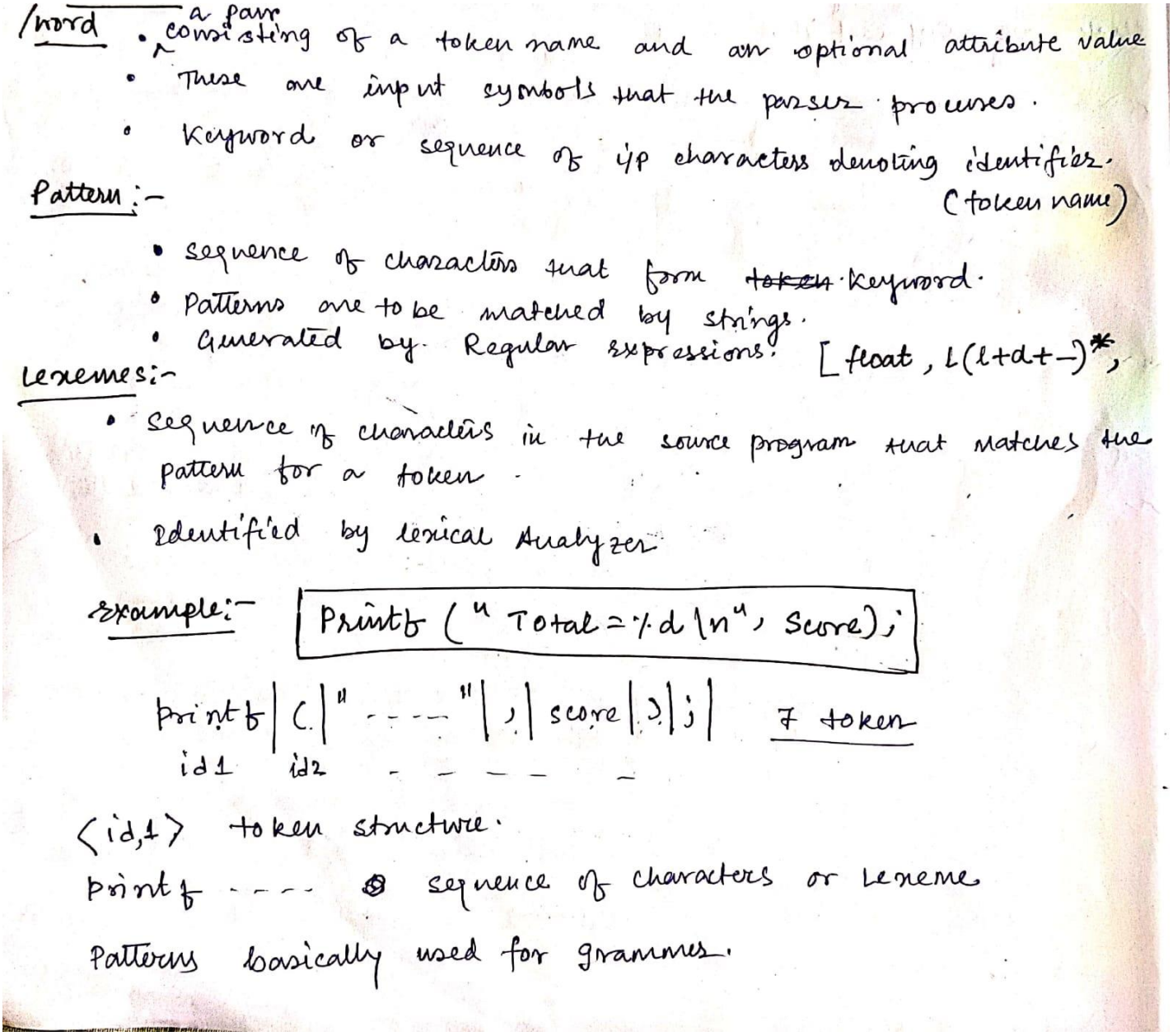
LEXICAL ANALYSIS



Role of Lexical Analyzer:

- Read the input characters of the source program.
- Group them into lexemes.
- Produces a sequence of tokens for each lexeme in the source program.
- LA is basically used to identify tokens.
- DFAs are used during lexical analysis to generate transition diagram for identification of tokens
- Keep track of newline characters, so that it can associate line number with each error messages.
- Performs the expansions for pre-processors.

(*) Input Buffering:

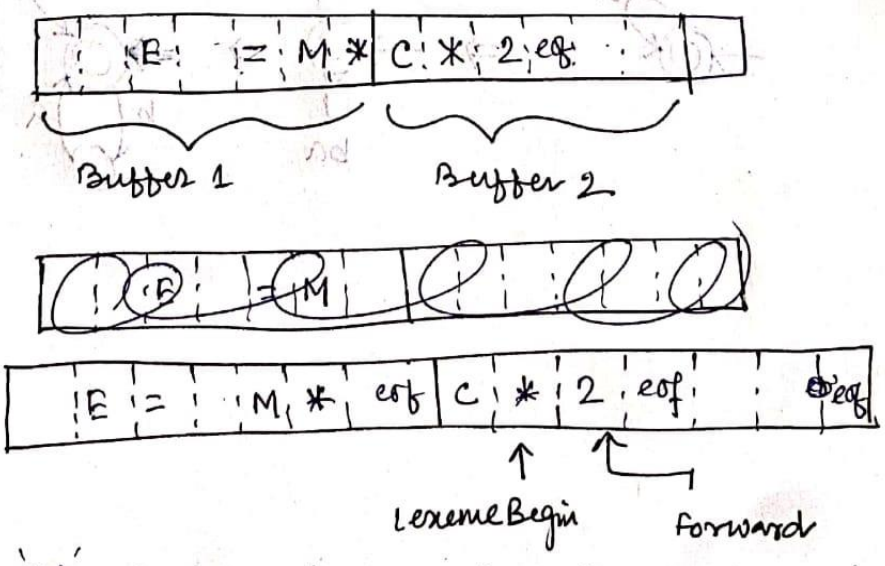
We often have to look one or more characters beyond the next lexeme before we can be sure we have the right lexeme. That need to look at least one additional character ahead. So, we need a two-buffer scheme to lookahead large.

(1) Buffer Pairs :-

- 2 buffers alternately reloaded
- Buffer size = disk block size = N.
- we can read N - characters into a buffer.
- If no. of characters are fewer than N, then 'eof' used.
- 2 pointers are used;
  - lexemeBegin (beginning of the current lexeme)
  - forward (scans ahead until a pattern match found)

(2) Sentinels :-

- A special character that cannot be the part of the source program; that is 'eof'.
- We can include a sentinel character at the end of each buffer to test the buffer end. (saves time).



sentinels 'eof' at the end of each buffer and end of file also.

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

Formal Definition of FA

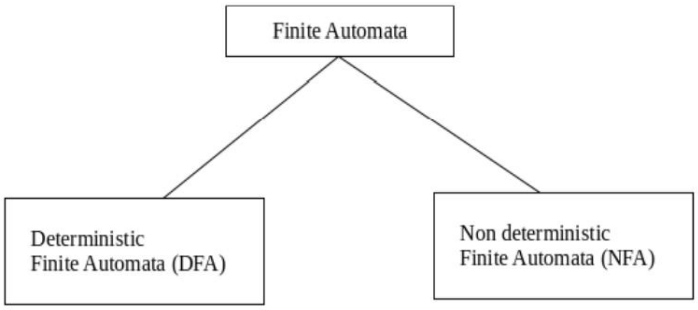
A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

1. Q : finite set of states
2. ∑ : finite set of the input symbol
3. q0 : initial state
4. F : final state
5. δ :Transition function

Types of Automata:

There are two types of finite automata:

1. DFA(deterministic finite automata)
2. NFA(non-deterministic finite automata)



❖ Deterministic Finite Automaton (DFA)

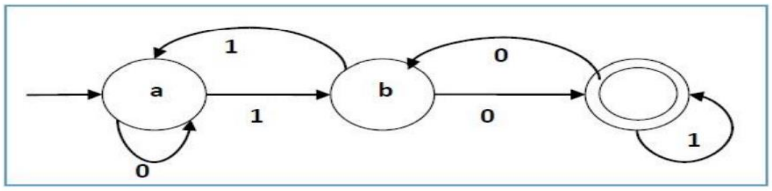
In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.** DFA does not accept the null move.

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × ∑ → Q
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Its graphical representation would be as follows −



❖ Non-deterministic Finite Automaton

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton**. As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**. It can accept the null move.

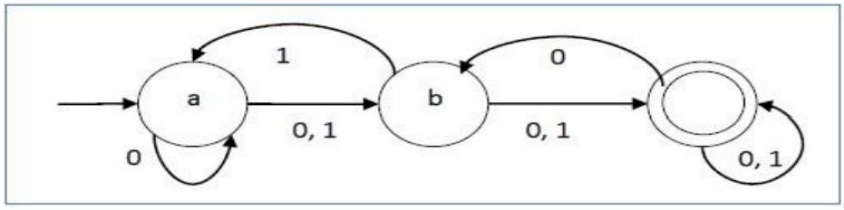
Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where −

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the alphabets.
- **δ** is the transition function where $\delta: Q \times \sum \rightarrow 2^Q$

  (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Its graphical representation would be as follows −



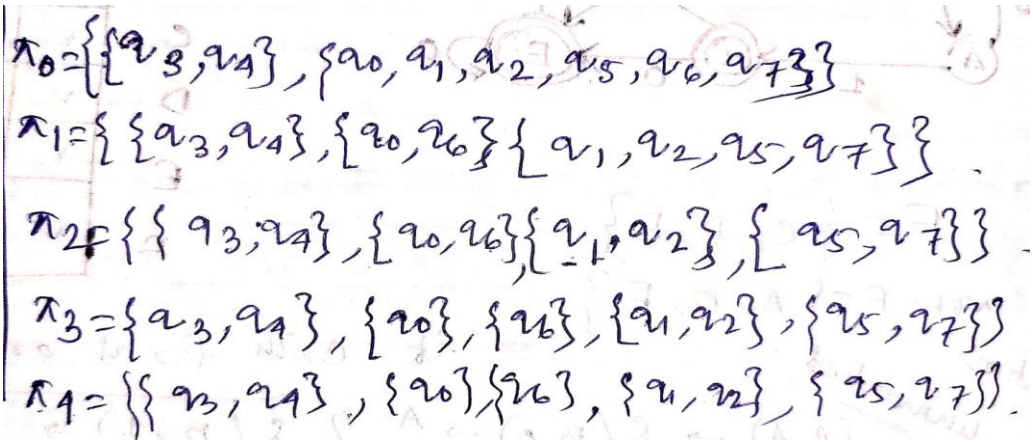
DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.

SOME MORE IMPORTANT TOPICS ON TOC

1. Minimize the states of FA

	a	b
Q0	Q1	Q2
Q1	Q4	Q3
Q2	Q4	Q3
Q3	Q5	Q6
Q4	Q7	Q6
Q5	Q3	Q6
Q6	Q6	Q6
Q7	Q4	Q6

Solution:



2. Construct the following DFA

i) Set of all strings over {a,b} in which no of a's and no of b's both are even.

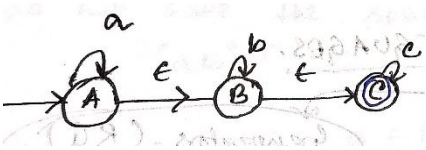
ii) Set of all strings over {a,b} in which no of a's are divisible by 3 and no of b's divisible by 2.

iv) DFA starts with 'a' and ends with 'b'.

v) DFA starts and ends with same symbol.

vi) L = { a^n b^m c^l | n,m,l>=0}

3. Convert the ϵ-NFA to NFA (Finding ϵ-closure property)



4. Convert NFA to DFA (Method of subset constructions)

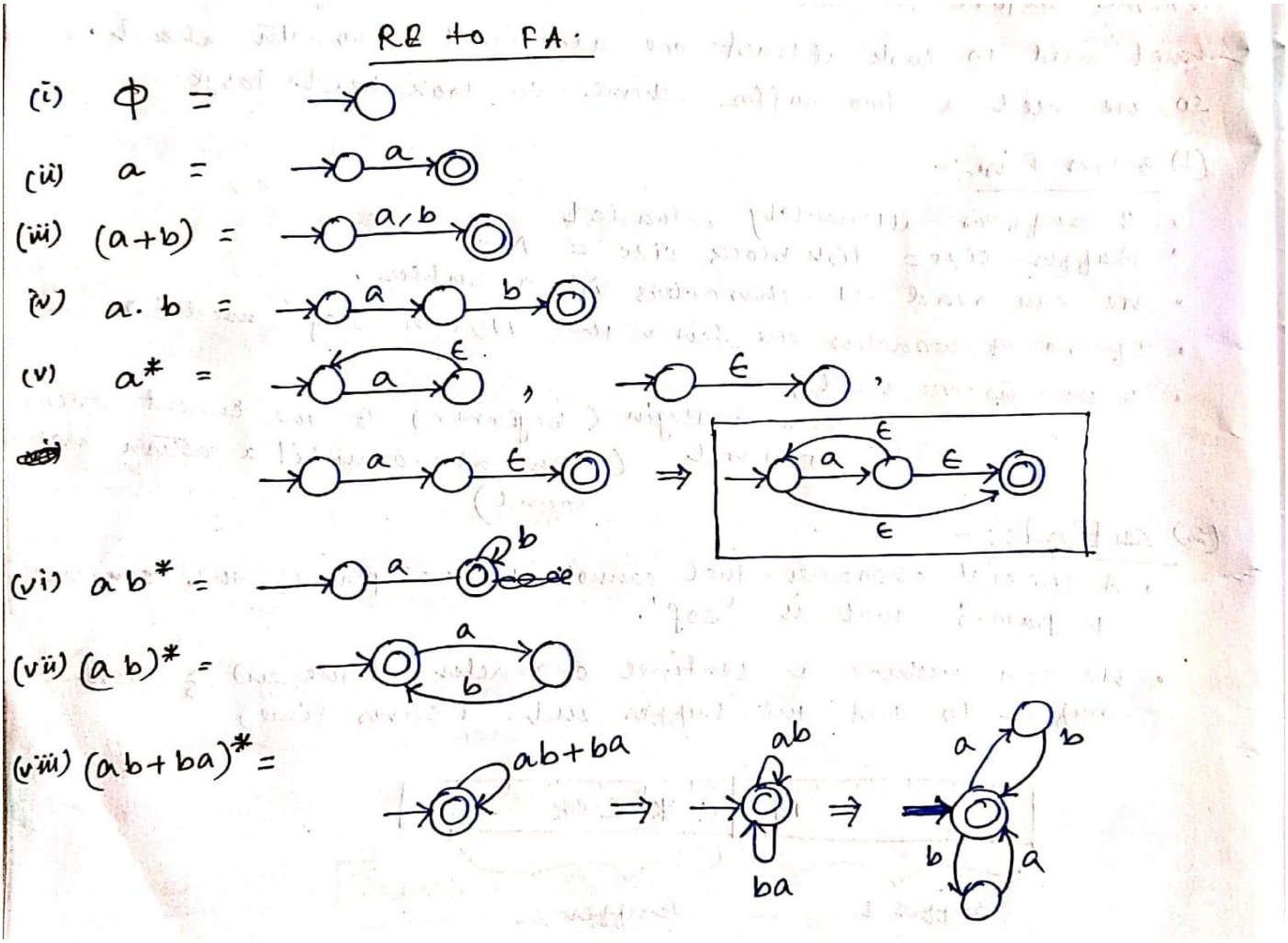
Construct the NFA on {a,b} where 3rd symbol from the RHS is 'a'. Then convert it into corresponding DFA.

5. Find the Regular Expressions:

i) Set of all strings of length at least 2

ii) Starting or ending with different with different symbol.

6. Converting Regular Expression to FA/NFA



(1) $[a + ba(a+b)]^* a(ba)^* b^*$