Assignment 9:

1. It is possible to get to a state where started holds, but ready does not hold.

**EF(started && !ready))**

1. For any state, if a request (of some resource) occurs, then it will eventually be acknowledged.

**AG(request-> AF (acknowledged))**

1. A certain process is enabled infinitely often on every computation path.

**AG(AF enabled)**

1. Whatever happens, a certain process will eventually be permanently deadlocked.

**AF(AG deadlocked)**

1. From any state, it is possible to get to a restart state.

**AG(EF restart)**

1. An upwards travelling elevator at the second floor does not change its direction when it has passengers wishing to go to the fifth floor.   
   Use predicates such as direction == up, floor == 2, and Button5Pressed.

**AG((direction==up & floor==2 & Button5Pressed) -> A[direction==up U floor==5])**

1. The elevator can remain idle on the third floor with its doors closed.

**AG(floor==3 && idle && door==closed -> EG(floor==3 && idle && door==closed))**

1. After switchClosed becomes true, valueOpen is never true.

AG(switchClosed->AG(!valueOpen))

*Note: this is same as our in class solution but I just added AG in front of it*

(Y) A[!switchClosed U (switchClosed && AG(!valueOpen))]

Strange, but I found somewhat similar (not exactly same) example from online reading. It sounds like ‘after’ represents as “A[!x U (x &&”. So in this case, ‘After switchClosed = true’ mean ‘in all case, switchClosed = false holds until ‘switch is closed and value is never open. Not too sure about this answer.

(M) My original answer is EF(switchClosed -> AG(!valueOpen))

The statement says nothing about state when !switchClosed so I prefer to avoid it.

1. After q, p is not true until r.

AG(q->A[!p U r])

(Y) A[!q U (q && A[!p U r])]

Again, ‘After’ is problem factor. Maybe the book has more accurate answer. I will let you know.

(M) q -> AF(!p U r)

I also agree that ‘After’ is a problem. I feel like I need a predicate at the beginning, just not sure what.

1. Variable toggle varies between true and false on successive states.

AG( (toggle==true -> AX(toggle==false)) & (toggle==false -> AX(toggle==true)))

(Y) I agree, but probably need more reading on recursive.

(M) A[toggle -> AX(!toggle)]

Not sure if that is legal syntax, but no matter if toggle==true or false it is always the opposite in the next state.

8. After switchClosed becomes true, valueOpen is never true.

I think we all agreed on 'switchClosed -> AG(!valueOpen)' part, but I guess it's either adding AG or EF to wrap it.

I will go with EF.

EF(switchClosed -> AG(!valueOpen))

9. After q, p is not true until r.

I would put predicate at the beginning. so

EF(q -> A[!p U r])

10. Variable toggle varies between true and false on successive states.

I ilke Mike's answer so I will go with

A[toggle -> AX(!toggle)]