CS 208 : Automata Theory and Logic 12^{th} February, 2024

Lecture - 14

Topic: DFA & NDFA

Scribed by: Ojas Maheshwari (22b0965)
Checked and compiled by: Anubhav Jana

1 Quiz Question Insights

Question:
Draw a Deterministic Finite Automaton (DFA) for $L := \{w \in \{a,b\}^* : 2 \text{ divides } n_a(w) \text{ and } 3 \text{ divides } n_b(w)\}$. Here $n_a(w)$ stands for the number of a's in w, and $n_b(w)$ stands for the number of b's in w. For example, $n_a(abbaab) = n_b(abbaab) = 3$. Therefore, $ababbaa \in L$ but $aabababa \notin L$.

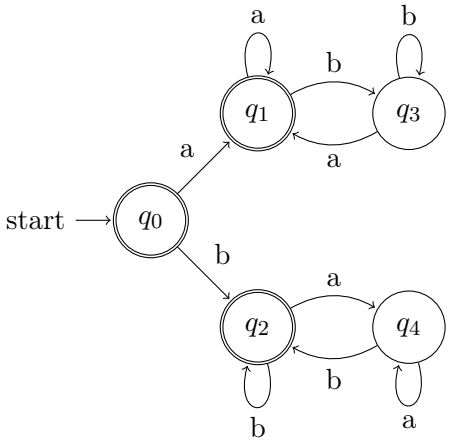
In certain scenarios, expressing a language solely through propositional logic becomes impractical, particularly when the length of the strings is unknown or variable. For instance, consider above example. In this case, the length n of the string is not explicitly provided, making it challenging to construct a propositional logic expression directly. Propositional logic typically operates on fixed, predetermined conditions or patterns within strings, which cannot accommodate variable lengths. However, deterministic finite automata (DFAs) offer a suitable alternative for such situations. DFAs are well-suited for languages where the structure and properties depend on the characters within the string rather than on fixed string lengths. By employing states and transitions based on input characters, DFAs can effectively recognize languages with variable-length strings and complex patterns, making them a more appropriate choice when string length is not predetermined.

2 Example:

Consider $L := \{w \in \{a,b\}^* : n_{ab}(w) = n_{ba}(w)\}$. Here $n_{ab}(w)$ stands for the number of "ab"s in w, and $n_{ab}(w)$ stands for the number of "ba"s in w. For example, $abaabbaa \in L$ as $n_{ba} = n_{ab} = 2$ but $abaab \notin L$ as $n_{ba} = 1$ and $n_{ab} = 2$.
At first glance, the question seems to be unsolvable using "finite automata". But one must note that $|n_{ab} - n_{ba}| \leq 1$ always. Why?
By the configuration of string w, between any two occurrences of ab, there must be an occurrence of ba and similarly, between any two occurrences of ba, there must be an occurrence of ab. Hence the difference cannot exceed 1. Thus, we can draw DFA as follows:

State	Transition on 'a'	Transition on 'b'	Description of State
q_0	q_1	q_2	ϵ (null string) $\in L$
q_1	q_1	q_3	$n_{ab} - n_{ba} = 0$ and last alphabet is a
q_2	q_4	q_2	$n_{ab} - n_{ba} = 0$ and last alphabet is b
q_3	q_1	q_3	$n_{ab} - n_{ba} = 1$
q_4	q_4	q_2	$n_{ab} - n_{ba} = -1$

Note that the last letter in states q_3 and q_4 are compelled to be b and a respectively.



Another beautiful insight gained is the fact that this problem is equivalent to the problem where the first and last letters of w are the same. Now consider another example:
$L := \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$. Here $n_a(w)$ stands for the number of "a"s in w, and $n_b(w)$ stands for the number of "b"s in w.
This language cannot be represented by a finite automaton. The proof for this will be discussed shortly.
But for now, we will solely be focusing on the Languages which *can* be represented by a Finite Automaton.

3 Deterministic Finite Automaton (DFA)

1. **Finite Set of States**: A DFA has a finite set of states, denoted as Q.
2. **Alphabet**: It operates on a finite alphabet, denoted as Σ, which consists of input symbols.
3. **Transition Function**: There exists a transition function $\delta : Q \times \Sigma \to Q$, which maps each state and input symbol pair to a unique state.
4. **Initial State**: There is a designated unique initial state $q_0 \in Q$ from which the computation starts.
5. **Accepting States (Final States)**: Some states in Q are designated as accepting states, denoted as F. If the DFA halts in an accepting state after processing the input string, then the input string is accepted.
6. **Deterministic Behavior**: For every state q in Q and every symbol a in Σ, there is exactly one transition defined.
7. **Language Recognition**: A DFA recognizes a language L, which consists of all strings over Σ that, when input into the DFA, result in the DFA halting in an accepting state.
8. **Uniqueness of Computation**: For any input string, there is a unique computation path in the DFA, leading to a unique final state.

4 Non-Deterministic Finite Automaton (NDFA)

1. **Finite Set of States**: An NDFA has a finite set of states, denoted as Q.
2. **Alphabet**: It operates on a finite alphabet, denoted as Σ, which consists of input symbols.

3. **Transition Function**: There exists a transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$, which maps each state and input symbol to a set of possible states.

4. **Initial State(s)**: There is a designated set of initial states (may be multiple) $q_0 \in Q$ from which the computation starts.

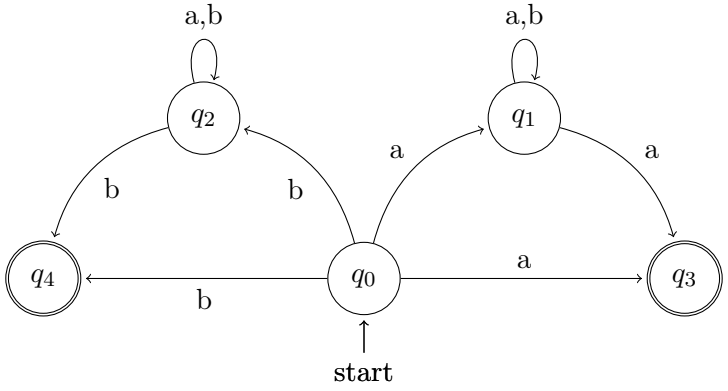
5. **Non-Deterministic Behavior**: For every state q in Q and every symbol a in Σ , there may be multiple transitions defined.

6. **Accepting States (Final States)**: Some states in Q are designated as accepting states, denoted as F. If atleast one computation path in the NDFA halts in an accepting state after processing the entire input string, then the input string is accepted.

Consider the previous example: $L := \{w \in \{a, b\}^* : n_{ab}(w) = n_{ba}(w)\}$. It can be represented by an NDFA as follows:

State	Transition on 'a'	Transition on 'b'
q_0	q_1, q_3	q_2, q_4
q_1	q_1, q_3	q_1
q_2	q_2	q_2, q_4
q_3	–	–
q_4	–	–

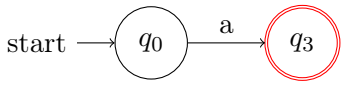
Here, "-" signifies that the transition is undefined.



Let us analyze this NDFA for the string **ababa**.
Things to note:

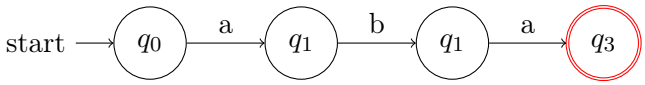
- If we reach states q_3 or q_4, the string will be accepted only if there is no more input left. If there is an input left, the state transition has to be rejected as the states q_3 and q_4 do not allow any input to be taken.
- There is no unique transition since the transition function is not one-one.
- If there is even a single transition that results in a halt at the accepting states, the string is accepted.

There are precisely **4** transitions possible:
Transition 1:



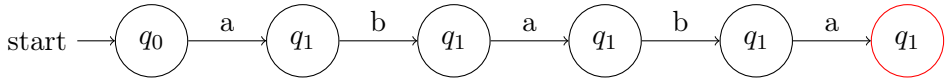
This is rejected as it does not allow the next input b to be taken even though it is an accepting state.

Transition 2:



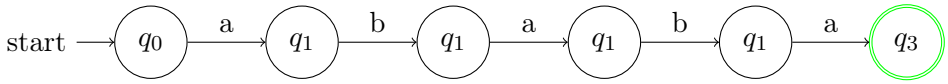
This is rejected as it does not allow the next input b to be taken even though it is an accepting state.

Transition 3:



This is rejected as q_1 is not an accepting state.

Transition 4:

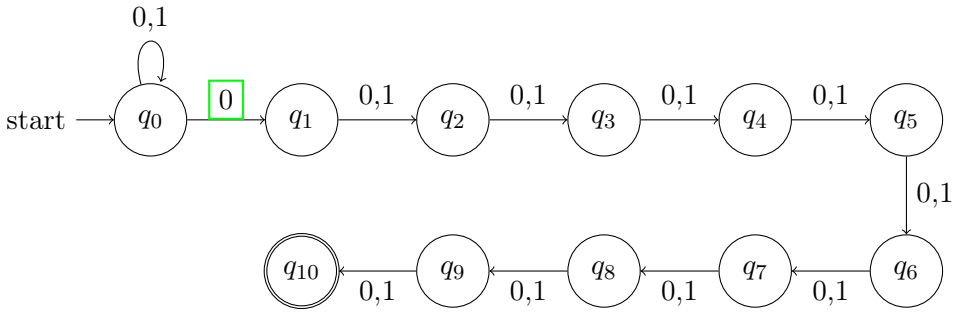


Thus the string *ababa* is accepted as we have found a computation path that halts at an accepting state.

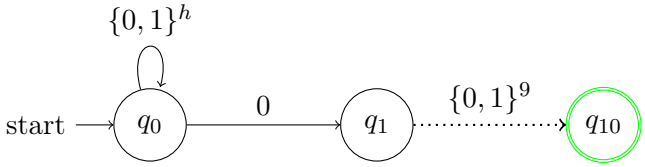
Clearly, the computational process outlined here is laborious, requiring scrutiny of all potential transitions to determine whether a string belongs to the language or not. Given this complexity, one might question the utility of Non-Deterministic Finite Automata (NDFA) over Deterministic Finite Automata (DFA).

NDFA, despite their apparent intricacy, offer advantages in certain scenarios. They provide a more expressive power by allowing multiple transitions from a single state on the same input symbol, facilitating more concise representations of certain languages. Additionally, NDFA can be easier to design for some languages that are inherently non-deterministic or where the determinization process of DFA might result in an impractically large automaton.

Consider the example: $L := \{w \in \{0,1\}^* : |w| \geq 10,\ 10^{th}$ last alphabet is $0\}$
For representing the above as a DFA, it will take us $2^{10} = 1024$ states to store all possible bit configurations of the last 10 letters. Hence, it is inefficient space-wise. But consider the below NDFA model of the same problem:



Consider any string of the format $\{0,1\}^h 0 \{0,1\}^9$ (where h is a finite number). This string must be accepted by our NDFA. We need to find a transition for which the is accepted i.e. the last bit halts at the state q_{10}. Consider the the following transition:
First h bits result in a transition from q_0 to itself. Then the **0** results in transition to q_1 and then the next 9 bits transition it to q_{10} and the NDFA halts at the accepted state. i.e.



The problem can be depicted as an NDFA using only 11 states. Hence, it is very efficient space-wise. Thus, NDFA can sometimes lead to more efficient solutions, particularly in cases where the language's structure aligns well with the non-deterministic nature of the automaton. This can result in fewer states and transitions compared to their deterministic counterparts, leading to reduced computational complexity.
In summary, while NDFA may introduce complexity in the analysis of individual strings, they offer advantages in terms of expressiveness, ease of design for certain languages, and potentially more efficient solutions for specific problem domains.