Automata Programming Assignment

Assessment video:

https://iiitaphyd-my.sharepoint.com/:v:/g/personal/hrishi_narayanan_research_iiit_ac_in/EUJLrRxVqyZDpFl4tyuLP4MB7841s6IyDkoOMt7solzz6g?e=wjF7Pu

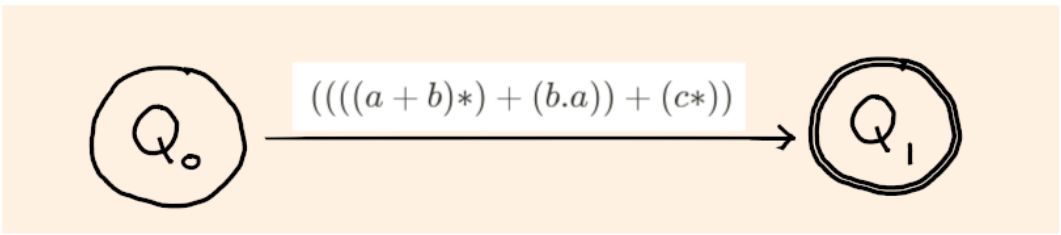
Question 1: Regular Expression → NFA

Algorithm:

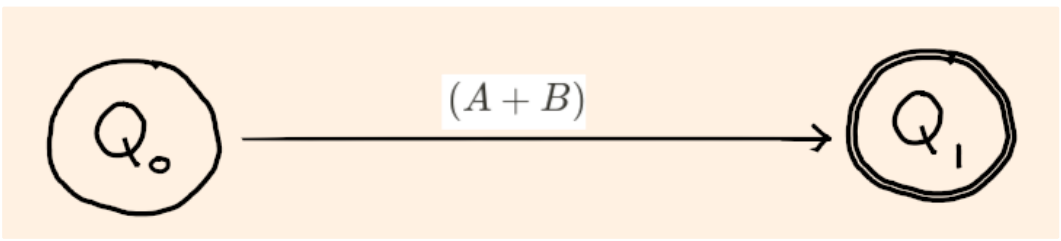
- Input is taken as a regex. This regex is parsed such that concatenation is denoted by "." operation, parthesis are applied in right to left order and each paranthesis contains only one operation. For example:

$$((a+b)*+ba+c*) \rightarrow ((((a+b)*)+(b.a))+(c*))$$

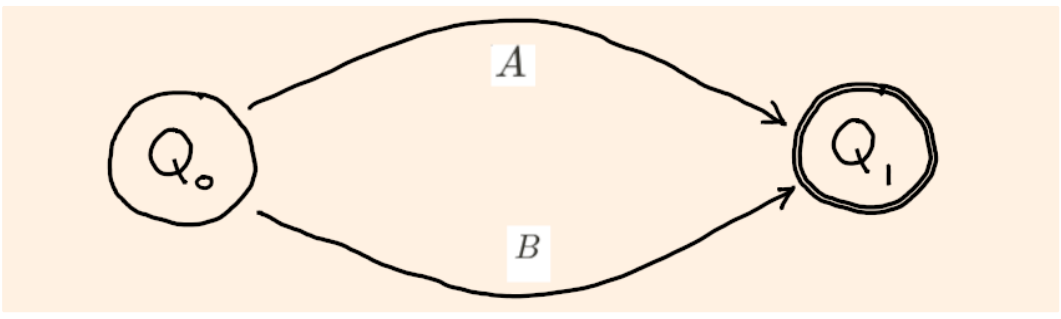
- Consider a start state (say Q_0) and a final state (say Q_1). We can consider the entire regex to be a transiton between Q_0 and Q_1, as shown below:



- Now, we recurse inside each layer of parenthesis to perform corresponding operation. In the given case, let $A = (((a+b)*)+(b.a))$ and $B = (c*)$. Thus, the outermost bracket is represented as the transition $(A+B)$:



Thus, the transition can be handled as:



A similar approach can be used to for each of the further layers of parenthesis and each of the regex operations, as well.

- The intermediate (compound) transitions are stored in an intermediate list (`interTransitions`) and the only the innermost transition (corresponding to transition due to input) is stored in the transition matrix. This is handled by the `makeTransition` function:

```
def makeTransition(start, transition, end): if isinstance(transition, list):
interTransitions.append([start, transition, end]) else: transitions.append([start, transition, end])
```

- The algorithm terminates when there are no more intermediate transitions in the `interTransitions` list.

Question 2: NFA → DFA

Algorithm:

- The states in DFA will be the elements of the powerset of the states in NFA:

$$\text{states} = \mathbb{P}(\ \text{NFA}[\text{“states”}]\ )$$

- The start states of DFA will be same as the start states of the NFA, while the end states of DFA will be any state which has any of the end state of NFA in it.

```
for i in NFA["final_states"]: for j in states: if (i in j) and (j not in final_states):
final_states.append(j)
```

- Transition of a state in DFA will be the union of the transition of each of the states within it in the NFA.

$$\delta'([\ q_0,\ q_1,\ \cdots\ ]) = \delta(q_0)\ \cup\ \delta(q_1)\ \cup\ \cdots$$

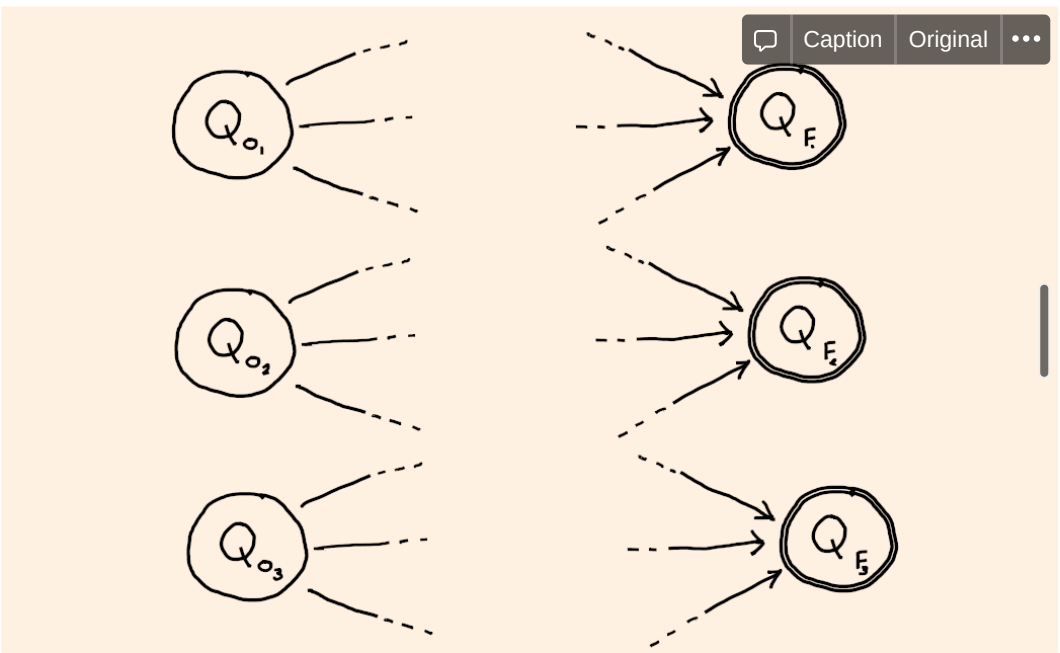
Assumption taken:

We have assumed that the given NFA has no ϵ transitions.

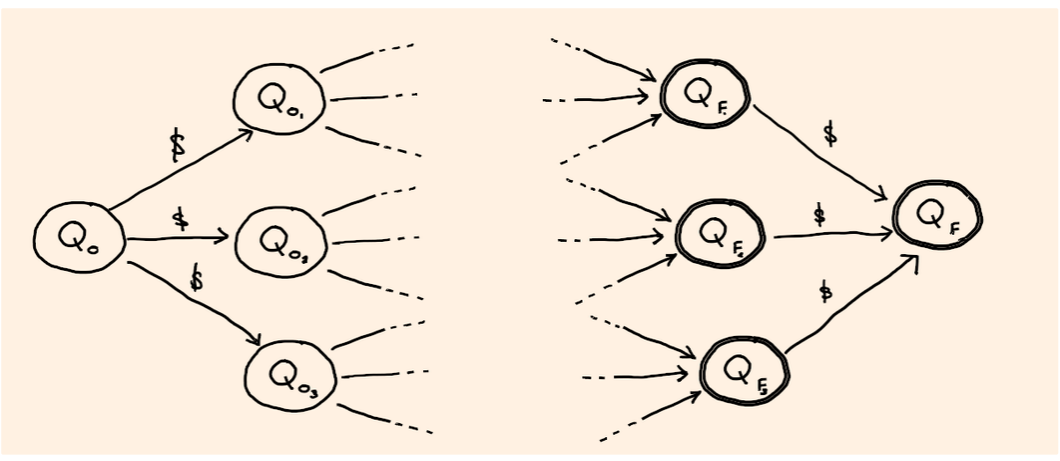
Question 3: DFA → Regular Expression

Algorithm:

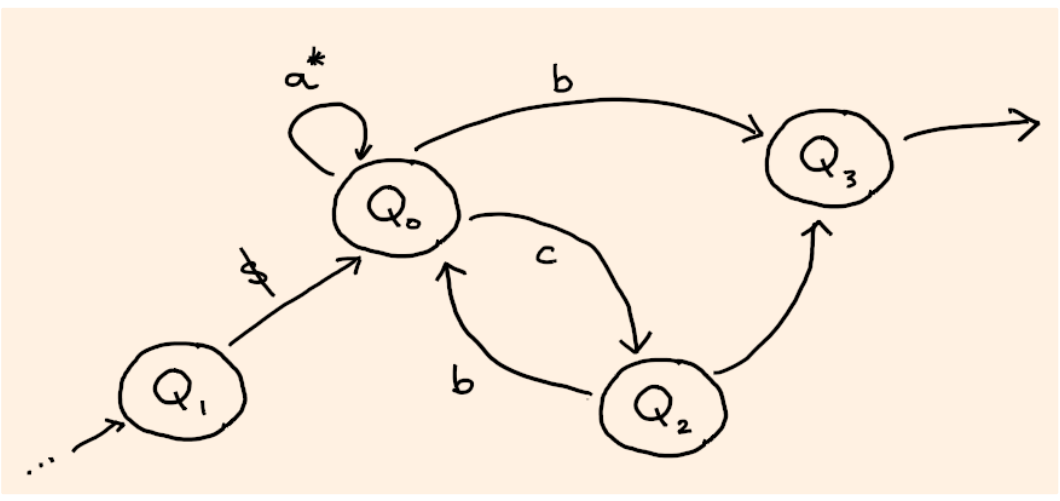
- Consider the DFA given:



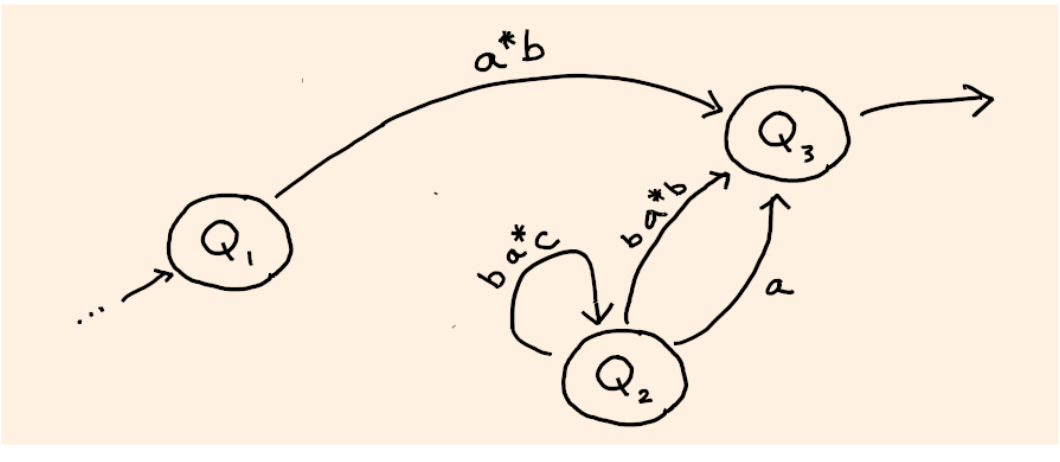
Generate a GNFA from the DFA by adding creating a new start state with epsilon transitions from it leading to the start states of DFA. Similarly, create a new end state with epsilon transtions from the end states to it.



- Now, proceed to remove each state, replacing each state with corresponding regex operation. For example:



After removing state Q_0, we will get:



- Continue till only the final and end state remain, and have a single transition between them. This transiton will be the required regex.

Question 4: DFA → Minimise DFA

Algorithm:

- Partition the DFA states into Accepting and Non-Accepting states.
- Check whether the states in each partion is distinguishable. Distinguishablity, here, is defined as whether or not two states have similar transitions. If two states do not have similar transition with respected to the partions, they are said to be distinguishable.
- Divide the each of the partitions into further partition (groups) on the basis of distinguishablity. Continue the process till all partitions have indistinguishable states.
- Group the states to get new states for the minimised DFA. The transition can also be obtained in a similar manner.