Brenden Nelson
ID # 11635716

350 HW 7

(7) 1) I denote regexp = (□ (yellow ∨ ◇blue))
I can first create a FA, M_G which will represent the graph G. Next, I can create a FA, M_{regexp} which will represent the regular expression above. By running a cartesian product on $M_G \times M_{regexp} = M$. We form a new FA, M which will represent the original graph such that it satisfies the given conditions. The FA, M will only accept four different color sequences.

1) All colors are yellow. (in sequence)
2) All colors in sequence are yellows and blue.
3) Colors in sequence appear at random but always followed by a blue.
4) Colors in sequence appear at random, until at a point, when the color is blue, the sequence only shows yellows for the rest of the ∞ sequence.

I can then run on SCC on M to check that there is an w walk on Graph G such that regexp is satisfied. If SCC returns true, then the path exists.

2) I denote regexp = (□◇(yellow ∨ ◇blue))
I can first create a FA, M_G which represents Graph G. Next, I create another FA, M_{regexp} which represents the regular expression above. By runing a cartesian product on $M_G \times M_{regexp} = M$, we form a new FA, M which represents the original graph such that it satisfies the given conditions on the original graph G. This would require all successful walks to have infinitely many blues and infinitely many yellows. This is because the regular expression requires infinitely many points to be either yellow, or eventually blue.
I can run on SCC on M to check that there is an w walk on Graph G such that regexp is satisfied. If SCC returns true, then the path exists.

(2) In order to satisfy the given conditions, there must be a point on the w walk of Graph G such that after that point, there is infinitely many red points, but no more blue points.
First, I would create a FA, M_G to represent the graph G. Next, I would remove all blue nodes from M_G to obtain the FA, M_{G-B} which represents graph G but without any blue nodes. I would then run an SCC on the new graph M_{G-B} to determine if an w walk exists satisfying the given requirements. If the SCC returns true, then M_G would have an w-path on which there are no blue nodes and at least one red node. Thus, A path exists satisfying the conditions.

(3) [redacted] good w-path

I begin by thinking of the graph as having two parts, α and β.

α: The walk from initial to some point v in which #red = #blue.

β: The walk from v looping back to itself also with #red = #blue nodes.

Due to the looping nature of β, I can only design my algorithm to focus on the α walk from start to node v.

I define $\hat{L}$ as the set of all sequences of nodes on which #red = #blue with v as the start node and v as the end node.

I can construct a PDA, $\hat{M}$ to accept $\hat{L}$.

I can then ~~select~~ let M be the FA obtained from modifying graph G to add a garbage node accepting state. I can run Cartesian product on the FA, M and the PDA, $\hat{M}$ to obtain a new PDA, m.

$$m = M \times \hat{M}$$

I construct $L(m) \neq 0$ iff there is a walking from v to V on which the #blue = #red.

To determine if this is true, I can check to see if the PDA m is empty. m is empty iff it can be accepted by an empty language. If $L(m)$ returns Yes or an empty language, then m is empty. If PDA m is empty, then we can determine that the #red = #blue nodes, and our algorithm would return true.

(4) To Decide whether there exists a bad w-path on Graph G, I have to design an algorithm to look for two cases.

1) We have a finite number of red nodes

2) We have an infinite number of red nodes

For case 1) we know there must be a point after which there are no more red nodes.

For case 2) we know that there must be an infinite number of series which satisfy $(\ldots red \ldots red \ldots red \ldots red \ldots red)^*$

I construct a regexp to show the requirements.

I define $\Sigma_{garbage}$: all colors except red.

$regexp = (\Sigma_{garbage}^* \cdot red \cdot \Sigma_{garbage}^* \cdot red \cdot \Sigma_{garbage}^* \cdot red \cdot \Sigma_{garbage}^* \cdot red \cdot \Sigma_{garbage}^* \cdot red \cdot \Sigma_{garbage}^*)$

knowing the above information, I first start by creating three FA's. M_G, representing the Graph G, $M_{red\%5}$, representing the Paths with red nodes at a multiple of 5, and M_{regexp}, representing all paths satisfying the regular expression above.

I then run a Cartesian product $M_G \times M_{red\%5} \times M_{regexp} = M$.

To obtain a FA, M which will satisfy all the requirements to satisfy a bad w-path. After modifying M to include bad accepting state, I then run DFS on M from own init to accepting to see if a walk exists.

This will return true if either case 1) or case 2) exists as a bad w-walk on Graph G.