Eric Altenburg ## Problem Set 2 ##

9/10/19 CS-334

Pledge: I pledge my honor that I have abided by the Stevens Honor System. - Eric Altenburg

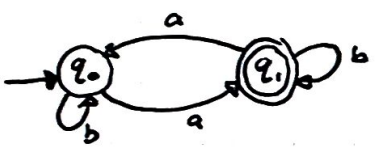
1. $L = \{w: w \text{ has an odd number of a's and ends with b}\}$

a) This can be broken up into languages L_1 and L_2.

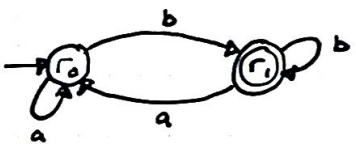
$L_1 = \{w: w \text{ has an odd number of a's}\}$

$L_2 = \{w: w \text{ ends with b}\}$

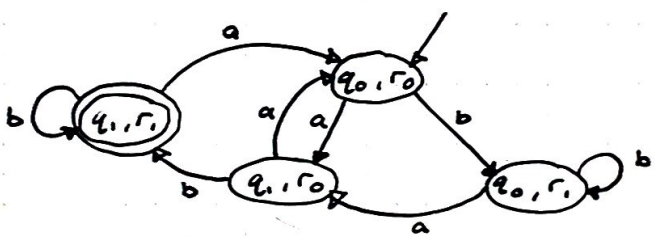
b) FSA M_1 for L_1



FSA M_2 for L_2



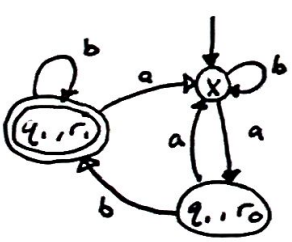
c) combine M_1 and M_2 to make M that recognizes L using cross-product method



d) Merge 2 states in M to reduce size.

Combine states q_0, r_0 and q_0, r_1 to form the new state X.

Now M looks like:



The reason as to why states (q_0, r_0) and (q_0, r_1) can be merged is because the outgoing transitions from each state do not have much connectivity to the other 2 states as they do themselves. The 'b' transitions between them stay in the two states, and both 'a' transitions lead to the same state.

2. For any language A, let $A^R = \{w^R \mid w \in A\}$. Show that if A is regular so is A^R.

Modify the machine that recognizes A to be an NFA that recognizes A^R. Since an NFA accepts A^R it will be considered regular. The way to modify said original machine is by:

① reversing all transitions between states, and if a transition leading from a states goes into itself will remain the same in that those specific transitions remain unchanged.

② Add an additional state q_{start}

③ Have a ε arrow pointing from the state q_{start} to all the old accept states (one ε arrow for each old accept state)

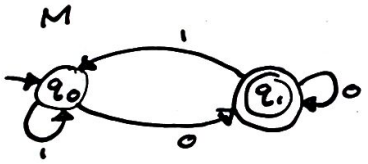
④ Make q_{start} the new start state

⑤ Make all old accept states regular states

⑥ Turn the old start state into an accept state.

Example: FSA that only recognizes inputs that end w/ a 0. called M.

M

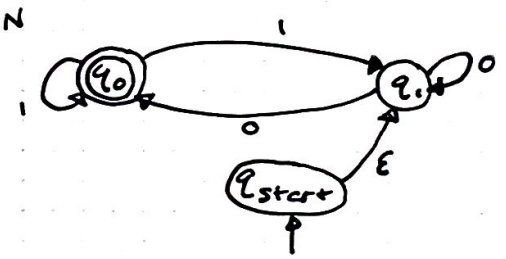


0110010 ✓

010011 X

Follow above procedures to get NFA N:

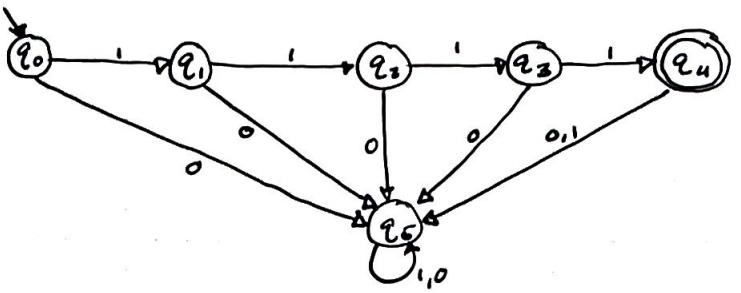
N



0100110 ✓ ← should accept if starts w/ a '0' since reversed.

1100110 X should not accept as it starts w/ a '1'

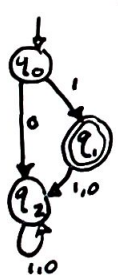
3. a) $L_4 = \{1111\}$



No, you cannot reduce this below 4 states. Any language w/ N amount of the same symbol concatenated together will always require N states to satisfy the transitions for all symbols of the string, then an additional accept state along w/ another final state connected to all the other states acting as a throw away for any other input that isn't '1', or additional input after being in the accept state. as the machine shouldn't accept this. The reason for N states for the transitions is because anything less would require a state to either loop back to itself or another state which would not work as the machine could potentially accept a string w/ more than 4 symbols. which shouldn't be allowed.

b) for all $K \geq 3$, there is a language that can be accepted by a K-state FSA that cannot be recognized by any FSA with fewer states.

FSA M



$K=3$

$L_1 = \{1\}$

Using the same style of language above – consecutive symbols that are the same, appended to each other, – it will always require N+2 states where N is the amount of symbols. This is shown by gradually removing the symbols one at a time until a base case is reached. For example, instead of 4 '1's concatenated, make it 3. That will require 5 states. Then try 2 '1's, which is 4 states, and finally, remove the last '1' leaving one '1'. The bare minimum states required is 3; one for the start state and to hold the '1' transition, another for the accept, and finally, a "throw away" state for all other input that should not be accepted. If you were to remove the last '1' then the language would not be the same as an empty string should not be accepted. If less than 3 states are used, a loop would occur and break the machine.