Department of Computer Science
Ashoka University

Discrete Mathematics: CS-1104-1 & CS-1104-2

Assignment 6

Collaborators: None **Name: Rushil Gupta**

---

1 Straightforward

1. (a) i. abc
   - G1: $S \to aS \to abA \to abcA \to abc$
   - G2: $S \to A \to aA \to aB \to abC \to abcC \to abc$

   ii. aabccc
   - G1: $S \to aS \to aaS \to aabA \to aabcA \to aabccA \to aabcccA \to aabccc$
   - G2: $S \to A \to aA \to aaA \to aabC \to aabcC \to aabccC \to aabcccC \to aabccc$

   iii. aacc
   - G1: not possible, since we cannot transition from S to A without a b
   - G2: not possible, since we cannot transition from B to C without a b

   iv. abbc
   - G1: not possible, since we cannot have two consequent b's (the only way to get a b is by transitioning from S to bA, and after state A, we cannot add more b's)
   - G2: not possible, since we cannot have two consequent b's (the only way to get a b is by transitioning from B to bC, and after state C, we cannot add more b's)

   (b) G1 is regular because it is a right-linear grammar. Each production in G1 has a non-terminal on the left hand side and a terminal or a terminal followed by a non-terminal on the right hand side.

   G2 is not regular because it is not a right-linear grammar. The production $S \to A$ is not a right-linear production because it has a non-terminal on the left hand side and a non-terminal on the right hand side.

   (c) Consider the general form of the strings generated by G1:
   $S \to aS \to aaS \to aa \dots aS \to a^n bA \to a^n bcA \to a^n bcc...cA \to a^n bc^m$

   Consider the general form of the strings generated by G2:
   $S \to A \to aA \to aaA \to aa \dots aA \to a^n bC \to a^n bcC \to a^n bcc...cC \to a^n bc^m$

   Since the strings generated by G1 and G2 are the same, $L(G1) = L(G2)$.

   (d) Yes, $L(G2)$ is a regular language because it is recognized by a right-linear grammar.

2. (a) Proof using counterexample:

Assume that $L(G_l) = L(G_r)$. Trying to generate the string aab using G_r and G_l:

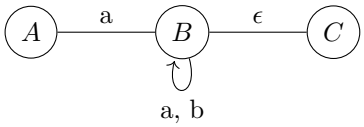
- G_r: $A \to aB \to aaB \to aabB \to aab$
- G_l: $A \to Ba \to Baa \to Baa \to \cdots$ (It's important to note that we cannot generate a string ending with b using G_l)

So, $L(G_l) \neq L(G_r)$.

(b) To convert a right-linear grammar $G_r =< T, N, S, P >$ to an equivalent left-linear grammar $G_l =< T, N, S, P' >$, we can follow the steps below:

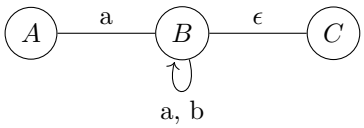
- Draw a finite automaton for the right-linear grammar using the production rules given.
- Reverse the initial and final states of the finite automaton.
- Reverse the edges direction in the finite automaton.
- Construct the left-linear grammar by using the production rules of the finite automaton.
- The left-linear grammar is now equivalent to the right-linear grammar.

(c) Consider the right-linear grammar $G_r =< T, N, S, P >$ where $T = \{a, b\}$, $N = \{A, B\}$, $S = A$, and $P = \{A \to aB, B \to aB|bB|\epsilon\}$. Here is the FA representing the grammar:



Note that C is the final/accept state.

Now, we reverse the initial and final states of the FA and reverse the edges direction. The new FA is as follows:



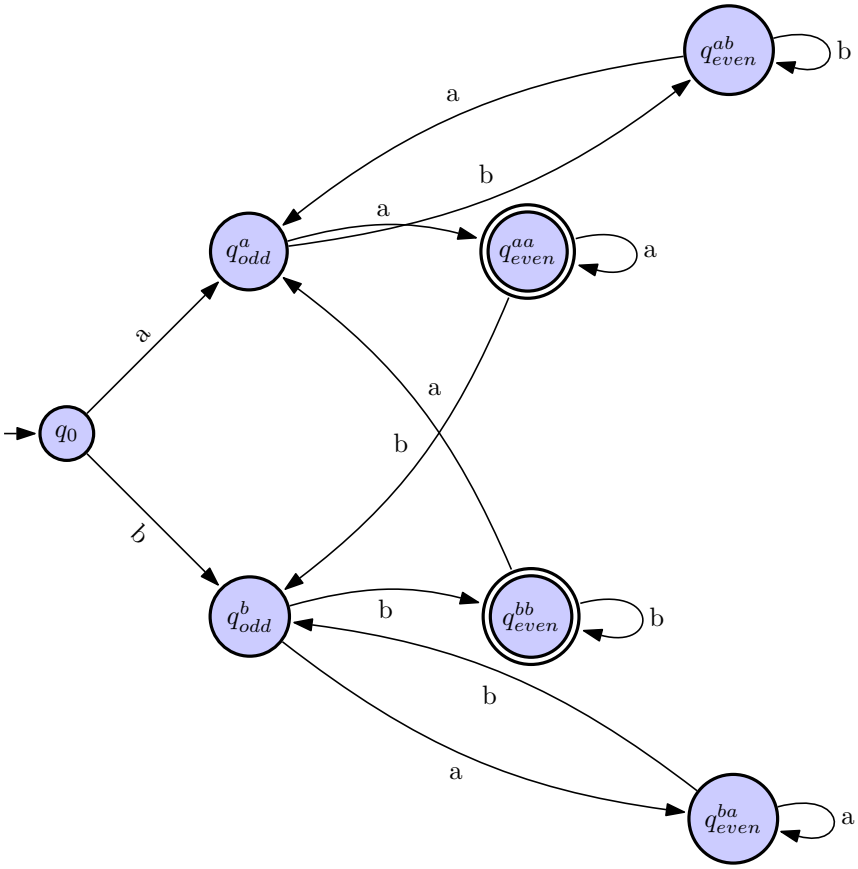
Here, A is the final/accept state.

Here we see that the FA is not deterministic. However, upon extracting the grammar from the FA, we get the left-linear grammar $G_l =< T, N, S, P' >$ where $T = \{a, b\}$, $N = \{A, B\}$, $S = A$, and $P' = \{A \to Ba, B \to Ba|Bb|\epsilon\}$.

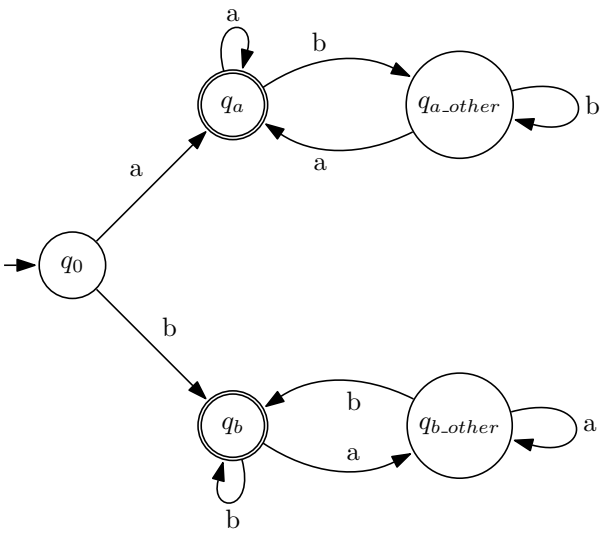
(d) To prove the correctness of the process, we need to show that $L(G_l) = L(G_r)$.
Clearly, from the FA, we can see that the strings generated by G_l are the same as the strings generated by G_r.

So, $L(G_l) = L(G_r)$.

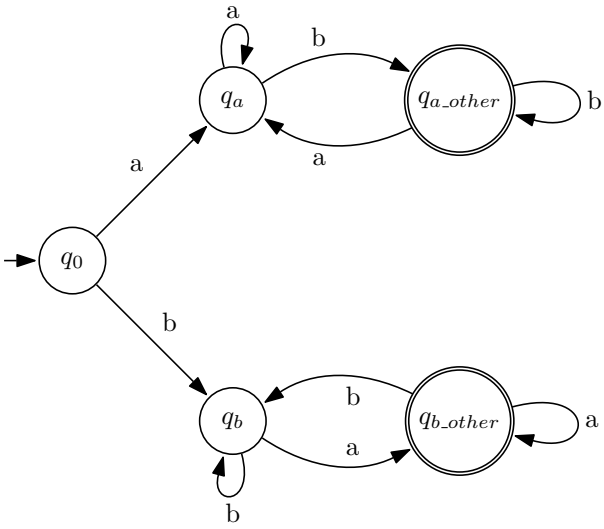
3. (a) The DFA M for L is as follows:



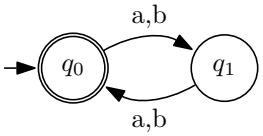
(b) The DFA M_s for L_s is as follows:



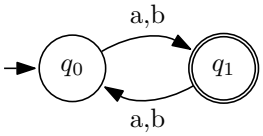
The DFA M_d for L_d is as follows:



The DFA M_e for L_e is as follows:



The DFA M_o for L_o is as follows:



(c) To express the language L in terms of L_s, L_d, L_e, and L_o, we should first understand how L is defined and then see how it can be represented using the intersection and union of these four languages. The language L is defined as:

$$L = \{w = w_1 w_2 \dots w_n \mid (w_1 = w_n \wedge |w| \equiv 0 \pmod 2) \vee (w_1 \neq w_n \wedge |w| \equiv 1 \pmod 2), n > 0\}$$

Breaking Down L: This definition can be split into two parts:

i. Strings where the first and last characters are the same, and the length of the string is even.

ii. Strings where the first and last characters are different, and the length of the string is odd.

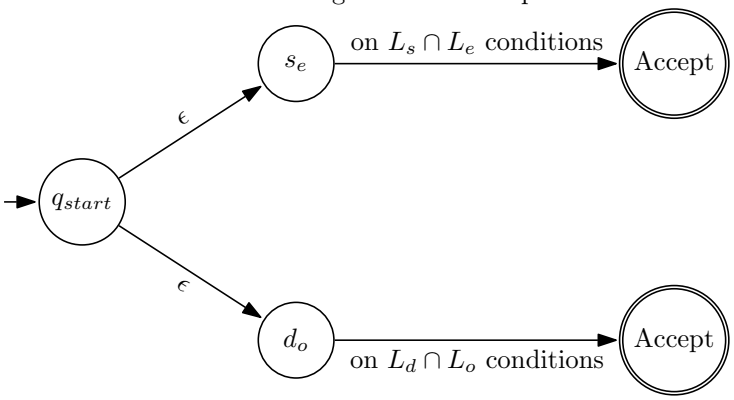
Given the languages:

- $L_s = \{w \mid w_1 = w_n, n > 0\}$ — Strings that start and end with the same character.
- $L_d = \{w \mid w_1 \neq w_n, n > 0\}$ — Strings that start and end with different characters.
- $L_e = \{w \mid |w| \equiv 0 \pmod 2, n > 0\}$ — Strings of even length.
- $L_o = \{w \mid |w| \equiv 1 \pmod 2, n > 0\}$ — Strings of odd length.

Expression Using Set Operations: We can now use set operations (intersection and union) to express L:

$$L = (L_s \cap L_e) \cup (L_d \cap L_o)$$

(d) Here is the NFA N for L using the relationship we found earlier:



(e) To prove that $L(M) = L(N) = L$, where M is the DFA for the language L and N is the NFA constructed to represent L, we need to first describe the conditions defined by L and then show how both automata satisfy these conditions.

Language L Description:

$$L = \{w = w_1 w_2 \dots w_n \mid (w_1 = w_n \wedge |w| \equiv 0 \pmod 2) \vee (w_1 \neq w_n \wedge |w| \equiv 1 \pmod 2), n > 0\}$$

This language consists of strings where:

i. The first and last characters are the same, and the length of the string is even.
ii. The first and last characters are different, and the length of the string is odd.

Breaking Down into Sublanguages: We defined the sublanguages as:

- $L_s = \{w \mid w_1 = w_n, n > 0\}$
- $L_d = \{w \mid w_1 \neq w_n, n > 0\}$
- $L_e = \{w \mid |w| \equiv 0 \pmod 2, n > 0\}$
- $L_o = \{w \mid |w| \equiv 1 \pmod 2, n > 0\}$

We then express L as:

$$L = (L_s \cap L_e) \cup (L_d \cap L_o)$$

Proof for M (DFA):

- We designed M with the understanding that it needs to recognize strings that fit one of the two conditions:
  i. First and last characters the same, even length.
  ii. First and last characters different, odd length.

- The DFA M was implicitly defined by combining the rules of L_s and L_e or L_d and L_o through specific states that track:
  i. The start character to check if it matches the end character.
  ii. The length of the string to determine if it is odd or even.

- Transitions in M ensure that if a string starts with a character, say 'a', it keeps track of this 'a' and whether the subsequent length is odd or even, thereby determining which part of the language it satisfies.

Proof for N (NFA):

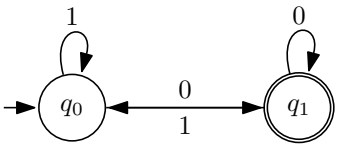
- N was constructed using ϵ-transitions to start either the simulation of $(L_s \cap L_e)$ or $(L_d \cap L_o)$.
- The paths $(L_s \cap L_e)$ and $(L_d \cap L_o)$ in N are independent and explicitly model the criteria set by these combinations.
- By using ϵ-transitions, N non-deterministically chooses to evaluate a string under one of the two required conditions, thus covering all elements of L.

Equality:

- $L(M)$ captures all strings as per the definition of L since the DFA M is constructed to switch states based on input while maintaining checks on start-end character equality/inequality and string length parity.
- $L(N)$ represents the same language L by nondeterministically deciding which combination of sublanguages a given string belongs to, checking the conditions simultaneously in separate branches of the computation. Thus, both M and N correctly implement the logic required to accept exactly the strings defined in L, hence $L(M) = L(N) = L$.

4. 1) L_I:

   (a) The NFA N for L_I is as follows:



   (b) The right linear grammar G for L_I is as follows:

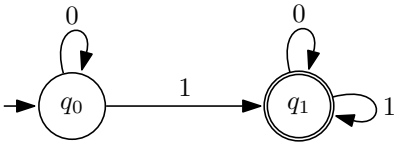
   - Variables: S, A
   - Productions: $S \to 0A \mid 1A$ and $A \to 0S \mid \epsilon$

   (c) To prove $L = L(N) = L(G)$, we need to show that the strings generated by N and G are the same as the strings generated by L.

   - $L(N)$: The NFA N accepts all strings where every character at an even index is a '0'.
   - $L(G)$: The right linear grammar G generates strings where every character at an even index is a '0'.

2) L_{II}:

   (a) The NFA N for L_{II} is as follows:



   (b) The right linear grammar G for L_{II} is as follows:

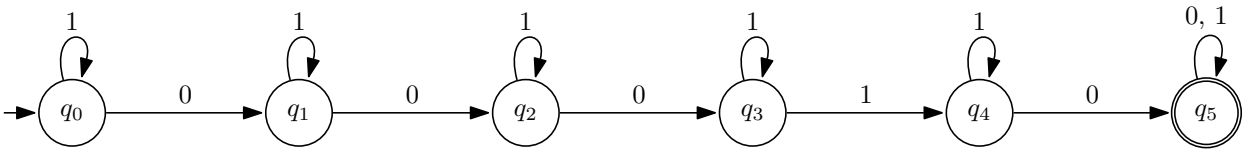
   - Variables: S, A
   - Productions: $S \to 0S \mid 1A$ and $A \to 0A \mid \epsilon$

   (c) To prove $L = L(N) = L(G)$, we need to show that the strings generated by N and G are the same as the strings generated by L.

   - $L(N)$: The NFA N accepts all strings where there is exactly one '1'.
   - $L(G)$: The right linear grammar G generates strings where there is exactly one '1'.

3) L_{III}:

(a) The NFA N for L_{III} is as follows:



(b) The right linear grammar G for L_{III} is as follows:

- Variables: S, A, B
- Productions: $S \to 1S \mid 0A \mid \epsilon$, $A \to 1S \mid 0B$, and $B \to 1S \mid \epsilon$

(c) To prove $L = L(N) = L(G)$, we need to show that the strings generated by N and G are the same as the strings generated by L.

- $L(N)$: The NFA N accepts all strings where the substring "000" does not appear.
- $L(G)$: The right linear grammar G generates strings where the substring "000" does not appear.

2 ¬Straightforward

1. a

3 Bonus

(a) Below is a table format describing the transitions for each state based on the corrected understanding of the diagram:

State	Reads 'a'	Reads 'b'
q_0	Goes to q_2	Loops on q_0
q_1	Goes to q_0	Loops on q_1
q_2	Goes to q_3	Loops on q_2
q_3	Goes to q_1	Goes to q_2

The table describes the transitions for each state based on the input read.

(b) The regular grammar G for the state machine M is as follows:

- Variables: S_0, S_1, S_2, S_3
- Productions: $S_0 \to aS_2 \mid bS_0$, $S_1 \to aS_0 \mid bS_1$, $S_2 \to aS_3 \mid bS_2$, and $S_3 \to aS_1 \mid bS_2$

(c) To prove that $L(G) = L(M)$, we need to establish two things:

(a) $L(G) \subseteq L(M)$: Every string generated by grammar G is recognized by automaton M.
(b) $L(M) \subseteq L(G)$: Every string recognized by automaton M can be generated by grammar G.

Proving $L(G) \subseteq L(M)$:
The production rules of G correspond to the transitions in M. Each rule in G is derived directly from the transitions of M. If a string w can be generated from a non-terminal S_i (corresponding to state q_i in M), then there exists a sequence of transitions in M starting from q_i that results in w being accepted if q_i leads to an accepting state after processing w. Since the grammar's production rules mimic the state transitions, each production $S_i \to aS_j \mid bS_k$ represents a valid transition q_i on 'a' to q_j and on 'b' to q_k in M. Thus, if w is generated by G, starting from the start symbol corresponding to the initial state of M, then following the transitions dictated by w in M must lead to an accepting state if w ends at a production that includes ϵ or at a non-terminal that corresponds to an accepting state in M.

Proving $L(M) \subseteq L(G)$:
Consider any string w that is accepted by M. Starting from the initial state q_0 (corresponding to the start symbol S_0), for every transition q_i to q_j on symbol x in M, there is a corresponding production $S_i \to xS_j$ in G. By following the path that M takes on w, we can construct a derivation in G that generates w, starting from S_0 and proceeding according to the symbols in w. If w ends in an accepting state in M, the corresponding sequence of productions in G will either end in a non-terminal that generates ϵ (if that state is considered accepting in G) or completes the generation of w without needing ϵ.

Conclusion:
Because the transitions and productions match exactly, and the criteria for acceptance are mirrored between M and G, it follows that every string in $L(M)$ can be generated by G and vice versa. Hence, $L(G) = L(M)$. This equivalence shows that the grammar G correctly models the automaton M, both structurally and in terms of language recognition.