the corner cases in the semantic definition of the $\mathcal{U}$ operator.

For Extra Credit. (Five more points added to Exercise 2: if you did everything perfectly in Exercise 2 and in this extra part, you get 15 points instead of just 10). LTL formulas are *universally quantified on paths* in an *implicit* manner:

$$\mathcal{K}, a \models \varphi \quad \Leftrightarrow \quad \forall \pi \in \mathit{Path}(\mathcal{K})_a \;\; \pi \models \varphi.$$

If we wanted to make this *explicit*, we could write $\forall\, \varphi$ instead of just φ. How about a formula $\exists\, \varphi$ existentially quantified on paths? Does this make sense? This makes perfect sense. We can define:

$$\mathcal{K}, a \models \exists\, \varphi \quad \Leftrightarrow \quad \exists \pi \in \mathit{Path}(\mathcal{K})_a \;\; \pi \models \varphi.$$

Such formulas, and in general arbitrary nestings of $\forall$ and $\exists$ path quantifiers, belong to a richer temporal logic[1] called CTL^*, which contains LTL as a sublogic. However, if keeping the nesting of quantifiers straight in one's head is not easy for first-order logic, it is *even harder* for temporal logic: LTL is much simpler to understand than CTL^*, not only for engineers, but even for logicians!

So, let us not go too far, and stick to the much simpler LTL. Still, here is an important *trick* question. What do we *know* when for an LTL formula φ we have $\mathcal{K}, a \not\models \varphi$? Do we then know that $\mathcal{K}, a \models \neg\varphi$? Not at all! This does *not* follow, and is a crass logical confusion. What we *do* know is:

$$\mathcal{K}, a \not\models \varphi \quad \Leftrightarrow \quad \neg(\forall \pi \in \mathit{Path}(\mathcal{K})_a \;\; \pi \models \varphi) \quad \Leftrightarrow \quad \exists \pi \in \mathit{Path}(\mathcal{K})_a \;\; \pi \models \neg\varphi \quad \Leftrightarrow \quad \mathcal{K}, a \models \exists \neg\varphi.$$

So, who cares? All of us should. This is not a useless logical *divertimento*, but a *very practical* piece of knowledge. Why? Because when you are trying to prove $\mathcal{K}, a \models \varphi$ and an LTL model checker gives you a *counterexample path*, it is not just giving you a *bug*, but is actually giving you a *constructive proof* that $\mathcal{K}, a \models \exists\, \neg\varphi$ holds. So what? So you can *use* an LTL model checker not only to verify LTL formulas, but *also* to *prove* formulas of the form $\exists\, \varphi$. How? By model checking the LTL formula $\neg\varphi$ from initial state a: $\mathcal{K}, a \models \neg\varphi$, *and* getting a counterexample that *you want to get* and is not a *bug* at all: it is the *proof* you wanted. For a very amusing and non-trivial example of how you can use the Maude LTL model checker in this way, you can take a look at Section 13.7 (Crossing the River) of the *All About Maude* book.

Here is the extra credit problem. Suppose that $p \in AP$ is a state predicate. Then, $[\![p]\!]$ denotes the set of states of $\mathcal{K}$ where p holds. The property that from an initial state a we can *reach* some state in $[\![p]\!]$ is *not* expressible as an LTL formula φ, but it *is* expressible as an existential path formula $\exists\, \varphi$. Do two things: (a) write such a formula $\exists\, \varphi$, and (b) write the LTL formula ψ such that a counterexample to model checking $\mathcal{K}, a \models \psi$ is a *proof* that some state in $[\![p]\!]$ is reachable from a in $\mathcal{K}$.

[1] In CTL^* $\forall$ (resp. $\exists$) is written $\mathbf{A}$ (resp. $\mathbf{E}$), but this is just a matter of notation.