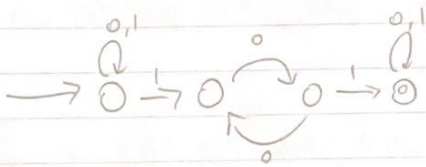
Nevin Liang

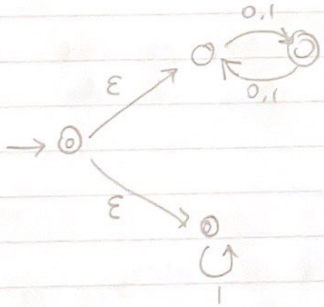
1. 00000111 ✓ 1110 ✓
1 ✓ 1011 ✓
0 ✗ 101 ✗
10 ✓ 1010 ✗
111 ✓
100 ✓
1000 ✓

the DFA recognizes the language of strings with an odd # of 1's over the alphabet {0,1}

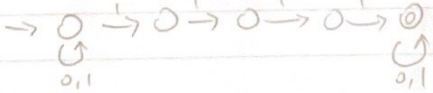
2. a. there EXISTS a pair of 1's separated by odd # 0's



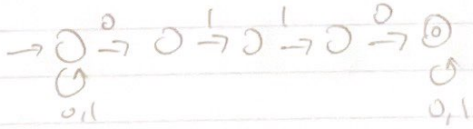
b.



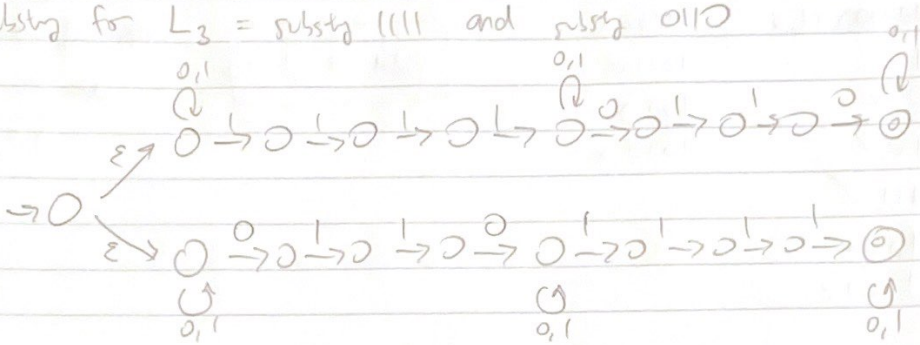
3. a) NFA N_1 for L_1 = substring of 1111



NFA N_2 for L_2 = substring of 0110



substg for L_3 = substg 1111 and substg 0110



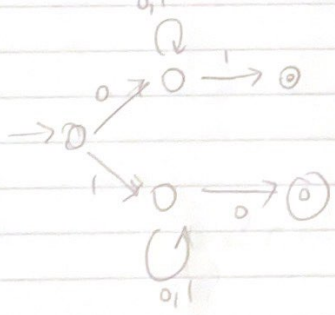
L_1 & L_2 & L_3 all regular b/c you can create NFAs for them

strgs containg 1111 or 0110 but not both is:

$L_4 = (L_1 \cup L_2) \setminus L_3$ $\cup$ and $\setminus$ are ops that reg langs closed under

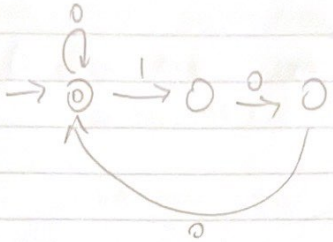
so L_4 regular ☐

b.



NFA creatable so regular ☐

c.



If there is no 1, it still works.

NFA created → language regular.

4. Let DFA D represent L: $D = (Q, \Sigma, \delta, q_0, F)$

NFA N represent L': $N = (Q', \Sigma, \delta', \quad)$

for each $q \in Q$, Q' contains both $(q,0)$ and $(q,1)$ signifying odd(1) or even(0) index.

Σ same

$\delta'((q,0),\sigma) = (\delta(q,\sigma), 1)$

$\delta'((q,1),\sigma) = (\delta(q,\sigma), 0)$

for all $q \in Q$ and $\sigma \in \Sigma$

there is an easier way

M be language with no 0 in odd # position (assume first element is odd # position)

M is regular s/c this NFA represents it

$\overline{M}$ is at least one 0 in odd # position and is regular s/c complement is valid.

$L' = L \cap \overline{M}$ aka set of all strings in L that

contain at least 1 0 in odd # posn

$\wedge$ is a valid symbol also reg langs closed under $\wedge$ op.

thus, L' is regular

5. Let the langage of Ds be plotters be D'

You can create the NFA for D' by taking the DFA D and removing all loops on the same state keeping everything else the same

This NFA should accept the same states, start from the same state, have all the states be same only a diff set of transitions

Since NFA creatable $\rightarrow$ D' is regular

6. (1) take DFA D and identify states & transition arrows.

(2) states are rewritten as vertices.
transition arrows are rewritten as directed edges.

(3) now we have a directed graph.

(4) BFS, DFS, or Flood fill the graph, recording lengths of all paths that end in an accept state.

(5) if there is an even # in your list return True else return FALSE.

Caveat: there might be a loop or cycle to cause infinite recursion to alleviate this, we DFS w/ a visited array and run cycle detection via DFS.

an even length cycle contributes nothing and an odd length cycle → the path can have odd or even length depending on # of times you cycle.

odd length cycle → if path exists to accept state return true.

even length cycle → continue DFS and use visited array to prevent further cycling on that cycle.

7. L's DFA $D(Q, \Sigma, \delta, q_0, F)$ L' 's NFA $N(Q', \Sigma, \delta', (q_0, 0), F')$

even # of flips

odd # of flips

$$\delta' = \begin{cases} \delta'((q,0), \overline{\sigma}) = (\delta(q,\sigma), 1) \\ \delta'((q,1), \overline{\sigma}) = (\delta(q,\sigma), 0) \\ \delta'((q,0), \sigma) = (\delta(q,\sigma), 0) \\ \delta'((q,1), \sigma) = (\delta(q,\sigma), 1) \end{cases}$$

Q': $\forall q \in Q \rightarrow (q,0)$ & $(q,1)$ in Q'

F': $\forall f \in F \rightarrow (f,1)$ in F'

Make a duplicate DFA and create edges so that δ' is followed. In addition label states in the first DFA w/ 0 as the second element of the tuple and 1 for the second DFA.

Since N' is NFA, L' is regular. □