COMPSCI 2AC3 Assignment 2

Prakhar Saxena

26 February 2024

Contents

1 Question 1

1.1 Part A

Instead of breaking the NFA into 2 different NFA's I used null transitions (ϵ). I use null transitions to move onto states q2 and q5. There is a self loop at state q5 for processing 'a' and the final state in that part of the NFA is state q6 for processing 'b', this part of the NFA accepts (a*b). The left hand side we basically creating a self looping NFA for (abc)*. It transitions to q2 with epsilon and self loops in states q3 and q4 by processing 'a', 'b' and 'c'.

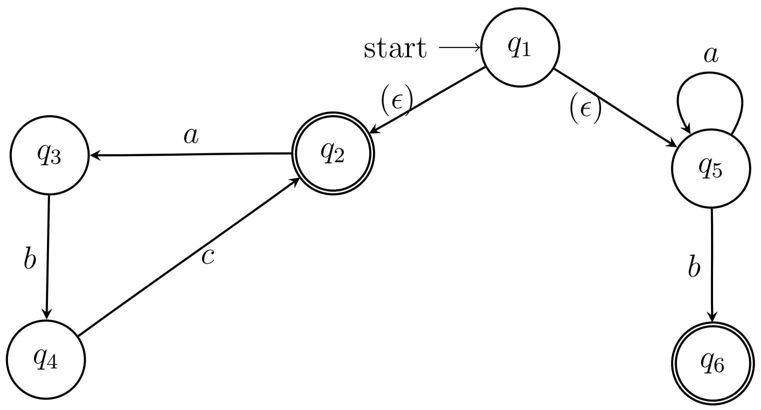

Figure 1: NFA for A

1.2 Part B

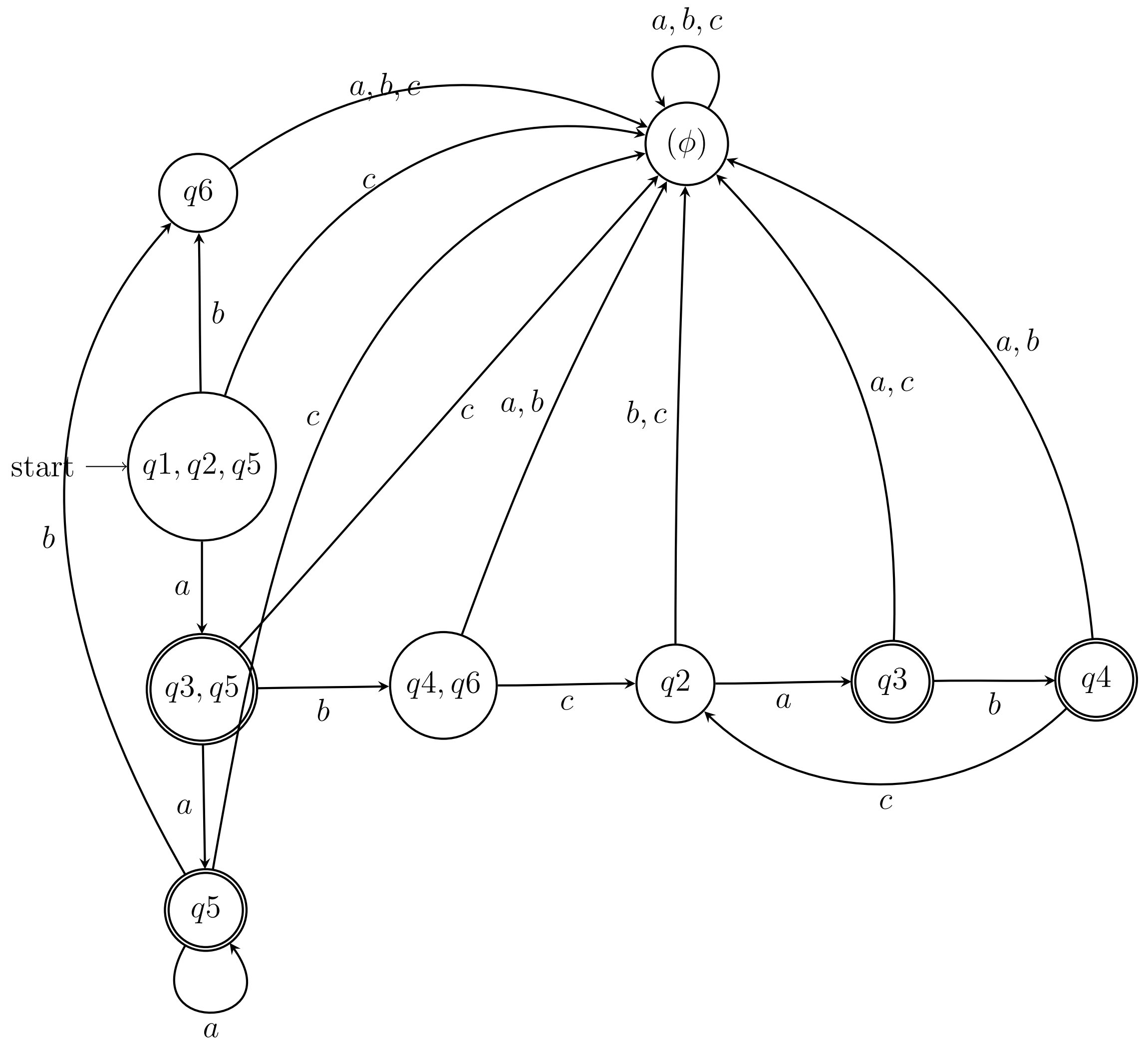

Figure 2: DFA M

1.3 Part C

Regex for the DFA using recursion:

$$\alpha^{Q}_{\{q_1,q_5,q_2\},\{q_5,q_3\}} + \alpha^{Q}_{\{q_1,q_5,q_2\},q_5} + \alpha^{Q}_{\{q_1,q_5,q_2\},q_3} + \alpha^{Q}_{\{q_1,q_5,q_2\},q_4}$$

Simplified: $a + \alpha^{Q}_{\{q_1,q_5,q_2\},q_5} + abca + \alpha^{Q}_{\{q_1,q_5,q_2\},q_4}$

Next we find $\alpha^{Q}_{\{q_1,q_5,q_2\},q_5}$ using recursion:

$$\alpha^{Q}_{\{q_1,q_5,q_2\},q_5} = \alpha^{Q-\{q_5,q_3\}}_{\{q_1,q_5,q_2\},q_5} + \alpha^{Q-\{q_5,q_3\}}_{\{q_1,q_5,q_2\},\{q_5,q_3\}}(\alpha^{Q-\{q_5,q_3\}}_{\{q_5,q_3\},\{q_5,q_3\}})^*\ \alpha^{Q-\{q_5,q_3\}}_{\{q_5,q_3\},q_5}$$

Simplifying:

$$\alpha^{Q}_{\{q_1,q_5,q_2\},q_5} = \phi + a(\epsilon)^* a^* \implies \alpha^{Q}_{\{q_1,q_5,q_2\},q_5} = a(a)^*$$

Next we finf$\alpha^{Q}_{\{q_1,q_5,q_2\},q_4}$ using recursion:

$$\alpha^{Q}_{\{q_1,q_5,q_2\},q_4} = \alpha^{Q-q_2}_{\{q_1,q_4,q_2\},q_4} + \alpha^{Q-q_2}_{\{q_1,q_5,q_2\},q_2}(\alpha^{Q-q_2}_{q_2,q_2})^*\ \alpha^{Q-q_2}_{q_2,q_4}$$

Simplifying:

$$\alpha^{Q}_{\{q_1,q_5,q_2\},q_4} = \phi + abc(abc)^*ab \implies \alpha^{Q}_{\{q_1,q_5,q_2\},q_4} = abc(abc)^*ab$$

Final regex is: $a + a(a^*) + abca + abc(abca^*)ab$

2 Question 2

The use of epsilon transitions is crucial to the proof. These transitions effectively simulate the recognition of substrings, enabling the automaton to jump non deterministically between states without consuming input symbols. By structuring the NFA in this way, any string that is a substring of a string in A will be accepted by M because the automata can non deterministically choose to start and end the substring at any point within the string. Since we can build such an NFA for B, and NFAs are equivalent to regular languages, B is also a regular language.

If A is a regular set then the set B composed of all substrings from strings in A is also regular. This argument is constructed on the foundation of a DFA M_A, that recognizes the regular set A. The DFA's paths denoted as P_i each correlate to a subset of strings that lead to an accepting state forming individual regular sets R_i. These regular sets together define the language $L(M_i)$ which is recognized by paths P_i. The set B is then represented as a union of these regular languages R_i equating to $B = R_1 \cup R_2 \cup ... \cup R_n$.This principle holds even when some paths P_i lead to non-accepting states; the union of the corresponding empty languages with the rest still results in a regular set.

Thus, since the union of regular sets R_i forms B, and because regular languages are closed under the operation of union, B retains the property of being regular.

3 Question 3

The below given DFA is designed to accept any string made up of alphabets $\Sigma = a, b$ of length less than or equal to 100.

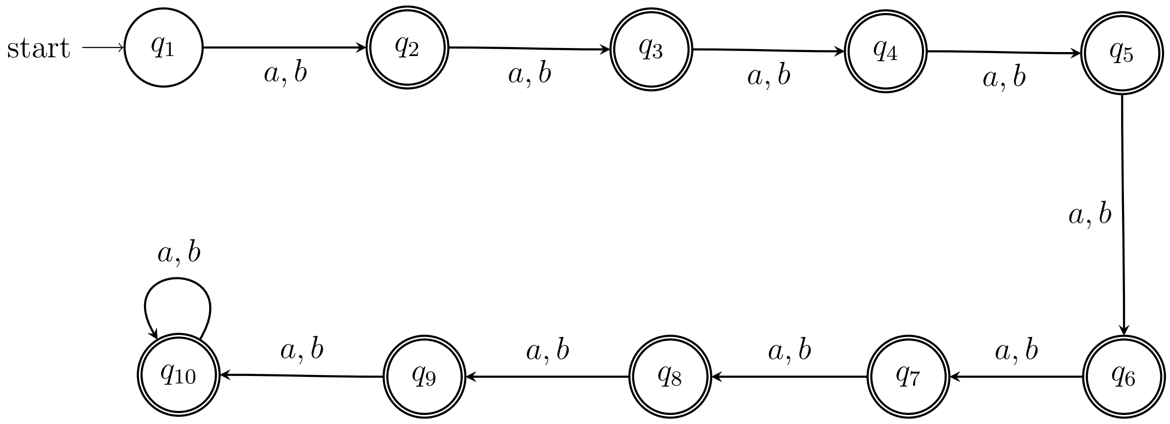

Figure 3: DFA M_1

The below given DFA is made to accept any string made up of alphabets $\Sigma = a, b$.

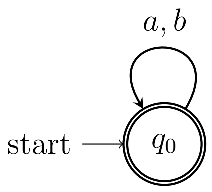

Figure 4: DFA A

Considering an input x = aaabaab. Clearly, x $\in M_1$ and x $\in$ A. However, this does not prove that L(M_1) = A because, both the DFA's are extremely different.

We could also prove that by giving a counter-example, let x be a string of 200 characters. This input would work for A but not M_1 once again, proving that L(M_1)=A is false.