What is Computation?

It is a step by step solution to a problem [Algorithm]

eg: Multiplication of 2 numbers.

finding a word in a dictionary.

checking whether there is a path b/w two vertices in a graph.

Computational devices

- eg: Calculator, Cell phone, Computer, pen and paper

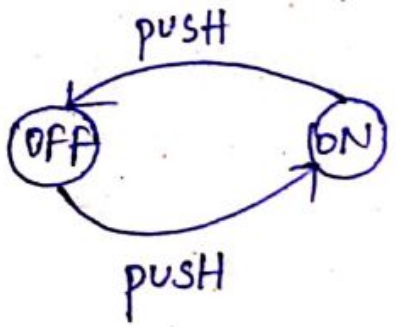
- Study computational devices based on the resources that use

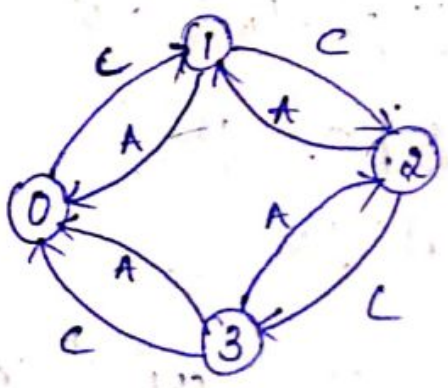
Finite Automata

It is a computational model which has finite amount of memory. It has states which are finite.

Automata is the plural of the word "Automaton"

eg: 1) An electric switch 2) Fan regulator





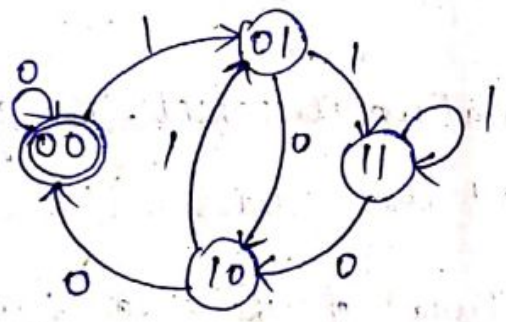
CACCA C

Scanned with CamScanner

3) $L = \{ x \mid x$ is a binary string divisible by $4 \}$

B	D	
100	4	✓
110	6	✗
1100	12	✓

observe that the set of binary nos divisible by 4 are exactly those that have 00 as a suffix.



00 : 1

01 : 1

start state 00

Accept state 00

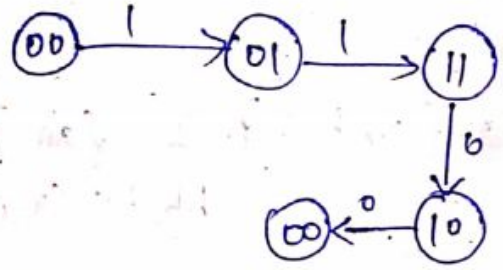
egs of strings

a) 110

This string is not accepted by Automata because we don't end up at accept state.

Scanned with CamScanner

b). 1100



It is accepted.

Definitions and Notations

- An alphabet is a finite set of symbols.

Denoted by Σ, Γ

$\Sigma = \{0, 1\}$, the binary alphabet.
$\Sigma = \{a, b, \ldots z\}$ set of all lower-case letters
set of all ASCII characters.

A string over an alphabet is a sequence of symbols from the alphabet.

Eg: $w = 01101$

Length of a string is the no: of symbols in the string.
Denoted as $|w|$

Eg: $|01101| = 5$

The empty string is the string consisting of 0 symbols

Denoted as 'ϵ'

Powers of an Alphabet : If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation.

$$\Sigma^i = \{ w \mid w \text{ is a string over } \Sigma \text{ and } |w| = i \}$$

if $\Sigma = \{0, 1\}$ then

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$

$\Sigma^+ = \bigcup_{i > 0} \Sigma^i$

we define Σ^i to be set of strings of length i, each of whose symbol is in Σ.

Scanned with CamScanner

A language L over an alphabet Σ, is a subset of Σ^*

If $\Sigma = \{a, b, c\}$ then $\Sigma^1 = \{a, b, c\}$ $\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$

Note that there is a slight confusion b/w Σ and Σ^1. The former is an alphabet with members a, b and c as symbols. The latter, however, is a set of strings with members as the strings a, b and c of length 1. We shall use the same notation for the two sets. Instead it may be inferred from the context whether we are speaking about an alphabet or a set of strings.

The set of all strings over an alphabet Σ is conventionally denoted Σ^*. for instance $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000 \ldots\}$

Put another way,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

Sometimes we wish to exclude the empty string from the set of strings. The set of nonempty strings from alphabet Σ is denoted Σ^+. Thus, two appropriate equivalences are:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Scanned with CamScanner

Finite Automata

- F.A with output
  - Moore Machine
  - Mealy Machine
- F.A without output
  - DFA
  - NFA
  - ε-NFA

DFA: Deterministic finite Automata

The term "deterministic" refers to the fact that on each input there is one and only one state to which the automaton can transition from its current state.

In contrast, "nondeterministic" finite automata can be in several states at once. ~~The~~

Definition of a Deterministic Finite Automaton:

A deterministic finite automaton consists of:

1) A finite set of states, often denoted Q
2) A finite set of input symbols, often denoted Σ.
3) A transition function that takes as argument a state and an input symbol and returns a state.
   The transition function will commonly be denoted δ.
   [~~In our informal graph representation of automata~~]
4) A start state, one of the states in Q
5) A set of final or accepting states F. The set F is a subset of Q.

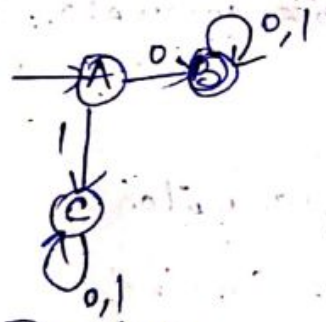
Scanned with CamScanner

* A deterministic finite automaton has a 5 tuple notation

$$A = (Q, \Sigma, \delta, q_0, F)$$

A → name of the DFA
Q → set of states
Σ → Its input symbols
δ → Its transition function
q_0 → start state
F → set of accepting states.

Eg: L_1 = set of all strings that start with '0'.
{0, 00, 01, 000, 010, 011, 0000 ...}



Dead state
or trap state.

Construct a DFA that accepts set of All strings over {0, 1} of length 2.

$\Sigma = \{0, 1\}$
$L = \{00, 01, 10, 11\}$

Scanned with CamScanner

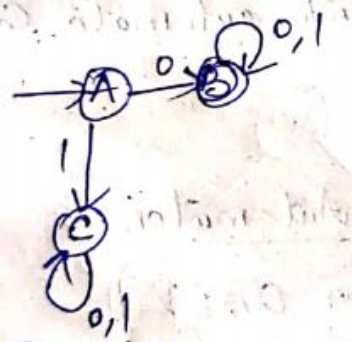
A deterministic finite automaton has a 5 tuple notation

$$A = (Q, \Sigma, \delta, q_0, F)$$

A → name of the DFA
Q → set of states
Σ → Its input symbols
δ → Its transition function
q_0 → start state
F → set of accepting states.

Eg: L_1 = set of all strings that start with '0'.

{0, 00, 01, 000, 010, 011, 0000 ...}



Dead state
or trap state.

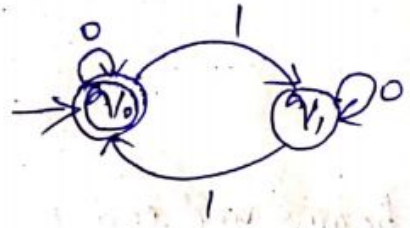
Construct a DFA that accepts set of All strings over {0, 1} of length 2.

$\Sigma = \{0, 1\}$

$L = \{00, 01, 10, 11\}$

Scanned with CamScanner

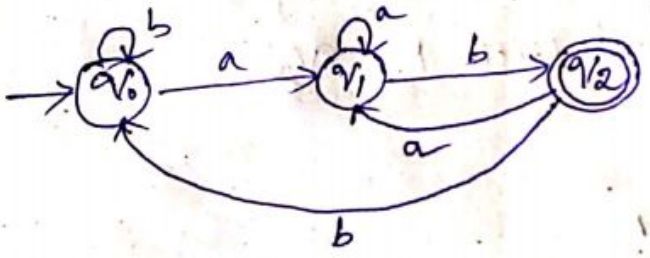
① $L_1 = \{ w \in \{0,1\}^* \mid w \text{ has an even no: of } 1\text{'s} \}$



q_0 corresponds to all strings having even no: of 1's

q_1 corresponds to all strings having an odd no of 1's

② $L_2 = \{ w \in \{a,b\}^* \mid w \text{ ends with the substring } ab \}$

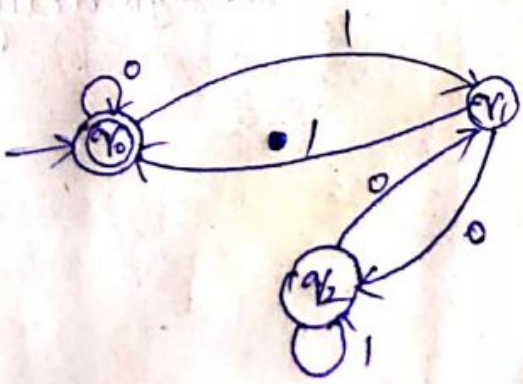


q_2 – all strings that end with ab

q_1 – all string that end with substring 'a'

q_0 – remaining strings.

③ $L_3 = \{ w \in \{0,1\}^* \mid w \text{ is divisible by } 3 \}$



0
1
10
11
100
101

Scanned with CamScanner

q_0 : ω string $\omega \bmod 3 \equiv 0$

q_1 : ω string $\omega \bmod 3 \equiv 1$

q_2 : ω string $\omega \bmod 3 \equiv 2$

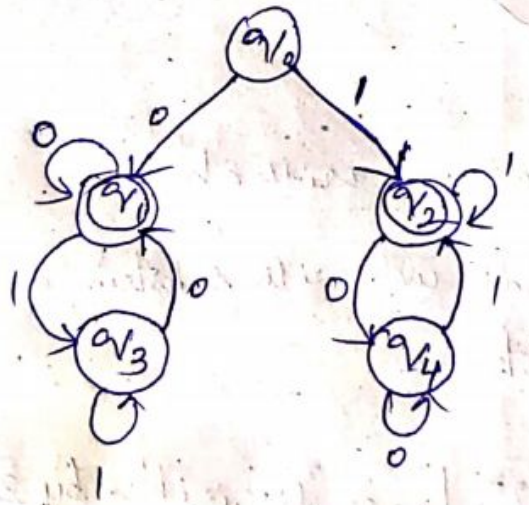
4) $L_4 = \{ \omega \in \{0,1\}^* \mid \omega$ begins and ends with the same symbol $\}$

Begin with 0 end with 0 q_1

- - - - - - - 0 end with 1 q_3

- - - - - - - 1 end with 0 q_4

- - - - - - - 1 end with 1 q_2



Note: for a language there can be many DFA's accepting it (in fact there can be infinitely many DFA's)

Scanned with CamScanner

Simpler Notations for DFA's

There are two preferred notations for describing automata:

1. A transition diagram, which is a graph such as the ones we saw earlier.

2. A transition table, which is a tabular listing of the δ-function, which by implication tells us the set of states and the input alphabet.

Transition Diagram

A transition diagram for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows:

a) For each state in Q there is a node.

b) For each state q in Q and each input symbol a in Σ, let $\delta(q, a) = p$. Then the transition diagram has an arc from node q to node p, labeled a. If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc, labeled by the list of these symbols.

c) There is an arrow into the start state q_0, labeled Start. This arrow does not originate at any node.

d) Nodes corresponding to accepting states (those in F) are marked by a double circle. States not in F have a single circle.

Scanned with CamScanner

DFA that accepts all and only the strings of 0's and 1's that have the sequence 01 some where in the string.

We can write the Language L as

$\{ w \mid w$ is of the form $x01y$ for some strings x and y consisting of 0's and 1's only $\}$

Transition tables:

A transition table is a conventional, tabular representation of a function like δ that takes two arguments and returns a value. The rows of the table correspond to the states, and the columns correspond to the inputs. The entry for the row corresponding to state q and the column corresponding to input a is the state $\delta(q, a)$.

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Transition table for the above DFA.

Scanned with CamScanner

The Language of a DFA

Now, we can define the language of a DFA $A = (Q, \Sigma, \delta, q_0, F)$. This language is denoted $L(A)$ and is defined by

$$L(A) = \{ w \mid \delta(q_0, w) \text{ is in } F \}$$

That is, the language of A is the set of strings w that take the start state q_0 into one of the accepting states. If L is $L(A)$ for some DFA A, then we say L is a regular language.

Nondeterministic finite Automata

A nondeterministic finite automaton (NFA) has the power to be in several states at once.

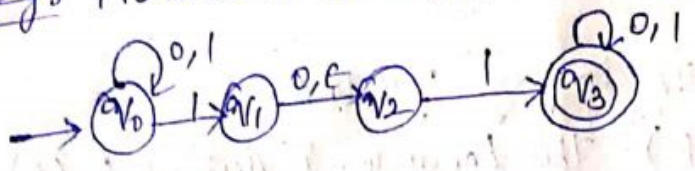
- from a state q, on an input symbol a, the automaton can go to multiple states K $(K \geq 0)$
- Computation happens simultaneously along each of these paths.
- Has ϵ-transitions if there is a ϵ-transition

The automaton moves to the states q_i and q_j without reading the next input

An input is accepted if there is some computation path that leads to an accept state

Scanned with CamScanner

eg: Nondeterministic finite Automata.



$(q_0, 1) \rightarrow \{q_0, q_1\}$

$(q_1, 1) \rightarrow \{\phi\}$

$(q_1, 0) \rightarrow \{q_2\}$

ε-transition

$(q_1, \epsilon) \rightarrow \{q_1, q_2\}$

Note: The transitions are labeled with symbols from the set

$$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$$

Consider an input $w = 010110$

Set of states the automata is in	Input symbol read
$\{q_0\}$ Initial state	0
$\{q_0\}$	1
$\{q_0 q_1 q_2\}$	0
$\{q_0 q_2\}$	1
$\{q_0 q_1 q_2 q_3\}$	1
$\{q_0 q_1 q_2 q_3\}$	0
$\{q_0, q_2, q_3\}$	Input accepted

Scanned with CamScanner

Definition of Non deterministic finite Automata

NFA is represented essentially like a DFA.

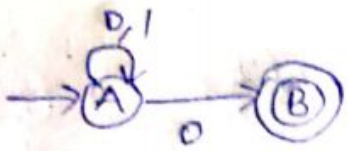
$$A = (Q, \Sigma, \delta, q_0, F)$$

1) Q is a finite set of states
2) Σ is a finite set of input symbols
3) q_0, a member of Q, is the start state
4) F, a subset of Q, is the set of final (or accepting) states
5) δ, the transition function is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q. Notice that the only difference b/w an NFA and a DFA is in the type of value that δ returns: a set of states in the case of an NFA and a single state in the case of a DFA.

DFA	NFA
$(q, a) \rightarrow$ single state	$(q, a) \rightarrow$ multiple states
Single Computation	multiple Computation
No ϵ-transition	we have ϵ-transitions.
Accept if the Computation ends at an accept state	Accept if one of the Computation paths ends at an accept state

Scanned with CamScanner

NFA example - 1



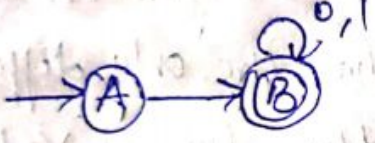
L = {set of all strings that end with 0}

Eg: 100

NFA - Example 2

L = {set of all strings that start with 0}

= {0, 00, 01, 000, ...}

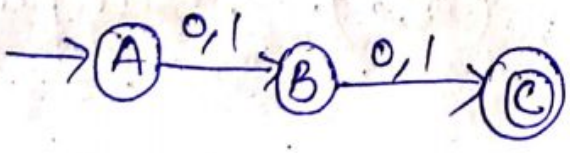


Eg: 001

Construct a NFA that accepts set of all strings over {0,1} of length 2

$\Sigma = \{0,1\}$

L = {00, 01, 10, 11}



Scanned with CamScanner

The Language of an NFA

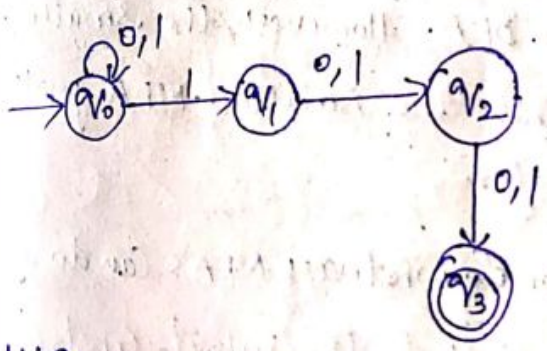
An NFA accepts a string 'w' if it is possible to make any sequence of choices of next state, while reading the characters of w, and go from the start state to any accepting state. The fact that other choices using the input symbols of w lead to a nonaccepting state or do not lead to any state at all, does not prevent w from being accepted by the NFA as a whole.

If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA, then

$$L(A) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \phi \}$$

That is, $L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

$L_1 = \{ w \in \{0,1\}^* \mid 3^{rd}$ last symbol of w is a $1 \}$



1110

$\{q_0\} \xrightarrow{1} \{q_0, q_1\} \xrightarrow{1} \{q_0, q_1, q_2\} \xrightarrow{1} \{q_0, q_1, q_2, q_3\} \xrightarrow{0} \{q_0, q_2, q_3\}$

Input accepted.

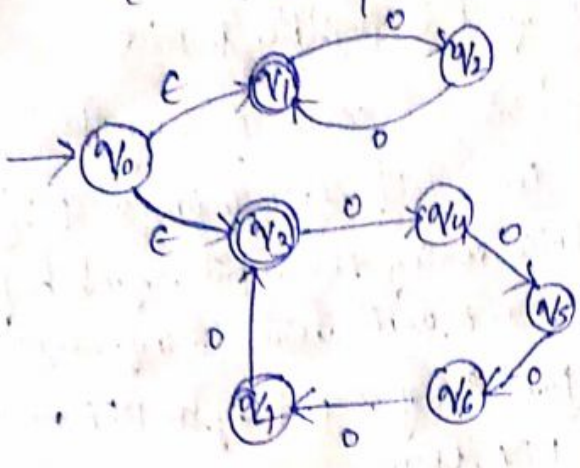
1010

$\{q_0\} \xrightarrow{1} \{q_0, q_1\} \xrightarrow{0} \{q_0, q_2\} \xrightarrow{1} \{q_0, q_1, q_3\} \xrightarrow{0} \{q_0, q_2\}$

Input Not accepted.

Scanned with CamScanner

$L_2 = \{ w \in \{0\}^* \mid |w| \text{ is divisible by 2 or by 5} \}$



Equivalence of Deterministic and Non deterministic finite Automata

Although there are many languages for which an NFA is easier to construct than a DFA. It is a surprising fact that every language that can be described by some NFA can also be described by some DFA. However, the smallest DFA can have 2^n states while the smallest NFA for the same language has only 'n' states.

The proof that DFA's can do whatever NFA's can do involves an important "Construction" called the subset construction because it involves constructing all subsets of the set of states of the NFA.

The subset construction starts from an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ Its goal is the description of a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(D) = L(N)$.

Scanned with CamScanner

Notice that the input alphabets of the two automata are the same and the start state of D is the set containing only the start state of N. The other components of D are constructed as follows.

- Q_D is the set of subsets of Q_N ie, Q_D is the power set of Q_N. Note that if Q_N has n states then Q_D will have 2^n states. often, not all these states are accessible from the start state of Q_D. Inaccessible states can be thrown away; so effectively, the number of states of D may be much smaller than 2^n.

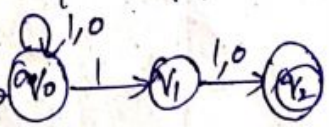
- F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \phi$ That is, F_D is all sets of N's states that include at least one accepting state of N.

- for each set $S \subseteq Q_N$ and for each input symbol a in Σ

$$\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$$

That is, to compute $\delta_D(S,a)$ we look at all the states p in S, see what states N goes to from p on input a, and take the union of all those states.

AN NFA accepting the set of all strings whose second last symbol is 1



δ	0	1
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$* \{q_1\}$	$\{q_2\}$	$\{q_2\}$
$* \{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$* \{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$* \{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
$* \{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

Complete subset construction from fig: 2.9

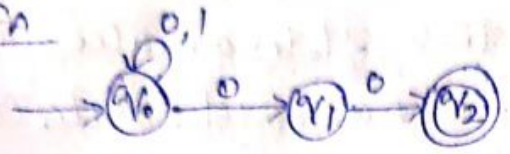
Scanned with CamScanner

Conversion of NFA to DFA

every DFA is an NFA but not vice versa
But there is an equivalent DFA for every NFA.

$L = \{000, 100, 1100 \ldots\}$

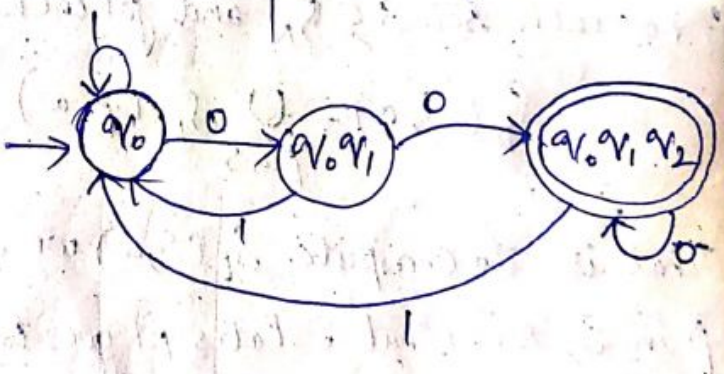
NFA



states	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	ϕ
q_2	ϕ	ϕ

DFA

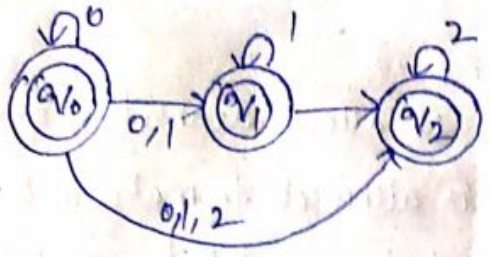
states	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
$q_0 q_1$	$\{q_0 q_1 q_2\}$	$\{q_0\}$
$q_0 q_1 q_2$	$\{q_0 q_1 q_2\}$	$\{q_0\}$



Scanned with CamScanner

NFA

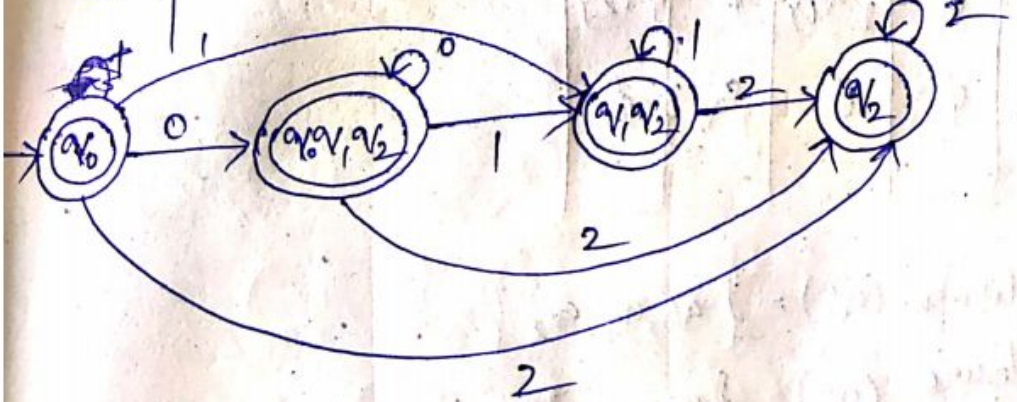
DFA



	0	1	2
$\rightarrow q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

DFA

	0	1	2
$\rightarrow q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_0, q_1, q_2	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1, q_2	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$



Scanned with CamScanner

ε-NFA

This is extension to NFA, it allows transition on empty string. In effect, an NFA is allowed to make a transition spontaneously, without receiving an input symbol.

Q · Finite set of states

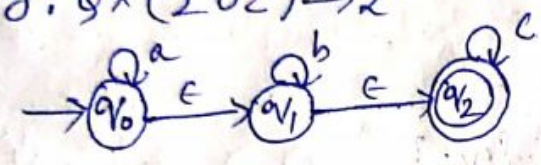
Σ input symbols

q_0 initial state

F set of final states

δ Transition function

$$\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$$



Every state on ε goes to itself.

ε-NFA

δ	a	b	c	ε
→ q_0	$\{q_0\}$	ϕ	ϕ	q_1
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

Scanned with CamScanner

NFA

δ'	a	b	c
$\rightarrow q_0$	$\{q_0, q_1, q_2\}$	$\{q_1\ q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1\ q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

$\delta'(q_0, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a))$

$\epsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, a))$

$\epsilon\text{-closure}(q_0)$

$= \{q_0, q_1, q_2\}$

$\delta'(q_0, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b))$

$\epsilon\text{-closure}(\delta((q_0, q_1, q_2), b))$

$\epsilon\text{-closure}(\{q_1\})$

$\{q_1, q_2\}$

$\delta'(q_0, c) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), c))$

$= \epsilon\text{-closure}(\delta((q_0, q_1, q_2), c))$

$= \epsilon\text{-closure}(\{q_2\})$

$= \{q_2\}$

Scanned with CamScanner

$\delta'(q_1, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), a))$

$= \epsilon\text{-closure}(\delta((q_1, q_2), a))$

ϕ

$\delta'(q_1, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b))$

$= \epsilon\text{-closure}(\delta((q_1, q_2), b))$

$= \epsilon\text{-closure}(\{q_1\}) = \{q_1, q_2\}$

$\delta'(q_1, c) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), c))$

$= \epsilon\text{-closure}(\delta(\{q_1, q_2\}, c))$

$\epsilon\text{-closure}(\{q_2\}) = \{q_2\}$

$\delta'(q_2, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a))$

$= \epsilon\text{-closure}(\delta(\{q_2\}, a))$

ϕ

$\delta'(q_2, b) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), b))$

$\epsilon\text{-closure}(\delta(\{q_2\}, b))$

ϕ

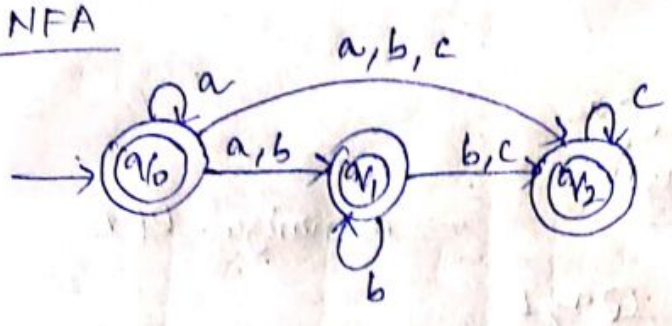
$\delta'(q_2, c) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), c))$

$\epsilon\text{-closure}\ \delta(\{q_2\}, c)$

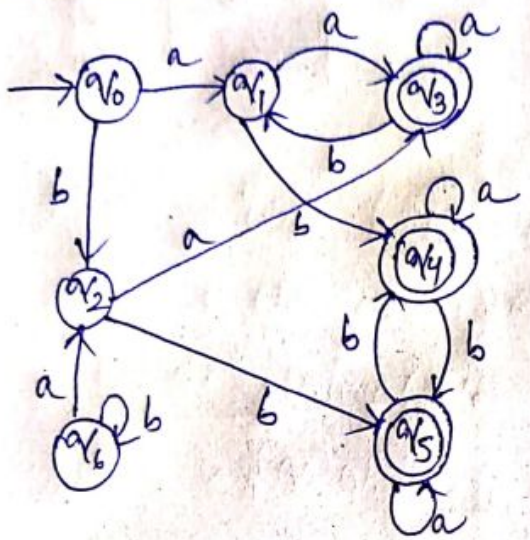
$= \epsilon\text{-closure}(\{q_2\})$

Scanned with CamScanner

NFA



Minimization of DFA



	a	b
q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_3	q_5
q_3	q_3	q_1
q_4	q_4	q_5
q_5	q_5	q_4
q_6	q_2	q_6

q_6 is unreachable from the start state q_0 so remove q_6

1) Remove unreachable
2) Mark the final states and divide into

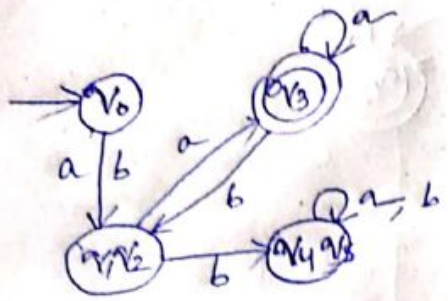
Accepting state (final) & Non accepting states (Non final)

$\pi_0 = \{q_0, q_1, q_2\}_{G_1}$ $\{q_3, q_4, q_5\}_{G_2}$

$\pi_1 = \{q_0\}_{G_{11}}$ $\{q_1, q_2\}_{G_{12}}$ $\{q_3\}_{G_{21}}$ $\{q_4, q_5\}_{G_{22}}$

$\pi_2 = \{q_0\}$ $\{q_1, q_2\}$ $\{q_3\}$ $\{q_4, q_5\}$

Scanned with CamScanner



→ minimized DFA.

Scanned with CamScanner

Moore Machine

- It is a 6 tuple notation

$M_0 = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

(q_0, q_1, q_2) Q = finite set of states

$(0,1)$ Σ = input symbol (alphabet)

δ = Transition function

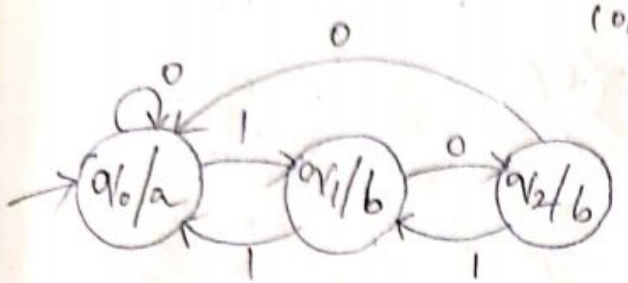
$Q \times \Sigma \rightarrow Q$

q_0 = start state

Δ = output symbol

λ = output function

$\lambda : Q \rightarrow \Delta$



00110

a a a b a a

Current	Next state 0	Next state 1	output
q_0	q_0	q_1	a
q_1	q_2	q_0	b
q_2	q_0	q_1	b

$q_0 \rightarrow a$

$q_1 \rightarrow b$

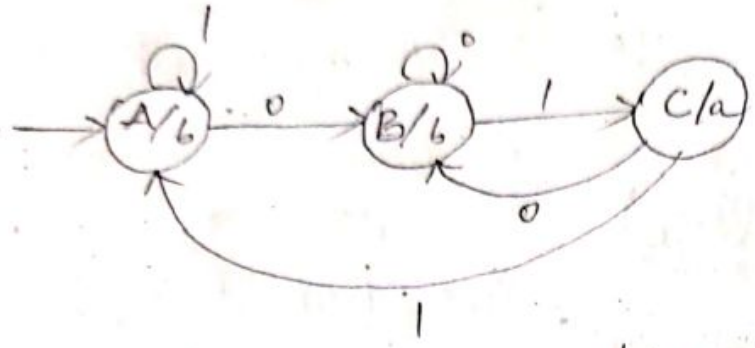
$q_2 \rightarrow b$

Scanned with CamScanner

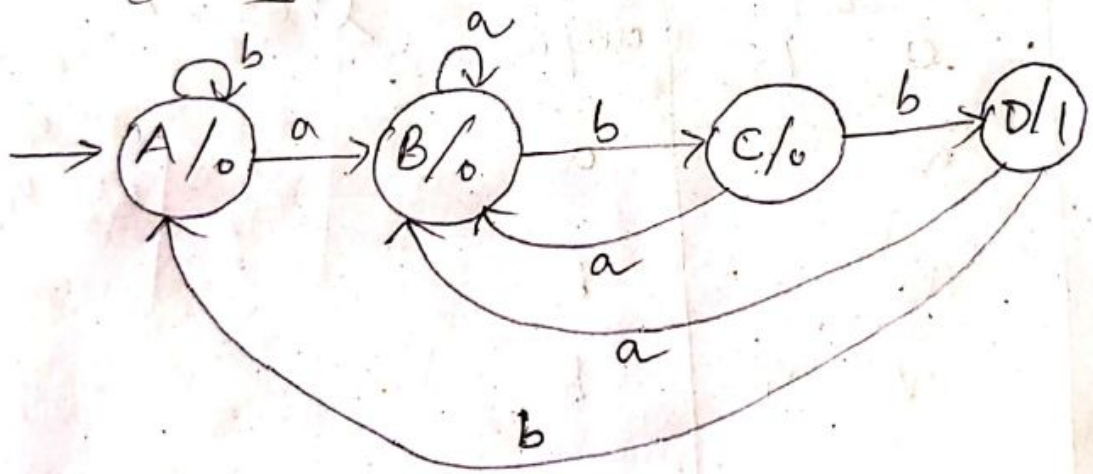
Construct a moore machine that prints 'a' whenever the sequence '01' is encountered in any i/p binary string.

$\Sigma = \{0, 1\}$

$\Delta = \{a, b\}$



Construction of Moore machine that counts the occurences of the sequence 'abb' in any input string over $\{a, b\}$



Scanned with CamScanner

Mealy Machine:

$M_e = \{Q, \Sigma, \Delta, \delta, \lambda, q_0\}$

$(q_0 q_1)$ Q = finite set of states

(a, b) Σ = input alphabet

δ = Transition function

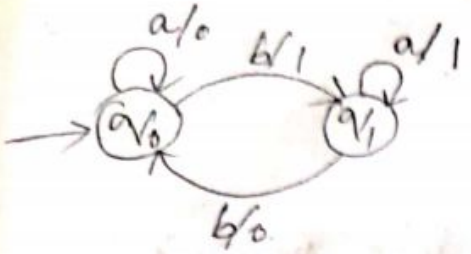
$\delta : Q \times \Sigma \rightarrow Q$

q_0 = Initial state

Δ = output symbol

λ = output function

$\lambda : Q \times \Sigma \rightarrow \Delta$



Current state	Next state			
	Input = a		Input = b	
	State	o/p	State	o/p
q_0	q_0	0	q_1	1
q_1	q_1	1	q_0	0

q_0 a $\rightarrow$ q_0

q_0 b $\rightarrow$ q_1

q_1 a $\rightarrow$ q_1

q_2 b $\rightarrow$ q_1

$q_0, a \rightarrow 0$

$q_0, b \rightarrow 1$

$q_1, a \rightarrow 1$

$q_1, b \rightarrow 0$

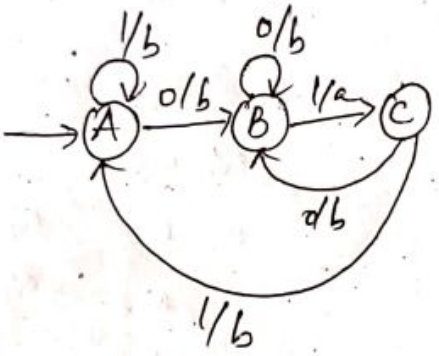
Scanned with CamScanner

Construction of Mealy machine

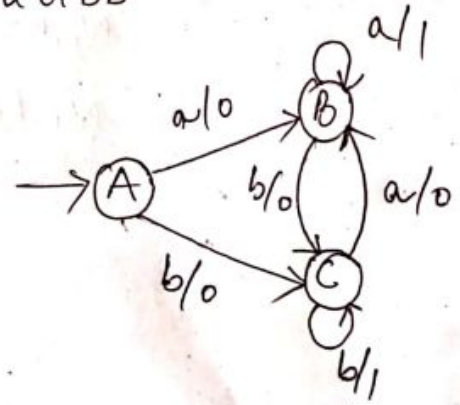
1) Construct a mealy machine that produces the 1's Compilement of any binary input string.

2) Construct a mealy machine that Prints 'a' whenever the sequence '01' is encountered in any input binary string.

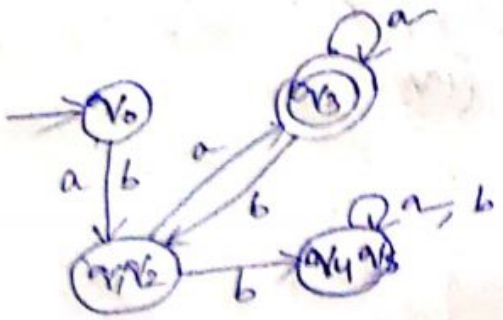
$\Sigma = \{0, 1\}$ $\Delta = \{a, b\}$



3) Design a mealy machine accepting the language Consisting of strings from Σ^* where $\Sigma = \{a, b\}$ and the strings should end with either a a or bb



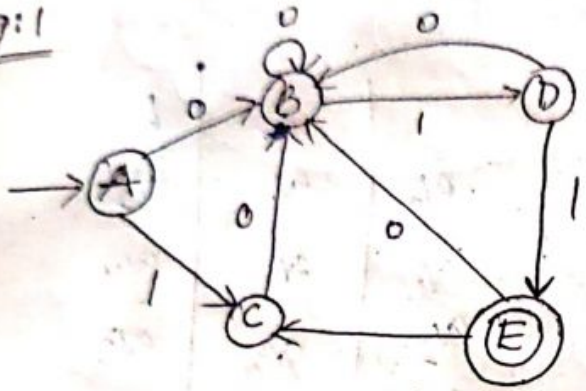
Scanned with CamScanner



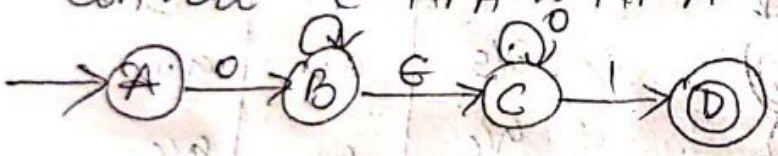
→ minimized DFA.

Minimization of DFA

eg: 1



Convert ε-NFA to NFA



Scanned with CamScanner

Regular Expressions

- Regular expressions are another type of language defining notation.
- They are algebraic representation of languages

The algebraic laws that apply to regular expressions have significant resemblance to the algebraic laws of arithmetic expressions.

Regular expression can define exactly the same language that the various forms of automata describe

However, regular expressions offer something that automata describe do not, ie a declarative way to express the strings we want to accept.

Thus, regular expressions serve as the input language for many systems that process strings.

Examples include

1) Search Commands such as the UNIX grep or equivalent Commands for finding strings that one sees in web browsers or text formatting systems. These systems use a regular expression like notation for describing patterns that the user wants to find in a file. Different search systems convert the regular expressions into either a DFA or an NFA and simulate that automaton on the file being searched.

Scanned with CamScanner

2) Lexical analyzer generators such as lex or flex. Recall that a lexical analyzer is the component of a compiler that breaks the source program into logical units (called tokens) of one or more characters that have a shared significance. Eg: of tokens include keywords (Eg: while), identifiers (Eg: any letter followed by zero or more letters and/or digits) and signs such as + or <=.
A lexical analyzer generator accepts descriptions of the forms of tokens, which are essentially regular expressions, and produces a DFA that recognizes which token appears next on the input.

The operators of Regular Expressions

(I) Regular expressions denote languages.
The Regular expression $01^* + 10^*$ denotes the language consisting of all strings that are either a single 0 followed by any number of 1's or a single 1 followed by any number of 0's.

we need to learn the three operations on languages that the operators of regular expressions represent

(1) The union of two languages L and M, denoted $L \cup M$, is the set of strings that are in either L or M or both. for Eg: if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$ then $L \cup M = \{\epsilon, 10, 001, 111\}$.

Scanned with CamScanner

(2) The Concatenation of Languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.

We denote concatenation of languages either with a dot or with no operator at all, although the concatenation operator is frequently called "dot".

eg: $L = \{001, 10, 111\}$ and $M = \{\epsilon, 001\}$ then $L \cdot M$ is

$\{001, 10, 111, 001001, 10001, 111001\}$. The first three strings in LM are the strings in L concatenated with ϵ. Since ϵ is the identity for concatenation, the resulting strings are the same as the strings of L. However, the last three strings in $L \cdot M$ are formed by taking each string in L and concatenating it with the second string in M, which is 001. For instance, 10 from L concatenated with 001 from M gives us 10001 for LM.

(3) The closure (or star or Kleene closure) of a language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (ie the same string may be selected more than once) and concatenating all of them. For instance, if $L = \{0, 1\}$ then L^* is all strings of 0's and 1's. If $L = \{0, 11\}$ then L^* consists of those strings of 0's and 1's such that the 1's come in pairs eg: 011, 11110 and ϵ but not 010111 or 101. More formally, L^* is the infinite union $\bigcup_{i \geq 0} L^i$; where $L^0 = \{\epsilon\}$, $L^1 = L$ and L^i for $i > 1$ is $LL \cdots L$ (the concatenation of i copies of L.

Scanned with CamScanner

Algebras of all kinds start with some elementary expressions, usually constants and/or variables. Algebras then allow us to construct more expressions by applying a certain set of operators to these elementary expressions ~~and to~~.

The algebra of regular expressions use constants and variables that denote languages and operators for the three operations of union, dot and star. In this definition, we not only describe what the Regular expressions are, but for each regular expression E we describe the language it represents, which we denote $L(E)$.

- The constants ϵ and ϕ are regular expressions denoting the languages $\{\epsilon\}$ and ϕ respectively. That is, $L(\epsilon) = \{\epsilon\}$ and $L(\phi) = \phi$.

- If E and F are regular expressions, then $E+F$ is a regular expression denoting the union of $L(E)$ and $L(F)$. That is, $L(E+F) = L(E) \cup L(F)$.

- If E and F are regular expressions, then EF is a regular expression denoting the concatenation of $L(E)$ and $L(F)$. That is, $L(EF) = L(E)L(F)$.

- If E is a regular expression, then E^* is a regular expression, denoting the closure of $L(E)$. That is $L(E^*) = (L(E))^*$

Scanned with CamScanner

- If E is a regular expression, then (E), a parenthesized E, is also a regular expression, denoting the same language as E. formally, $L((E)) = L(E)$

write regular expressions for the following languages over $\{0,1\}^*$

(a) The set of all strings that begin with 110

R.E = $110(0+1)^*$

(b) The set of all strings that contain 1011

$(0+1)^* 1011 (0+1)^*$

(c) The set of all strings that contain exactly three 1's.

$0^* 1 0^* 1 0^* 1 0^*$

(d) The set of all strings such that the no. of 0s is odd

$1^*(01^*01^*)^* 01^*$

Regular expression for the set of all strings that do not contain 1101

$(0+10)^* 1^*$

Set of all strings of 0's and 1's not containing 101 as a substring.

$0^*(1^* 000^*)^* 1^* 0^*$.

Scanned with CamScanner

The Regular expression $(0+1)^* 0 (0+1)^* 0 (0+1)^*$ represents the Language. The set of all strings Containing atleast two 0's.

The regular expression
$(b+ab)^* + (b+ab)^* a$
$a(b+ba)^* + (b+ba)^*$
represents the Language having input alphabets $\{a, b\}$ in which two a's do not Come together

Design Regular Expressions for following languages over $\{a,b\}$

1) Language accepting strings of length exactly 2.
2) Language accepting strings of length atleast 2
3) Language accepting strings of length atmost 2.

1) $L_1 = \{aa, ab, ba, bb\}$
$R = aa + ab + ba + bb$
$= a(a+b) + b(a+b)$
$(a+b)(a+b)$

2) $L_2 = \{aa, ab, ba, bb, aaa \ldots\}$
$R = (a+b)(a+b)(a+b)^*$

3) $L_3 = \{\epsilon, a, b, aa, ab, ba, bb\}$
$= (\epsilon + a + b)(\epsilon + a + b)$

Scanned with CamScanner

Regular Expressions

- Algebraic representation of languages.
- Applications in text editor to find strings.

  Compiler design.

Eg of regular expression:

Reg Exp	language
0	$\{0\}$
ϵ	$\{\epsilon\}$
$0 \cup 01$	$\{0, 01\}$
1^*	$\{\epsilon, 1, 11, 111, \ldots\}$
$(0 \cup 01)\, 1^*$	$\{0, 01, 011, 0111, \ldots\}$

Definition : R is said to be a regular expression (in short RE) if R has one of the following forms.

Reg Exp	Language	Comment
ϕ	$\{\ \}$	
ϵ	$\{\epsilon\}$	
a	$\{a\}$	$a \in \Sigma$
$R_1 \cup R_2$	$L(R_1) \cup L(R_2)$	
$R_1 \cdot R_2$	$L(R_1) \cdot L(R_2)$	
R_1^*	$(L(R_1))^*$	

Scanned with CamScanner

Note:

- for every regular expression R there is a unique lang $L(R)$ corresponding to it but converse is not true.

we often replace $\cup$ with + symbol and drop the $\cdot$ symbol

- Precedence of the (symbol) operators High to low

$(\,), *, \cdot, \cup$

$\phi^* = \{\epsilon\}$

More examples

R.E	language
01	$\{01\}$
$01+1$	$\{01, 1\}$
$(01+\epsilon)1$	$\{011, 1\}$
$(0+10)^*(\epsilon+1)$	$\{\epsilon, 1, 0, 01, 10, 101$ $00, 001, 010, 0101, 100, 1001$ $1010, 10101 \ldots\}$

Scanned with CamScanner

Examples (Lang $\longrightarrow$ RE)

Lang	R.E
$\{w \mid w$ has a single $1\}$	$0^* 1 0^*$
$\{w \mid w$ has atmost a single $1\}$	$0^* + 0^* 1 0^*$
$\{w \mid |w|$ is div by $3\}$	$[(0+1)(0+1)(0+1)]^*$
$\{w \mid w$ has a 1 at every odd position and $|w|$ is odd	$1((0+1)1)^*$

Scanned with CamScanner

Algebraic Laws for Regular expressions.

(i) Associativity and Commutativity.

Here are three laws that hold for regular expression

- $L+M = M+L$. This law, the Commutative law for union, says that we may take the union of two languages in either order.

- $(L+M)+N = L+(M+N)$. This law, the associative law for union, says that we may take the union of three languages either by taking the union of the first two initially or taking the union of the last two initially.

- $(LM)N = L(MN)$. This law, the associative law for Concatenation, says that we can Concatenate three languages by Concatenating either the first two or the last two initially.

Missing from this list is the "law" $LM = ML$, which would say that Concatenation is Commutative. However, this law is false.

Scanned with CamScanner

Identities and Annihilators

An identity for an operator is a value such that when the operator is applied to the identity and some other value, the result is the other value. For instance, 0 is the identity for addition, since $0+x = x+0 = x$, and 1 is the identity for multiplication since $1 \times x = x \times 1 = x$. An annihilator for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is the annihilator. For instance, 0 is an annihilator for multiplication, since $0 \times x = x \times 0 = 0$. There is no annihilator for addition.

There are three laws for regular expressions involving these concepts.

$\phi + L = L + \phi = L$. This law asserts that ϕ is the identity for union

$\epsilon L = L \epsilon = L$. This law asserts that ϵ is the identity for concatenation.

$\phi L = L \phi = \phi$. This law asserts that ϕ is the annihilator for concatenation.

These laws are powerful tools in simplifications.

Scanned with CamScanner

Distributive Laws

A distributive law involves two operators and asserts that one operator can be pushed down to be applied to each argument of the other operator individually. The most common example from arithmetic is the distributive law of multiplication over addition, that is, $x \times (y+z) = x \times y + x \times z$. Since multiplication is commutative, it doesn't matter whether the multiplication is on the left or right of the sum.

$L(M+N) = LM + LN$. This law, is the left distributive law of concatenation over union.

$(M+N)L = ML + NL$. This law, is the right distributive law of concatenation over union.

The Idempotent Law

An operator is said to be idempotent if the result of applying it to two of the same values as arguments is that value. The common arithmetic operators are not idempotent, $x+x \neq x$ in general and $x \times x \neq x$ in general. However, union and intersection are common examples of idempotent operators. Thus, for regular expressions, we may assert the following law:

$L+L=L$. This law, the idempotence law for union, states that if we take the union of two identical expressions, we can replace them by one copy of the expression

Scanned with CamScanner

Laws involving closures

$(L^*)^* = L^*$. This law says that closing an expression that is already closed does not change the language. The language of $(L^*)^*$ is all strings created by concatenating strings in the language of L^*. But those strings are themselves composed of strings from L. Thus, the string in $(L^*)^*$ is also a concatenation of strings from L and is therefore in the language of L^*.

- $\phi^* = \epsilon$. The closure of ϕ contains only the string ϵ,
- $\epsilon^* = \epsilon$. It is easy to check that the only string that can be formed by concatenating any number of copies of the empty string is the empty string itself.
- $L^+ = LL^* = L^*L$. L^+ is defined to be $L + LL + LLL + \ldots$

Also $L^* = \epsilon + L + LL + LLL + \ldots$

$LL^* = L\epsilon + LL + LLL + LLLL + \ldots$

- $L^* = L^+ + \epsilon$

$(L+M)^* = (L^* M^*)^*$

Scanned with CamScanner

Identities of Regular expression

1. $\phi + R = R$
2. $\phi R + R\phi = \phi$
3. $\epsilon R = R\epsilon = R$
4. $\epsilon^* = \epsilon$ and $\phi^* = \epsilon$
5. $R + R = R$
6. $R^* R^* = R^*$
7. $RR^* = R^* R$
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = \epsilon + R^* R = R^*$
10. $(PQ)^* P = P(QP)^*$
11. $(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^*$
12. $(P+Q)R = PR + QR$ and $R(P+Q) = RP + RQ$

Scanned with CamScanner

Ardens Theorem

If P and Q are two Regular Expressions over Σ, and if P does not contain ϵ, then the following equation in R given by $R = Q + RP$ has a unique solution i.e. $R = QP^*$

Scanned with CamScanner

Prove that $(1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$

is equal to $0^*1(0+10^*1)^*$

$L.H.S = (1+00^*1)+(1+00^*1)(0+10^*1)^*(0+10^*1)$

$(1+00^*1)\left[\epsilon+(0+10^*1)^*(0+10^*1)\right]$

$\boxed{\therefore \epsilon+R^*R=R^*}$

$(1+00^*1)(0+10^*1)^*$

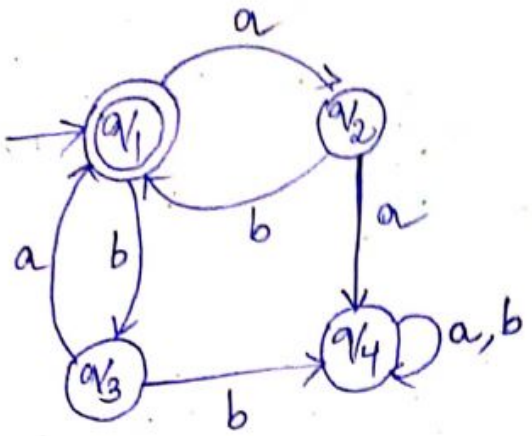
$(\epsilon(1+00^*1))(0+10^*1)^*$ $\qquad \epsilon\cdot R=R.$

$(\epsilon+00^*)1(0+10^*1)^*$

$0^*1(0+10^*1)^* = R.H.S$

Scanned with CamScanner

Designing Regular Expression from a DFA



$q_1 = \epsilon + q_2 b + q_3 a$ —— (1)

$q_2 = q_1 a \rightarrow$ (2)

$q_3 = q_1 b \rightarrow$ (3)

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b \rightarrow$ (4)

(I) $q_1 = \epsilon + q_2 b + q_3 a$

putting the values of q_2 and q_3 from (II) & (III)

$q_1 = \epsilon + q_1 ab + q_1 ba$

$\underbrace{q_1}_{R} = \underbrace{\epsilon}_{Q} + \underbrace{q_1}_{R} \underbrace{(ab+ba)}_{P}$

$R = Q + RP$

$R = QP^*$ Ardens theorem

$q_1 = \epsilon \cdot (ab+ba)^*$

$q_1 = (ab+ba)^* \rightarrow$ Regular expression.

Scanned with CamScanner

Designing Regular Expression – Examples

When there are multiple final states.

Find the Regular Expression for the following DFA

$q_1 = \epsilon + q_1 0 \rightarrow (1)$

$q_2 = q_1 1 + q_2 1 \rightarrow (2)$

$q_3 = q_2 0 + q_3 0 + q_3 1 \rightarrow (3)$

Final state q_1

$q_1 = \epsilon + q_1 0$ $\quad R = Q + RP$

R Q R P $\quad R = QP^*$ Arden's theorem

$q_1 = \epsilon \cdot 0^*$ $\quad \epsilon \cdot R = R$

$q_1 = 0^*$

Final state q_2

$q_2 = q_1 1 + q_2 1$

$q_2 = 0^* 1 + q_2 1$ Putting value of q_1 from (4)

R Q R P $\quad R = Q + RP$

$R = QP^*$

$q_2 = 0^* 1 (1)^*$

R = union of both final states

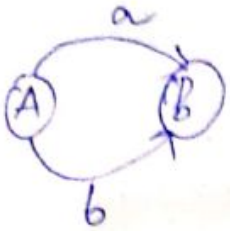
$= 0^* + 0^* 11^*$ $\quad \epsilon + RR^* = R^*$

$= 0^* (\epsilon + 11^*) \rightarrow 0^* 1^*$

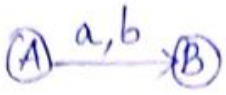
Scanned with CamScanner

Conversion of Regular Expression to Finite Automata

$(a+b)$



or



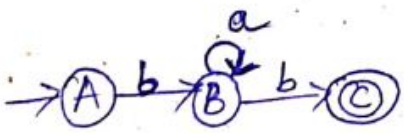
$(a \cdot b)$.

a^*

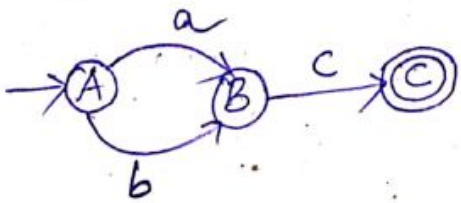
Examples

Convert the following Regular Expressions to their equivalent finite Automata.

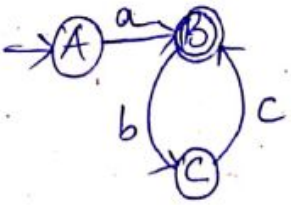
① ba^*b bb, bab, baab



② $(a+b)c$ ac, bc



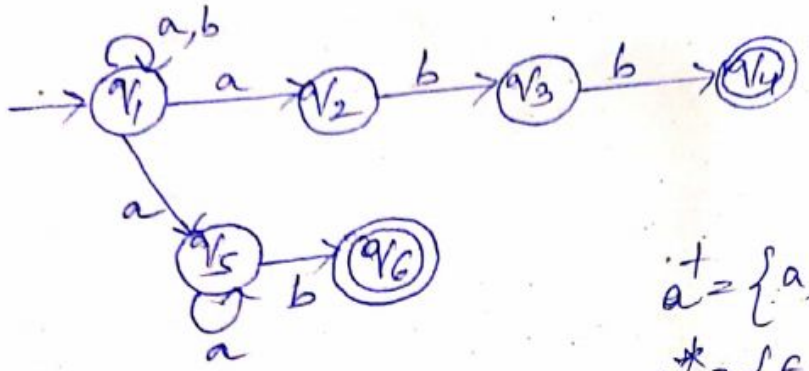
③ $a(bc)^*$ a, abc, abcbc.



Scanned with CamScanner

Convert the following Regular Expression to its Equivalent finite Automata

$(a+b)^* (abb + a^+ b)$



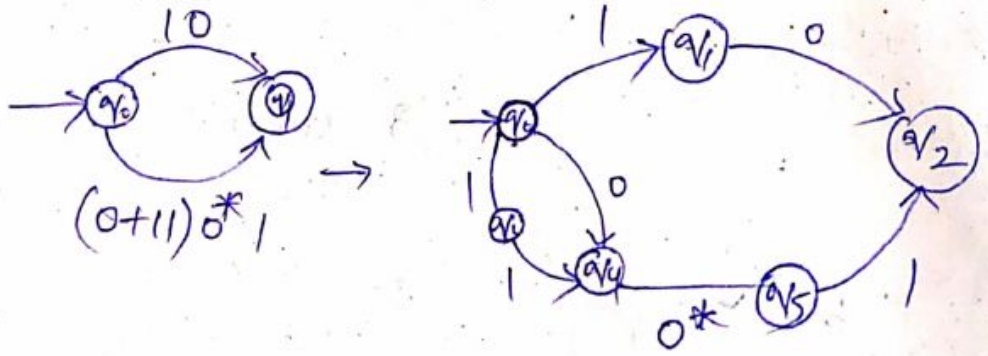
→ NFA.

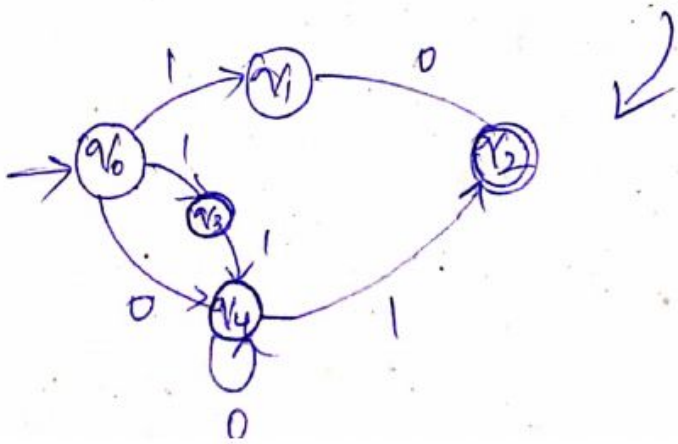
$a^+ = \{a, aa, aaa, \ldots\}$

$a^* = \{\epsilon, a, aa, \ldots\}$

Convert the following Regular Expression to its equivalent finite Automata

$10 + (0+11)0^* 1$





Scanned with CamScanner

Pumping Lemma (for Regular Languages)

Pumping Lemma is used to prove that a language is NOT Regular

It cannot be used to prove that a language is Regular.

If A is a Regular Language, then A has a pumping length 'p' such that any string 'S' where $|S| \geq p$ may be divided into 3 parts $S = xyz$ such that the following conditions must be true.

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

To prove that a language is not Regular using pumping LEMMA, follow the below steps:
(we prove using Contradiction)

→ Assume that A is Regular

→ It has to have a pumping Length (say p)

→ All strings longer than p can be pumped $|S| \geq p$

→ Now find a string 'S' in A such that $|S| \geq p$

→ Divide S into xyz

→ Show that $xy^iz \notin A$ for some i.

Scanned with CamScanner

→ Then Consider all ways that S can be divided into XYZ.

→ Show that none of these can satisfy all the 3 pumping Conditions at the same time.

→ S cannot be pumped == Contradiction.

Example

$A = \{a^n b^n \mid n \geq 0$ is not Regular $\}$

pumping length – P

Assume that A is regular

A should have a pumping length 'p'

any string $|s| \geq p$

$x \quad y \quad z$ for example pumping length = 12

we should divide this into 3 parts

(i) $|xy| \leq p$

(ii) $|y| \geq 1$

(iii) $xy^iz \in A$ for every $i \geq 0$

aaaa | a | a bbbbbb

$x \quad y \quad z$

$|xy| = 5 \leq 12$

$|y| = 1 \geq 1$

$xy^iz \in A$ for every $i \geq 0$

$xy^0z \in A$ if $i = 0$

$\underbrace{aaaa}_{5} (a)^0 a \underbrace{bbbbbb}_{6} \in A$

Scanned with CamScanner

Pumping Lemma (for Regular languages) Example - 2

Using pumping lemma prove that the Language $A = \{yy \mid y \in \{0,1\}^*\}$ is Not Regular

Assume that A is Regular
then it must have a pumping length p
Choose any string 's'.

$s = 0^p 1 0^p 1$

$x \quad y \quad z \quad p = 7$

$\underbrace{0000000}_{x\ y}\ \underbrace{1\ 0000001}_{z}$

$xy^iz = xy^2z$

$00000000010000001 \notin A$

$|y| = 4 \geq 1$

$|xy| \leq p$

$2+4 \leq 7$

$6 \leq 7$

It is Contradiction to our assumption

So the given language is Not Regular

Scanned with CamScanner

Closure Properties of Regular Languages

If certain languages are regular and a language L is formed from them by certain operations (eg: L is the union of two regular languages) then L is also regular. These theorems are often called closure properties of the regular languages, since they show that the class of regular languages is closed under the operation mentioned. Closure properties express the idea that when one languages are regular, then certain related languages are also regular.

Here is a summary of the principal closure properties for regular languages.

1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular
3. The complement of a regular language is regular.
4. The difference of two regular languages is regular
5. The reversal of a regular language is regular.
6. The closure (star) of a regular language is regular.
7. The concatenation of regular languages is regular
8. A homomorphism (substitution of strings for symbols) of a regular language is regular.
9. The inverse homomorphism of a regular language is regular

Scanned with CamScanner

Closure under union

If L and M are regular languages, then so is $L \cup M$

Closure under Concatenation

If L and M are regular languages, then so is LM.

Closure under Kleene closure

If L is a regular language, then so is L^*

Closure under Complementation

Starting with a regular expression, we could find a regular expression for its complement as follows.

1. Convert the regular expression to an ϵ-NFA
2. Convert that ϵ-NFA to a DFA by the subset construction.
3. Complement the accepting states of that DFA.
4. Turn the complement DFA back into a regular expression

If L is a regular language over alphabet Σ, then $\bar{L} = \Sigma^* - L$ is also a regular language.

Closure under Intersection

Once we have the ways of performing complementation and union, we can obtain the intersection of languages L and M by the identity.

$$L \cap M = \overline{\bar{L} \cup \bar{M}}$$

The intersection of two sets is the set of elements that are not in the complement of either set.

If L and M are regular languages, then so is $L \cap M$

Scanned with CamScanner

Closure under Difference

The fourth operation often applied to sets is related to the boolean operations : set difference. In terms of Languages, $L-M$ the difference of L and M, is the set of strings that are in Language L but not in Language M. The regular languages are also closed under this operation.

If L and M are regular Languages, then so is $L-M$

Reversal : If L is a Regular Language, so is L^R

The reversal of a string $a_1 a_2 \cdots a_n$ is the string written backwards that is $a_n a_{n-1} \cdots a_1$. We use w^R for the reversal of strings w. Thus, 0010^R is 0100, and $\epsilon^R = \epsilon$.

The reversal of a Language L, written L^R, is the language consisting of the reversals of all its strings. For instance, if $L = \{001, 10, 111\}$ then $L^R = \{100, 01, 111\}$.

Reversal is another operation that preserves regular languages, that is, if L is a regular languages, so is L^R

Given a language L that is $L(A)$ for some finite Automaton perhaps with nondeterminism and ϵ-transitions, we may construct an automaton for L^R by

1. Reverse all the arcs in the transition diagram for A.
2. Make the start state of A be the only accepting state for the new automaton.
3. Create a new start state p_0 with transitions on ϵ to all the accepting states of 'A'.

The result is an automaton that simulates A "in reverse" and therefore accepts a string w if and only if A accepts w^R.

Scanned with CamScanner

Homomorphisms

A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

example: The function h given by $h(0) = abb$ and $h(1) = ba$ is a homomorphism which replaces each 0 by abb and each 1 by ba. Thus $h(1011) = baabbbaba$.

formally, if h is a homomorphism on alphabet Σ, and $w = a_1 a_2 \cdots a_n$ is a string of symbols in Σ, then $h(w)$
$h(w) = h(a_1)h(a_2)\cdots h(a_n)$. That is, we apply h to each symbol of w and concatenate the results, in order.

for instance, if h is the homomorphism and $w = 0011$
then $h(w) = h(0)h(0)h(1)h(1) = (ab)(ab)(\epsilon)(\epsilon) = abab$

further, we can apply a homomorphism to a language by applying it to each of the strings in the language.
If 'L' is a language over alphabet Σ, and h is a homomorphism on Σ, then $h(L) = \{h(w) \mid w \text{ is in } L\}$. If L is the language of regular expression 10^*1 ie any number of 0's surrounded by single 1's, then $h(L)$ is the language $(ab)^*$
The reason is that h effectively drops the 1's, since they are replaced by ϵ, and turns each 0 into ab.

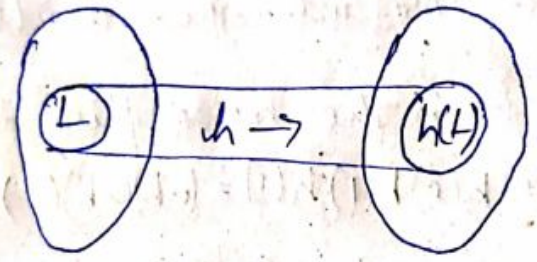
If L is a regular language over alphabet Σ, and h is a homomorphism on Σ, then $h(L)$ is also regular.

Scanned with CamScanner

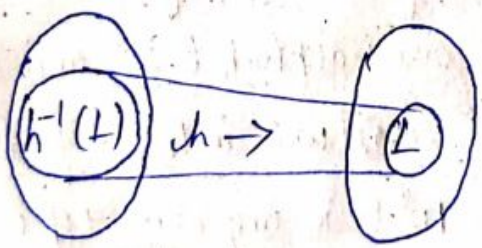
Inverse Homomorphism

Homomorphism may be applied "backwards" and in this mode they also preserve regular languages. Suppose h is a homomorphism from some alphabet Σ to strings in another alphabet T. Let L be a language over alphabet T. Then $h^{-1}(L)$ (read "h inverse of L"), is the set of strings w in Σ^* such that $h(w)$ is in L.

The following figure represents the effect of a homomorphism on a language L in part (a) and the effect of an inverse homomorphism in part (b).



(a)



(b)

Homomorphism applied in the forward and inverse directions.

Scanned with CamScanner

Example

Let L be the language of regular expression $(00+1)^*$. That is, L consists of all strings of 0's and 1's such that all the 0's occur in adjacent pairs. Thus, 00100111 and 10000111 are in L, but 000 and 10100 are not.

Let h be the homomorphism defined by $h(a)=01$ and $h(b)=10$. We claim that $h^{-1}(L)$ is the language of regular expression $(ba)^*$, that is, all strings of repeating ba pairs. We shall prove that $h(w)$ is in L if and only if w is of the form baba ... ba.

Suppose w is n repetitions of ba for some $n \geq 0$. Note that $h(ba) = 1001$, so $h(w)$ is n repetitions of 1001. Since 1001 is composed of two 1's and a pair of 0's, we know that 1001 is in L. Therefore any repetition of 1001 is also formed from 1 and 00 segments and is in L. Thus, $h(w)$ is in L.

Now, we must assume that $h(w)$ is in L and show that w is of the form baba ... ba. There are 4 conditions under which a string is not ~~in~~ of that form, and we shall show that if any of them hold then $h(w)$ is not in L. That is, we prove the contrapositive of the statement we set out to prove

① If w begins with a, then $h(w)$ begins with 01. It therefore has an isolated 0, and is not in L.

② If w ends in b, then $h(w)$ ends in 10, and again there is an isolated 0 in $h(w)$.

③ If w has two consecutive a's, then $h(w)$ has a substring 0101. Here too, there is an isolated 0 in w

Scanned with CamScanner

(4) Likewise, if w has two consecutive b's, then $h(w)$ has substring 1010 and has an isolated 0.

Thus, whenever one of the above cases hold, $h(w)$ is not in L. If none of 1 through 4 hold. Then (1) tells us w must begin with b and (2) tells us w ends with a. statements (3) and (4) tell us that a's and b's must alternate in w. Thus, the logical "OR" of (1) through (4) is equivalent to the statement "w is not of the form baba...ba". We have proved that the "OR" of (1) through (4) implies $h(w)$ is not in L. That statement is the contrapositive of the statement we wanted " if $h(w)$ is in L, then w is of the form baba...ba".

Scanned with CamScanner

Context free grammar and Language

Let us consider the language of palindromes. A palindrome is a string that reads the same forward and backward. In another way, string w is a palindrome if and only if $w = w^R$. If we describe only palindrome with alphabet $\{0,1\}$. This language includes strings like 0110, 11011 and ϵ but not 011 or 0101.

~~It is easy to verify that the language L.~~

There is a natural, recursive definition of when a string of 0's and 1's is in L_{pal}. It starts with a basis saying that a few obvious strings are in L_{pal}, and then exploits the idea that if a string is a palindrome it must begin and end with the same symbol. Further, when the first and last symbols are removed, the resulting string must also be a palindrome.

Basis: ϵ, 0 and 1 are palindromes.

INDUCTION: If w is a palindrome, so are 0w0 and 1w1.

No string is a palindrome of 0's and 1's, unless it follows from this basis and induction rule.

A context-free grammar is a formal notation for expressing such recursive definition of languages. A grammar consists of one or more variables that represent classes of strings. ie languages. In this example we have need for only one variable P, which represents the set of palindromes, that is the class of strings forming the language L_{pal}.

Scanned with CamScanner

There are rules that say how the strings in each class are constructed. The construction can use symbols of the alphabet, strings that are already known to be in one of the classes or both.

<u>Example:</u> The rules that define the palindromes, expressed in the context free grammar notation, are shown in figure

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

The first three rules form the basis. They tell us that the class of palindromes includes the strings ϵ, 0 and 1. None of the right sides of these rules contains a variable which is why they form a basis for the definition.

The last two rules form the inductive part of the definition for instance, rule 4 says that if we take any string w from the class P, then 0w0 is also in class P. Rule 5 likewise tells us that 1w1 is also in P.

Scanned with CamScanner

Definition of Context Free Grammars.

(1) There is a finite set of symbols that form the strings of the languages over $\{0,1\}$ we call this alphabet the terminals or terminal symbols.

(2) There is a finite set of variables, also called sometimes nonterminals or syntactic categories. Each variable represents a language ie a set of strings. In our example above, there was only one variable P, which we used to represent the class of palindromes over alphabet $\{0,1\}$

(3) One of the variables represents the language being defined, it is called the start symbol. other variables represent auxiliary classes of strings that are used to help define the language of the start symbol. In our example, P the only variable, is the start symbol.

4. There is a finite set of productions or rules that represents the recursive definition of a language. Each production consists of.

(a) A variable that is being defined by the production This variable is often called the head of the production

(b) The Production symbol $\rightarrow$

(c) A string of zero or more terminals and variables. This string, called the body of the production, represents one way to form strings in the language of the variable of the head. In so doing, we leave terminals unchanged and substitute for each variable of the body any string that is known to be in the language of that variable.

Scanned with CamScanner

The four components just described form a Context-free grammar or just grammar or CFG. We shall represent a CFG 'G' by its four components, that is $G = (V, T, P, S)$ where V is the set of variables, T - the terminals, P the set of productions and S the start symbol.

1. $E \rightarrow I$
2. $E \rightarrow E+E$
3. $E \rightarrow E*E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

fig

In this grammar we have 2 variables.

E represents ~~expression~~ start symbol.

I represents Identifier

The grammar for expressions is stated formally as $G = (\{E, I\}, T, P, E)$ where T is the set of symbols $\{+, *, (,), a, b, 0, 1\}$ and P is the set of Productions shown in above fig.

Scanned with CamScanner

Compact Notation for production

The productions $A \to \alpha_1$, $A \to \alpha_2$, ... $A \to \alpha_n$ can be replaced by the notation $A \to \alpha_1 | \alpha_2 | \ldots \alpha_n$.
For instance, the grammar for palindromes from can be written as $P \to \epsilon | 0 | 1 | 0P0 | 1P1$.

Derivation using a grammar

The process of deriving strings by applying productions from head to body requires the definition of a symbol.
$G = (V, T, P, S)$ is a CFG. Let $\alpha A \beta$ be a string of terminals and variables, with A a variable. That is, α and β are strings in $(V \cup T)^*$ and A is in V.
Let $A \to \gamma$ be a production of G. Then we say $\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$
If G is understood, we just say $\alpha A \beta \Rightarrow \alpha \gamma \beta$. Notice that one derivation step replaces any variable anywhere in the string by the body of one of its production.

Scanned with CamScanner

Notation for Context-free Grammar Derivations

① Lower-case letters near the beginning of the alphabet a, b and so on, are terminal symbols. we shall also assume that digits and other characters such as + or parentheses are terminals.

② upper case letters near the beginning of the alphabet A, B and so on, are variables.

③ Lower-case letters near the end of the alphabet, such as w or z; are strings of terminals.

④ upper-case letters near the end of the alphabet such as X or Y are either terminals or variables

⑤. Lower-case Greek letters, such as α and β are strings consisting of terminals and/or variables.

Leftmost and Rightmost Derivations

In order to restrict the number of choices we have in deriving a string, it is often useful to require that at each step we replace the leftmost variable by one of its production bodies. Such a derivation is called a leftmost derivation, we indicate that a derivation is leftmost by using the relations $\underset{lm}{\Rightarrow}$ and $\overset{*}{\underset{lm}{\Rightarrow}}$, for one or many steps, respectively

Scanned with CamScanner

Similarly, it is possible to require that at each step the rightmost variable is replaced by one of its bodies. If so, we call the derivation rightmost (~~derivation steps~~, ~~respectively~~ and use the symbols $\underset{rm}{\Rightarrow}$ and $\overset{*}{\underset{rm}{\Rightarrow}}$ to indicate one or many rightmost derivation steps

<u>Example</u> Leftmost derivation

$$E \underset{lm}{\Rightarrow} E*E \underset{lm}{\Rightarrow} (E)*E \Rightarrow (E+E)*E \underset{lm}{\Rightarrow} (I+E)*E$$

$$\underset{lm}{\Rightarrow} (I|+E)*E$$

$$\underset{lm}{\Rightarrow} (I0|+E)*E \underset{lm}{\Rightarrow} (I|0|+E)*E \underset{lm}{\Rightarrow} (a|0|+E)*E$$

$$\underset{lm}{\Rightarrow} (a|0|+I)*E \underset{lm}{\Rightarrow} (a|0|+I|)*E \underset{lm}{\Rightarrow} (a|0|+b|)*E$$

$$\underset{lm}{\Rightarrow} (a|0|+b|)*(E) \underset{lm}{\Rightarrow} (a|0|+b|)*(E+E) \underset{lm}{\Rightarrow} (a|0|+b|)*(I+E)$$

$$\underset{lm}{\Rightarrow} (a|0|+b|)*(I|+E) \underset{lm}{\Rightarrow} (a|0|+b|)*(a|+E)$$

$$\underset{lm}{\Rightarrow} (a|0|+b|)*(a|+I) \underset{lm}{\Rightarrow} (a|0|+b|)*(a|+b)$$

$$E \underset{lm}{\Rightarrow} (a|0|+b|)*(a|+b)$$

Scanned with CamScanner

There is also a rightmost derivation which makes the same replacements for each of the variables though in a different order. It is given by

$$E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*(E) \underset{rm}{\Rightarrow} E*(E+E) \underset{rm}{\Rightarrow} E*(E+I) \underset{rm}{\Rightarrow} E*(E+b)$$

$$\underset{rm}{\Rightarrow} E*(I+b)$$

$$\underset{rm}{\Rightarrow} E*(I1+b) \underset{rm}{\Rightarrow} (E)*(a1+b) \underset{rm}{\Rightarrow} (E+E)*(a1+b)$$

$$\underset{rm}{\Rightarrow} (E+I)*(a1+b) \underset{rm}{\Rightarrow} (E+I1)*(a1+b) \underset{rm}{\Rightarrow} (E+b1)*(a1+b)$$

$$\underset{rm}{\Rightarrow} (I+b1)*(a1+b) \underset{rm}{\Rightarrow} (I1+b1)*(a1+b) \underset{rm}{\Rightarrow} (I01+b1)*(a1+b)$$

$$\underset{rm}{\Rightarrow} (I101+b1)*(a1+b) \underset{rm}{\Rightarrow} (a101+b1)*(a1+b).$$

(This derivation allows us to conclude $E \overset{*}{\underset{rm}{\Rightarrow}} a*(a+b00)$)

Scanned with CamScanner

Sentential form

Derivations from the start symbol produce strings and we call them sentential forms.

That is, if $G = (V, T, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \overset{*}{\Rightarrow} \alpha$ is a sentential form.

If $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, then α is a left sentential form.

and if $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is a right sentential form.

Note that the language $L(G)$ is those sentential forms that are in T^*, ie, they consist solely of terminals.

Example

Consider the grammar for expressions from fig: a For example $E*(I+E)$ is a sentential form, since there is a derivation.

$$E \Rightarrow E*E \Rightarrow E*(E) \Rightarrow E*(E+E) \Rightarrow E*(I+E)$$

However this derivation is neither leftmost nor rightmost since at the last step, the middle E is replaced.

As an example of a left-sentential form, $a*E$ with the leftmost derivation.

$$E \underset{lm}{\Rightarrow} E*E \underset{lm}{\Rightarrow} a*E$$

Additionally, the derivation

$$E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*(E+E)$$

shows that $E*(E+E)$ is a right-sentential form.

Scanned with CamScanner

Parse Tree

- Parse tree is the hierarchical representation of terminals or non-terminals.
- These symbols (terminals or non-terminals) represent the derivation of the grammar to yield input strings.
- The start symbol of the grammar must be used as the root of the parse tree.
- Leaves of parse tree represent terminals.
- Each interior node represents productions of grammar.

Rules to Draw a parse tree

1) All leaf nodes need to be terminals
2) All interior nodes need to be non-terminals
3) In-order traversal gives original input string

Constructing parse tree

Let us fix on a grammar $G = (V, T, P, S)$. The parse trees for G are trees with the following conditions.

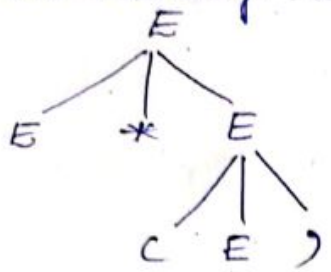
① Each interior node is labeled by a variable in V.

② Each leaf is labeled by either a variable, a terminal or ϵ. However, if the leaf is labeled ϵ, then it must be the only child of its parent.

③ If an interior node is labeled A, and its children are labeled $X_1, X_2, \ldots X_k$ respectively

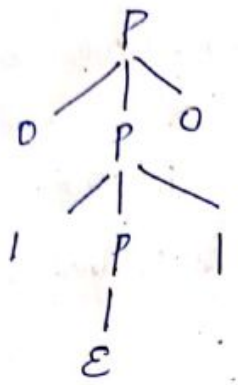
Scanned with CamScanner

A parse tree showing the derivation of E*(E) from E

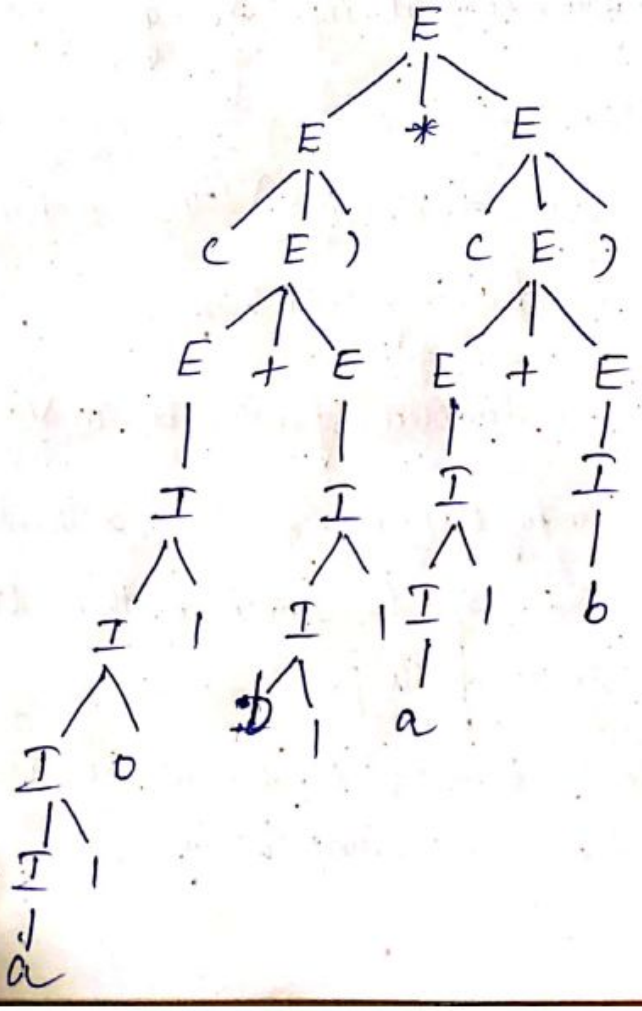


A parse tree showing the derivation $P \overset{*}{\Rightarrow} 0110$

$P \rightarrow \epsilon$
$P \rightarrow 0P0$
$P \rightarrow 1P1$
} grammar for palindrome



Parse tree showing (a|0|+b|)*(a|+b) is in the language of our expression grammar.



Scanned with CamScanner

Ambiguous Grammar

A Grammar is said to be Ambiguous if there exists two or more derivation tree for a string w.

Example: $G = \{S\}, \{a, b, +, *\}, P, S\}$ where P consists of

$S \rightarrow S+S \mid S*S \mid a \mid b$

The string a+a*b can be generated as

left derivation tree

$S \rightarrow \underline{S}+S$

$\rightarrow a+S$

$\rightarrow a+S*S$

$\rightarrow a+a*S$

$\rightarrow a+a*b$

left derivation tree

$S \rightarrow \underline{S}*S$

$\rightarrow S+S*S$

$\rightarrow a+S*S$

$\rightarrow a+a*S$

$\rightarrow a+a*b$

This Grammar is Ambiguous.

Remove useless Production

$S \rightarrow a / bXY$

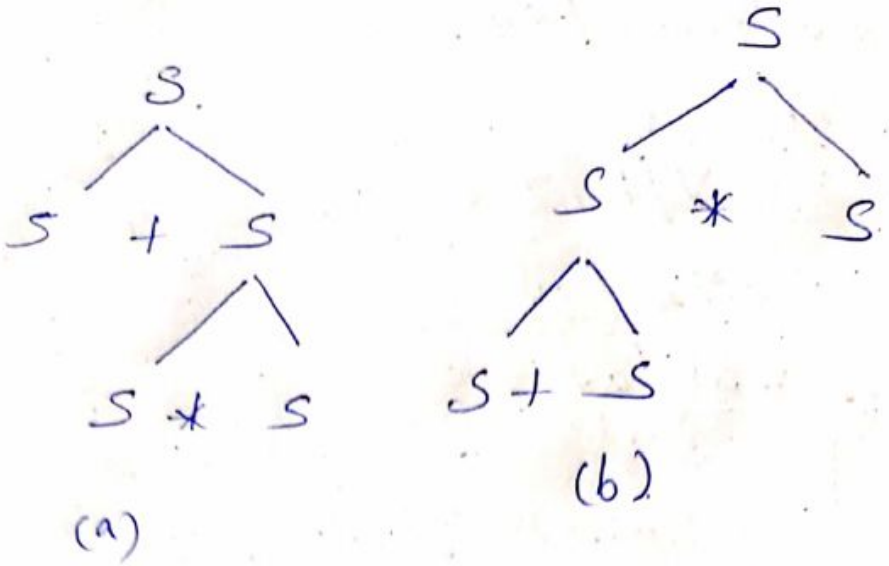
$A \rightarrow Bad / bSX / a$

$B \rightarrow aSB / bBX$ ✓

$X \rightarrow SBD / aBX / ad$

$Y \rightarrow SBa / XYb$ ✓

Scanned with CamScanner



Here we have two parse trees which are distinct

Derivation (1) says that the second and third expressions are multiplied and the result is added to the first expression.

Derivation (2) adds the first two expressions and multiplies the result by the third.

first derivation suggests that 1+2*3 should be grouped 1+(2*3) = 7, while the second derivation suggests the same expression should be grouped (1+2)*3 = 9

Scanned with CamScanner

Remove the Non-generating symbols

Here Nongenerating symbols are B and Y (Variables or Nonterminals)

Now in the production wherever we have B and Y we need to Remove them.

$S \rightarrow a / bXY$

$A \rightarrow Bad / bSX / a$

$X \rightarrow SBD / aBX / ad$.

$\downarrow$ further Reduced to

$S \rightarrow a$

$A \rightarrow bSX / a$

$X \rightarrow ad$

Now remove the unreachable ~~states~~ symbols from the start state.

Both A and X are unreachable

So $\boxed{S \rightarrow a}$

Scanned with CamScanner

(2)

$S \rightarrow az / sy / xA$

$X \rightarrow bsza$

$Y \rightarrow asy / byz$ ✓

$Z \rightarrow ayz / \underline{ad}$

$A \rightarrow \underline{ab} / aA$

Remove X and Y Non generating symbols.

$S \rightarrow az / sy / xA$

$\boxed{\begin{array}{l} X \rightarrow bsza \\ Y \rightarrow asy / byz \end{array}}$ X

$Z \rightarrow ayz / ad$

$A \rightarrow ab / aA$

↓

$S \rightarrow az$

$Z \rightarrow ad$

$A \rightarrow ab / aA$

Remove the unreachable symbols from start state.

A is unreachable so remove it.

$\boxed{\begin{array}{l} S \rightarrow az \\ Z \rightarrow ad \end{array}}$

Scanned with CamScanner

Eliminate Null production

(i) $S' \to S\#$

$S \to ABC$

$A \to a/bbD$

$B \to a/\epsilon$

$C \to b/\epsilon$

$D \to c/d$

Null Variables $\{C, B\}$

first eliminate $B \to \epsilon$

$S' \to S\#$

$S \to ABC/AC$

$A \to a/bbD$

$B \to a$

$C \to b/\epsilon$

$D \to c/d$

Removing $C \to \epsilon$

$S' \to S\#$

$S \to ABC/AC/AB/A$

$A \to a/bbD$

$B \to a$

$C \to b$

$D \to c/d$

(ii) $S \to aS/A$

$A \to \epsilon$

Null variable $\{A, S\}$

$S \to A$

$A \to \epsilon$ so $S \to \epsilon$

$S \to aS/a/\epsilon$

~~If~~ ϵ is part of language you cannot remove it.

a^n $n \geq 0$ is produced by the grammar.

a^* $n \geq 0$

a^+ $n > 0$

(iii) $S \to ABC$

$A \to aA/\epsilon$

$B \to bB/\epsilon$

$C \to c$

Null Variable $\{A, B\}$

$S \to ABC/BC/AC/C$

$A \to aA/a$

$B \to bB/b$

$C \to c$

Scanned with CamScanner

(2) $S \to ABC / cbB / Ba$

$A \to da / BC$

$B \to gc / \epsilon$

$C \to ha / \epsilon$

Removing $B \to \epsilon$

$S \to ABC / cbB / Ba / AC / cb / a$

$A \to da / BC / C$

$B \to gC$

$C \to ha / \epsilon$

Removing $C \to \epsilon$

$S \to ABC / cbB / Ba / AC / cb / a$

$AB / bB / A / b$

$A \to da / BC / C / B / \epsilon$

$B \to gC / g$

$C \to ha$

Removing $A \to \epsilon$

$S \to ABC / cbB / Ba / AC / cb / a$
$AB / bB / A / b / BC / C / B / \epsilon$

$A \to da / BC / C / B$

$B \to gC / g$

$C \to ha$

Scanned with CamScanner

Remove unit production $(\alpha \to \beta)$ $(\alpha, \beta \in V)$

1) $M \to N \mid QNnqQ$

$N \to nqN \mid n$

$Q \to qQ \mid \epsilon$

Here the unit productions are

$M \to N$

ie $A \to B$

$B \to C$

$A \to C$

$M \to nqN \mid n \mid QNnqQ$

$N \to nqN \mid n$

$Q \to qQ \mid \epsilon$

(2) $S \to Aa \mid B \mid c$

$B \to A \mid bb$

$A \to a \mid bc \mid B$

unit productions are

$S \to B$

~~$S \to A$~~

$B \to A$

$A \to B$

Now we can write it as.

$S \to Aa \mid c \mid bb \mid a \mid bc$

$B \to bb \mid a \mid bc$

$A \to a \mid bc \mid bb$

Scanned with CamScanner

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow c/b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Unit productions are $B \rightarrow C$
$C \rightarrow D$
$D \rightarrow E$

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a/b$

$C \rightarrow a$
$D \rightarrow a$
$E \rightarrow a$
} these are Not utilized so remove them

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow a/b$

Scanned with CamScanner

*) $S \to aA / B$

$A \to ba / bb$

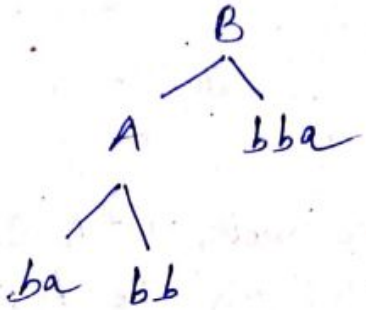
$B \to A / bba$

unit productions are

$S \to B$

$B \to A$

$S \to B \to A$



$S \to aA / ba / bb / bba$

$A \to ba / bb$

$B \to ba / bb / bba$

B is not reachable from the start state so remove it

$S \to aA / ba / bb / bba$

$A \to ba / bb$

Scanned with CamScanner

Chomsky Normal form:

every non empty CFL without ϵ has a grammar G in which all productions are in one of two simple forms either

(1) $A \rightarrow BC$, where A, B, C are variables.

(2) $A \rightarrow a$, where A is a variable and a is a terminal.

further G has no useless symbols, unit productions, ϵ-productions such a Grammar is said to be in Chomsky Normal form.

Steps to Convert CFG to CNF

Step 1: Eliminate start symbol from R.H.S and generate a new production

$$S' \rightarrow S$$

Step 2: Eliminate null and unit production

Step 3: Eliminate R.H.S with more than two non-terminals

Eg: Production rule $X \rightarrow XYZ$ can be written as

$$X \rightarrow PZ$$
$$P \rightarrow XY$$

Scanned with CamScanner

Let us take an example to Convert CFG to CNF

$S \rightarrow ASB$
$A \rightarrow aAS \mid a \mid \epsilon$
$B \rightarrow SbS \mid A \mid bb$

Step 1: Eliminate S in R.H.S

$S' \rightarrow S$
$S \rightarrow ASB$
$A \rightarrow aAS \mid a \mid \epsilon$
$B \rightarrow SbS \mid A \mid bb$

Step 2: Remove 'ϵ' production

use where is ϵ in the above production $A \rightarrow \epsilon$. So substitute ϵ where every 'A' is there in the production.

$S' \rightarrow S$
$S \rightarrow ASB \mid SB$
$A \rightarrow aAS \mid aS \mid a$
$B \rightarrow SbS \mid A \mid \epsilon \mid bb$

Now we have $B \rightarrow \epsilon$ remove that

$S' \rightarrow S$
$S \rightarrow ASB \mid AS \mid SB \mid S$
$A \rightarrow aAS \mid aS \mid a$
$B \rightarrow SbS \mid A \mid bb$

Remove unit productions $S' \rightarrow S$

$S' \rightarrow ASB \mid AS \mid SB \mid S$
$S \rightarrow ASB \mid AS \mid SB \mid S$
$A \rightarrow aAS \mid aS \mid a$
$B \rightarrow SbS \mid A \mid bb$

Still we have $B \rightarrow A$, $S \rightarrow S$, $S' \rightarrow S$

Scanned with CamScanner

$S' \rightarrow ASB \mid AS \mid SB$

$S \rightarrow ASB \mid AS \mid SB$

$A \rightarrow aAS \mid as \mid a$

$B \rightarrow SbS \mid bb \mid aAS \mid as \mid a$

Step 3: Eliminate R.H.S with more than two non terminals

$S' \rightarrow ASB$. $S \rightarrow ASB$

Now $P \rightarrow AS$

Altering remaining productions we get:

$S' \rightarrow PB \mid AS \mid SB$

$S \rightarrow PB \mid AS \mid SB$

$A \rightarrow aAS \mid aS \mid a$

$B \rightarrow SbS \mid bb \mid aAS \mid aS \mid a$

Still we have

$A \rightarrow aAS \mid as$

$B \rightarrow SbS \mid bb \mid aAS \mid as$

Now $Q \rightarrow b$, since $P \rightarrow AS$

$A \rightarrow aP \mid as$

$B \rightarrow SQS \mid QQ \mid aP \mid as$

Now $R \rightarrow a$

$A \rightarrow RP \mid RS$

$B \rightarrow SQS \mid QQ \mid RP \mid RS$

Now $T \rightarrow SQ$

Scanned with CamScanner

final Production

$S' \rightarrow PB \mid AS \mid SB$

$S \rightarrow PB \mid AS \mid SB$

$A \rightarrow RP \mid RS \mid a$

$B \rightarrow TS \mid RP \mid RS \mid QS \mid a$

$P \rightarrow AS$

$T \rightarrow SQ$

$Q \rightarrow b$

$R \rightarrow a$

Scanned with CamScanner

Greibach Normal form

A CFG is in Greibach Normal form if the productions are in the following forms

$$A \rightarrow b$$
$$A \rightarrow b C_1 C_2 \cdots C_n$$

where $A, C_1, \ldots C_n$ are Non terminals and b is a terminal.

Steps to convert CFG to GNF

Step1: check if the given CFG has any unit productions or Null productions and Remove if there are any.

Step2: check whether the CFG is already in CNF and Convert it to CNF if it is not

Step3: change the names of the Non terminal symbols into A_i in ascending order of i.

$S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$.

Replace S with A_1
C with A_2
A with A_3
B with A_4

we get

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid A_1 A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Scanned with CamScanner

Step 4: Alter the rules so that the Non-terminals are in ascending order, such that, if the production is of the form $A_i \rightarrow A_j X$, then $i < j$ and should never be $i \geq j$

$A_4 \rightarrow b \mid A_1 A_4$ $\qquad$ $A_1 \rightarrow A_2 A_3 \mid A_4 A_4$

$A_4 \rightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$ Replace $A_2 \rightarrow b$

$A_4 \rightarrow \underbrace{b \mid bA_3 A_4}_{\text{GNF}} \mid \underbrace{A_4 A_4 A_4}_{\text{Left Recursion}}$

Step 5: Remove Left Recursion.

$A_1 \rightarrow A_2 A_3 \mid A_4 A_4.$

$A_4 \rightarrow b \mid bA_3 A_4 \mid \underbrace{A_4 A_4 A_4}_{\text{left Recursion.}}$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Introduce a new variable to remove the left Recursion

$A_4 \rightarrow b \mid bA_3 A_4 \mid A_4 A_4 A_4$

$Z \rightarrow A_4 A_4 Z \mid A_4 A_4$

$A_4 \rightarrow b \mid bA_3 A_4 \mid bZ \mid bA_3 A_4 Z$ } this is in GNF

Scanned with CamScanner

Now the grammar is

$A_1 \rightarrow A_2A_3 \mid A_4A_4$

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow A_4A_4 \mid A_4A_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

$A_1 \rightarrow bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

$A_4 \rightarrow b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \rightarrow bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$

$\mid bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Scanned with CamScanner

① Convert the following CFG to CNF

$P: S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$.

② Convert the following CFG to GNF

$S \to AB$

$A \to BS \mid a$

$B \to SA \mid b$

① Since S appears in RHS, we add a new state S' and $S' \to S$ is added to the production.

$P: S' \to S$, $S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \epsilon$

② Remove the Null productions $B \to \epsilon$ and $A \to \epsilon$.

After Removing $B \to \epsilon$: $P: S' \to S$, $S \to ASA \mid aB \mid a$, $A \to B \mid S \mid \epsilon$
$B \to b$

After Removing $A \to \epsilon$: $P: S' \to S$, $S \to ASA \mid aB \mid a \mid AS \mid SA \mid S$,
$A \to B \mid S$, $B \to b$

③ Remove the unit productions: $S \to S$, $S' \to S$, $A \to B$ and $A \to S$.

Scanned with CamScanner

After Removing $S \to S$:

P: $S' \to S$, $S \to ASA|aB|a|AS|SA$, $A \to B|S$, $B \to b$

After Removing $S' \to S$:

P: $S' \to ASA|aB|a|AS|SA$

$S \to ASA|aB|a|AS|SA$

$A \to B|S$, $B \to b$

After Removing $A \to B$:

P: $S' \to ASA|aB|a|AS|SA$,

$S \to ASA|aB|a|AS|SA$,

$A \to b|S$; $B \to b$

After Removing $A \to S$:

P: $S' \to ASA|aB|a|AS|SA$,

$S \to ASA|aB|a|AS|SA$

$A \to b|ASA|aB|a|AS|SA$,

$B \to b$

(4) Now find out the productions that has more than Two variables in RHS

$S' \to ASA$, $S \to ASA$ and $A \to ASA$.

After removing these, we get : P: $S' \to AX|aB|a|AS|SA$

$S \to AX|aB|a|AS|SA$

$A \to b|AX|aB|a|AS|SA$

$B \to b$

$X \to SA$

Scanned with CamScanner

Now change the productions $S' \to aB$, $S \to aB$ and $A \to aB$

Finally we get : P: $S' \to AX \mid YB \mid a \mid AS \mid SA$

$S \to AX \mid YB \mid a \mid AS \mid SA$

$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \to b$

$X \to SA$

$Y \to a$

Scanned with CamScanner

Eg:

$S \to AB$

$A \to BS / a$

$B \to SA / b$

It is already in CNF

So first rename all the variables

$[S \to A_1]$ $[A \to A_2]$ $[B \to A_3]$

1) $A_1 \to A_2 A_3$

$A_2 \to A_3 A_1 / a$

$A_3 \to A_1 A_2 / b$

2) $A_i \to A_j V$ where $J > i$ (or) $A_i \to a\gamma$

$A_3 \to \underline{A_1} A_2 / b$

Substitute

$A_3 \to A_2 A_3 A_2 / b$

$A_3 \to a A_3 A_2 / \underbrace{A_3 A_1 A_3 A_2}_{\text{left recursion}} / b$

Now we have left Recursion

$A_3 \to A_3 A_1 A_3 A_2$ we should Remove this by introducing new variable

Now the production will become.

$A_3 \to a A_3 A_2 B_3 / b B_3 / a A_3 A_2 / b$

$B_3 \to A_1 A_3 A_2 B_3 / A_1 A_3 A_2$

So we will see which productions are in GNF & which are not in GNF.

Scanned with CamScanner

A_1, A_2, B_3 are not in GNF. So we will start from A_2 Production

$$A_2 \rightarrow A_3 A_1$$

$$A_2 \rightarrow a A_3 A_2 B_3 A_1 / bB_3 A_1 / a A_3 A_2 A_1 / bA_1 / a$$

$$A_1 \rightarrow A_2 A_3$$

$$A_1 \rightarrow a A_3 A_2 B_3 A_1 A_3 / bB_3 A_1 A_3 / a A_3 A_2 A_1 A_3 / bA_1 A_3 / a A_3$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3 / A_1 A_3 A_2$$

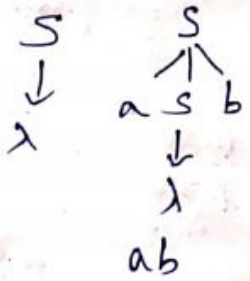
Substitute A_1 Production with these two.

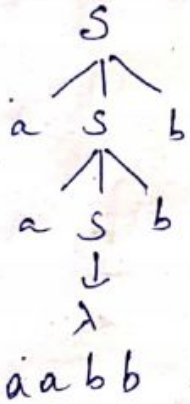
Scanned with CamScanner

Convert Context free language to Context free grammar

① $a^n b^n$, $n \geq 0$

$S \rightarrow aSb \mid \lambda$





If $n \geq 1$ then $S \rightarrow aSb \mid ab$

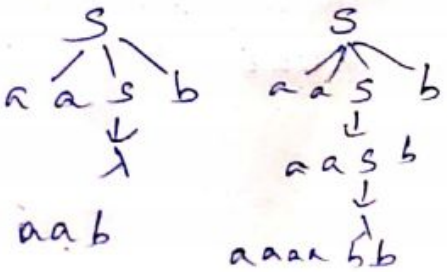
② $a^n b^{n+2}$, $n \geq 0$

$S \rightarrow aSb \mid bb$

③ $a^{2n} b^n$ $n \geq 0$

If $n \geq 1$

$S \rightarrow aaSb \mid \lambda$



Scanned with CamScanner

(4) $a^{2n+3} b^n, n \geq 0$

$S \to aaSb \mid aaa$

(5) $a^m b^n$ $m > n$
$n \geq 0$

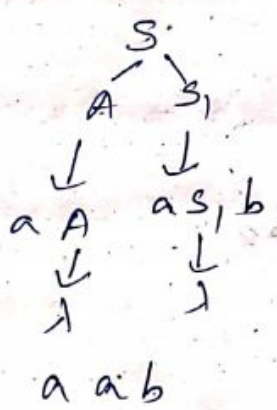
$S \to AS_1$

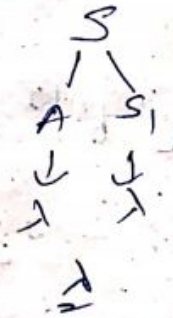
$S_1 \to aS_1b \mid \lambda$

$A \to aA \mid a$

$m \geq n$

$A \to aA \mid \lambda$

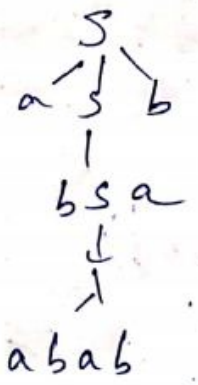


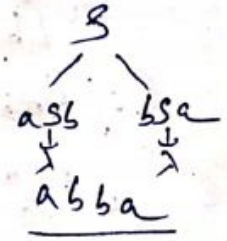


(6) $\{w \mid n_a(w) = n_b(w)\}$

$S \to aSb \mid bSa \mid \lambda$

$S \to aSb \mid bSa \mid SS \mid \lambda$





Scanned with CamScanner

(7) $ww^R \cup w(a+b)w^R$.

$S \to aSa \mid bSb \mid \lambda$ even palindrome

S
aSa
bSb
λ
$abba$

$S \to aSa \mid bSb \mid a \mid b \mid \lambda$

(8) $a^m b^m c^n$, $m, n \geq 0$.

$S \to S_1 C$

$S_1 \to aS_1 b \mid \lambda$

$C \to cC \mid \lambda$

Scanned with CamScanner

Pumping Lemma

Pumping lemma (for CFL) is used to prove that a language is Not Context free.

If A is a Context free language, then A has a pumping length 'p' such that any string 'S', where $|S| \geq p$ may be divided into 5 pieces $S = uvxyz$ such that the following conditions must be true:

① uv^ixy^iz is in A for every $i \geq 0$

② $|vy| > 0$

③ $|vxy| \leq p$

To prove that a Language is Not Context free using pumping Lemma (for CFL) follow the steps given below.

- Assume that A is Context free
- It has to have a pumping length (say p)
- All strings longer than p can be pumped $|S| \geq p$
- Now find a string 'S' in A such that $|S| \geq p$
- Divide S into $uvwxyz$
- Show that $uv^ixy^iz \notin A$ for some i.
- Then consider the ways that S can be divided into $uvxyz$.
- Show that none of these can satisfy all the 3 pumping conditions at the same time
- S cannot be pumped == Contradiction.

Scanned with CamScanner

Show that $L = \{a^N b^N c^N \mid N \geq 0\}$ is Not Context free

- Assume that L is context free
- L must have a pumping length P
- Now we take a string S such that $S = a^P b^P c^P$
- we divide S into parts uvxyz.

Eg $P = 4$ so $S = a^4 b^4 c^4$

Case I : v and y each contain only one type of symbol

$$\underbrace{aa}_{u}\underbrace{a}_{v}\underbrace{abbbb}_{x}\underbrace{c}_{y}\underbrace{ccc}_{z}$$

If $i=2$ $uv^i xy^i z$

$uv^2 xy^2 z$

aaaaa abbbbc cccc

$a^6 b^4 c^5 \notin L$

Hence it is not a context free

Case II : Either v or y has more than one kind of symbols

$$\underbrace{aa}_{u}\underbrace{aa\,bb}_{v}\underbrace{b}_{x}\underbrace{b}_{y}\underbrace{cccc}_{z}$$

$uv^i xy^i z$ $(i=2)$ $uv^2 xy^2 z$

aa aabbaabb bbbcccc

$a^4 b^2 a^2 b^5 c^4 \notin L$

Hence it is not a context free language.

Scanned with CamScanner

Show that $L = \{ww \mid w \in \{0,1\}^*\}$ is Not Context free

- Assume that L is context free.
- L must have a pumping length p
- Now we take a string s such that $s = 0^p 1^p 0^p 1^p$
- we divide s into parts uvxyz

Eg: $p=5$. So $s = 0^5 1^5 0^5 1^5$

Case 1: vxy does not straddle a boundary

$$\underbrace{00000}_{u}\underbrace{1111}_{vxy}\underbrace{1\,00000\,1111}_{z}$$

$uv^i xy^i z$

uv^2xy^2z

00000111111000001111

$0^5 1^7 0^5 1^5 \notin L$

Hence it is not context free

Case 2a: vxy straddle the first boundary.

$$\underbrace{0000}_{u}\underbrace{0\,1111}_{v\,xy}\underbrace{1\,00000\,1111}_{z} \qquad uv^2xy^2z$$

00000001111111000001111

$\underbrace{0^7 1^7}\underbrace{0^5 1^5} \notin L$

Scanned with CamScanner

Case 2b: vxy straddles the third boundary

$$\underbrace{00000'11111'000}_{u}\,\underbrace{0'0'1111}_{vxy}\,\underbrace{1}_{z}$$

uv^2xy^2z

0000011111000 0000 11111 11

$0^5 1^5 0^7 1^7 \notin L$

Case 3: vxy straddles the midpoint

$$\underbrace{00000'11111}_{u}\,\underbrace{'00}_{vxy}\,\underbrace{0000'11111}_{z}$$

uv^2xy^2z

00000111111 0000 0001111 1

$0^5 1^7 0^7 1^5 \notin L$

Scanned with CamScanner

Closure Properties of Context free Languages

Substitution

Let Σ be an alphabet and suppose that for every symbol a in Σ, we choose a Language L_a. These chosen Languages can be over any alphabets, not necessarily Σ and This choice of Languages defines a function s (a substitution) on Σ, and we shall refer to L_a as $s(a)$ for each symbol a.

If $w = a_1 a_2 \cdots a_n$ is a string in Σ^*, then $s(w)$ is the language of all strings $x_1 x_2 \cdots x_n$ such that string x_i is in the language $s(a_i)$, for $i = 1, 2, \ldots n$. put another way, $s(w)$ is the Concatenation of the Languages $s(a_1) s(a_2) \cdots s(a_n)$

we can further extend the definition of s to apply to languages: $s(L)$ is the union of $s(w)$ for all strings with L.

example: Suppose $s(0) = \{a^n b^n \mid n \geq 1\}$ and $s(1) = \{aa, bb\}$. That is, s is a substitution on alphabet $\Sigma = \{0, 1\}$. Language $s(0)$ is the set of strings with one or more a's followed by an equal number of b's, while $s(1)$ is the finite language consisting of the two strings aa and bb.

Let $w = 01$. Then $s(w)$ is the Concatenation of the Languages $s(0)s(1)$. To be exact, $s(w)$ Consists of all strings of the forms $a^n b^n aa$ and $a^n b^{n+2}$, where $n \geq 1$

Now, Suppose $L = L(0^*)$, that is, the set of all strings of 0's. Then $s(L) = (s(0))^*$. This Language is the set of all strings of the form

$$a^{n_1} b^{n_1} a^{n_2} b^{n_2} \cdots a^{n_k} b^{n_k} \quad \text{for some } k \geq 0$$

and any sequence of choices of positive integers $n_1, n_2, \ldots n_k$.

Scanned with CamScanner

If L is a context free Language over alphabet Σ and S is a substitution on Σ such that $S(a)$ is a CFL for each a in Σ, then $S(L)$ is a CFL.

Scanned with CamScanner

It includes strings such as ϵ, aabbaaabbb, and abaabbabab.

Applications of the substitution Theorem.

The context free languages are closed under the following operations.

(1) union (2) Concatenation (3) Closure (*) and positive Closure (+)

(4) Homomorphism.

(1) Union: Let L_1 and L_2 be CFL's. Then $L_1 \cup L_2$ is the language $S(L)$, where L is the language $\{1,2\}$ and S is the substitution defined by $S(1) = L_1$ and $S(2) = L_2$.

(2) Concatenation: Again let L_1 and L_2 be CFL's. Then L_1L_2 is the language $S(L)$, where L is the language $\{12\}$ and S is the same substitution as in case (1).

(3) Closure and positive closure: If L_1 is a CFL, L is the language $\{1\}^*$, and S is the substitution $S(1) = L_1$, then $L_1^* = S(L)$. Similarly, if L is instead the language $\{1\}^+$, then $L_1^+ = S(L)$.

(4) Suppose L is a CFL over alphabet Σ, and h is a homomorphism on Σ. Let S be the substitution that replaces each symbol a in Σ by the language consisting of the one string that is $h(a)$. That is, $S(a) = \{h(a)\}$, for all a in Σ. Then $h(L) = S(L)$.

Scanned with CamScanner

Reversal

The CFL's are also closed under reversal.

If L is a CFL, then so is L^R.

Intersection with a Regular Language

The CFL's are not closed under intersection

example: $L = \{0^n 1^n 2^n \mid n \geq 1\}$

is not a context free language. However, the following two languages are context free

$$L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$$

A grammar for L_1 is:

$$S \rightarrow AB$$
$$A \rightarrow 0A1 \mid 01$$
$$B \rightarrow 2B \mid 2$$

In this grammar, A generates all strings of the form $0^n 1^n$ and B generates all strings of 2's. A grammar for L_2 is:

$$S \rightarrow AB$$
$$A \rightarrow 0A \mid 0$$
$$B \rightarrow 1B2 \mid 12$$

It works similarly, but with A generating any string of 0's and B generating matching strings of 1's and 2's.

Scanned with CamScanner

However, $L = L_1 \cap L_2$. To observe that L_1 requires that there be the same number of 0's and 1's, while L_2 requires the number of 1's and 2's to be equal.

A string in both languages must have equal numbers of all three symbols and thus be in L.

Since the languages are not closed under intersection we conclude they are not context free.

The context free languages are closed under the operation of "intersection with a regular language".

If L is a CFL and R is a regular language, then $L \cap R$ is a CFL.

<u>Inverse Homomorphism</u>

If L is a CFL and h is a homomorphism.

Then $h^{-1}(L)$ is a CFL.

Scanned with CamScanner

Push Down Automata

The Context-free languages have a type of automaton that defines them. This automaton, called a pushdown automaton PDA is essentially finite automata with the addition of stack. The stack can be read, pushed and popped only at the top, just like the 'stack' data structure.

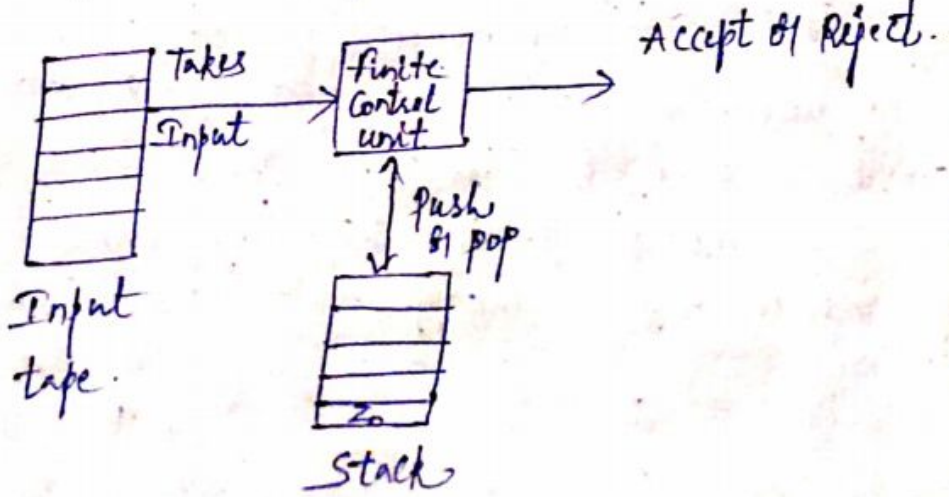
A 'PDA' is a way to implement a context free grammar in a similar way we design finite Automata for Regular Grammar.

- It is more powerful than FSM (finite statemachine)
- FSM has a very limited memory but PDA has more memory.

PDA = FSM + stack.

A push Down Automata has 3 components

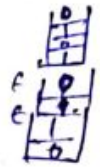
1. An input tape
2. A finite Control unit
3. A Stack with infinite size



Scanned with CamScanner

Formal Definition of Push Down Automata

Push Down Automaton (PDA) involves seven components



PDA → P

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings.

Σ: A finite set of input symbols, also analogous to the corresponding component of a finite automaton.

Q: A finite set of states, like the states of a finite Automaton.

Γ: A finite stack alphabet. This component, which has no finite-automaton analog is the set of symbols that we are allowed to push onto the stack.

δ: The transition function. Formally, δ takes as argument a triple $\delta(q, a, X)$, where:

- q is a state in Q
- a is either an input symbol in Σ or $a = \epsilon$, the empty string, which is assumed not to be an input symbol.
- X is a stack symbol, that is, a member of Γ

The output of δ is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols that replaces X at the top of the stack. For instance, if $\gamma = \epsilon$, then the stack is popped, if $\gamma = X$, then the stack is unchanged and if $\gamma = YZ$, then X is replaced by Z, and Y is pushed onto the stack.

Scanned with CamScanner

q_0: The start state. The PDA is in this state before making any transitions.

Z_0: The start symbol. Initially, the PDA's stack consists of one instance of this symbol and nothing else.

F: The set of accepting states or final states.

Design PDA to accept the Language

$$L_{wwr} = \{ ww^R \mid w \in (0+1)^* \}$$

$$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where δ is defined by the following rules.

① $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ and $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$

one of these rules applies initially, when we are in state q_0 and we see the start symbol Z_0 at the top of the stack. We read the first input and push it onto the stack, leaving Z_0 below to mark the bottom.

② $\delta(q_0, 0, 0) = \{(q_0, 00)\}$, $\delta(q_0, 0, 1) = \{(q_0, 01)\}$
$\delta(q_0, 1, 0) = \{(q_0, 10)\}$; and $\delta(q_0, 1, 1) = \{(q_0, 11)\}$

These four, similar rules allow us to stay in state q_0 and read inputs, pushing each onto the top of the stack and leaving the previous top stack symbol alone.

Scanned with CamScanner

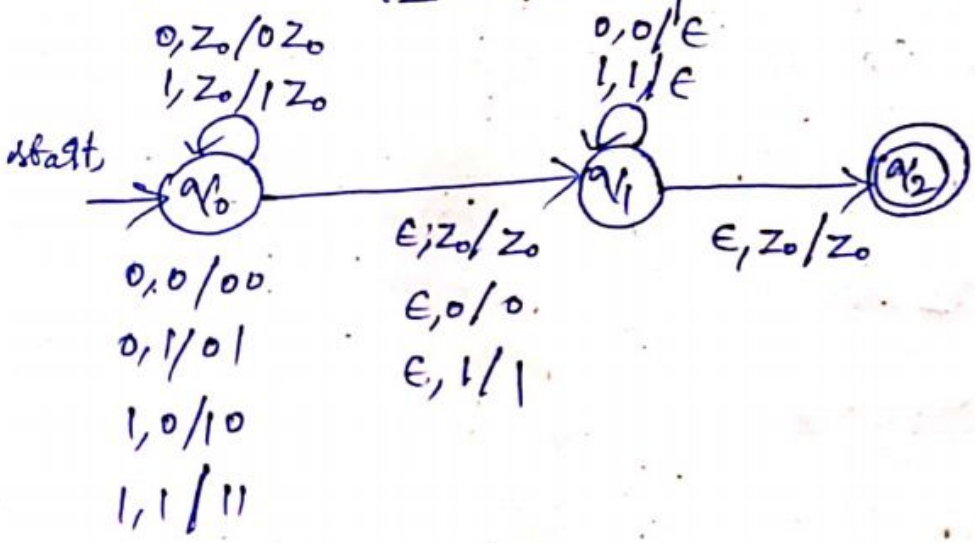
③ $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$, $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$ and $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$

These three rules allow p to go from state q_0 to state q_1 spontaneously (on ϵ input), leaving intact whatever symbol is at the top of the stack.

④ $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$ and $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$.

Now, in state q_1 we can match input symbols against the top symbols on the stack and pop when the symbols match.

5. $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$. Finally, if we expose the bottom of stack marker Z_0 and we are in state q_1, then we have found an input of the form ww^R. We go to state q_2 and accept.



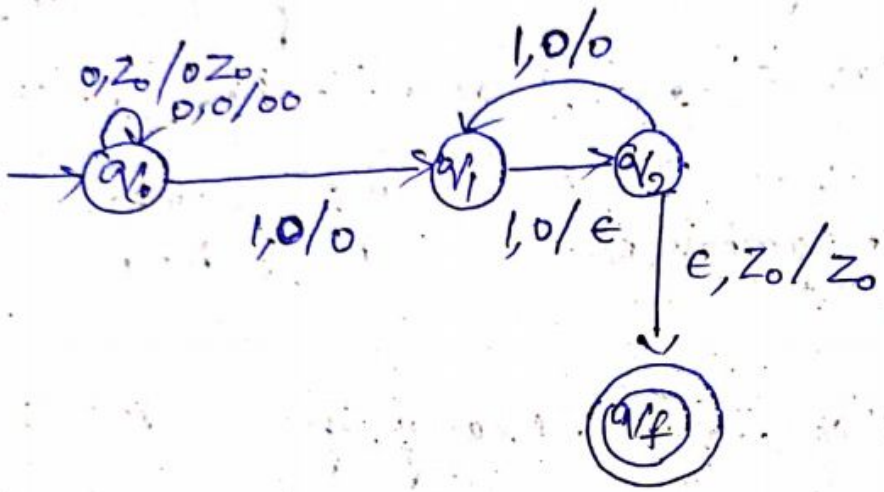
Scanned with CamScanner

PDA for $0^n 1^{2n}$ CFL language

$L = \{0^n 1^{2n}, n>0\}$

011, 001111, 000111111

0	0	1	1	1	1	ϵ



1) $q_0, 0, Z_0 \rightarrow 0Z_0 \; (q_0, 0Z_0)$

2) $q_0, 0, 0 \rightarrow 00 \; (q_0, 00)$

3) $q_0, 1, 0 \rightarrow (q_1, 0)$

4) $q_1, 1, 0 \rightarrow (q_2, \epsilon)$

5) $q_2, 1, 0 \rightarrow (q_1, 0)$

6) $q_2, \epsilon, Z_0 \rightarrow (q_f, Z_0)$

Scanned with CamScanner

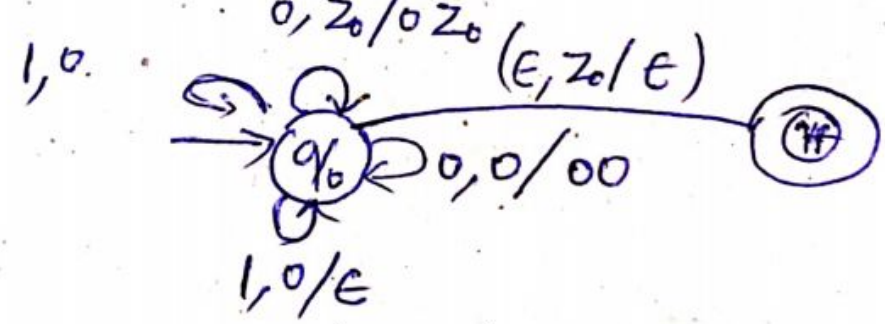
$L = \{ w \in (0,1)^* \mid n_0(w) = n_1(w) \}$

$\delta = Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

0	1	0	0	1	1

01, 10, 00, 11, 0011

1100, 0101, 1010



Scanned with CamScanner

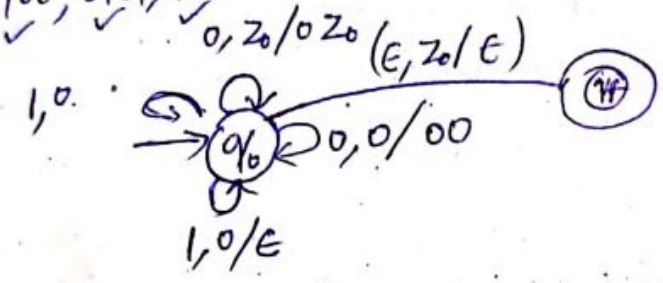
$L = \{ w \in (0,1)^* \mid n_0(w) = n_1(w) \}$

$\delta = Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$

0	1	0	0	1	1

01, 10, 00, 11, 0011

1100, 0101, 1010



The Languages of a PDA

PDA accepts its input by consuming it and entering an accepting state. We call this approach "acceptance by final state".

There is a second approach to defining the language of a PDA that has important applications. We may also define for any PDA the language "accepted by empty stack". That is, the set of strings that cause the PDA to empty its stack, starting from the initial ID.

Scanned with CamScanner

Acceptance by final state

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $L(P)$, the language accepted by P by final state is.

$$\{ w \mid (q_0, w, Z_0) \overset{*}{\vdash}_P (q, \epsilon, \alpha) \}$$

for some state q in F and any stack string α.

That is, starting in the initial ID with 'w' waiting on the input, P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

Acceptance by Empty stack

for each PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ we also define

$$N(P) = \{ w \mid (q_0, w_0, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon) \}$$

for any state q. That is, $N(P)$ is the set of inputs w that P can consume and at the same time empty its stack.

Example: The PDA P for language L_{ww^r} never empties its stack; so $N(P) = \phi$. However, a small modification will allow P to accept L_{ww^r} by empty stack as well as by final state.

Scanned with CamScanner

Instead of the transition $\delta(q_1, \epsilon, Z_0) = \{q_2, Z_0\}$ use $\delta(q_1, \epsilon, Z_0) = \{(q_2, \epsilon)\}$. Now, P pops the last symbol off its stack as it accepts and $L(P) = N(P) = L_{wsr}$

We can write P as a six-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ when the language is accepted by empty stack

Equivalence of PDA's and CFG's

We shall demonstrate that the languages defined by PDA's are exactly the Context free languages.

The goal is to prove that the following 3 classes of languages

1. The Context-free languages ie the languages defined by CFG's
2. The languages that are accepted by final state by some PDA
3. The languages that are accepted by empty stack by some PDA.

Conversion from grammar to pushdown Automata

Let $G = (V, T, Q, S)$ be a CFG. Construct PDA P that accepts $L(G)$ by empty stack as follows

$$P = (\{q\}, T, V \cup T, \delta, q, S)$$

where transition function δ is defined by:

1. For each variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \text{ is a production of } P\}$$

2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$.

Scanned with CamScanner

Example: Let us convert the expression grammar given below ~~Fig 5.1~~ to a PDA.

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$
$$E \rightarrow I \mid E*E \mid E+E \mid (E)$$

The set of terminals for the PDA is $\{a, b, 0, 1, (,), +, *\}$. These eight symbols and the symbols I and E form the stack alphabet. The transition function for the PDA is

a) $\delta(q, \epsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$

b) $\delta(q, \epsilon, E) = \{(q, I), (q, E+E), (q, E*E), (q, (E))\}$

c) $\delta(q, a, a) = \{(q, \epsilon)\}$

$\delta(q, b, b) = \{(q, \epsilon)\}$

$\delta(q, 0, 0) = \{(q, \epsilon)\}$

$\delta(q, 1, 1) = \{(q, \epsilon)\}$

$\delta(q, (, ()) = \{(q, \epsilon)\}$

$\delta(q,),)) = \{(q, \epsilon)\}$

$\delta(q, +, +) = \{(q, \epsilon)\}$

$\delta(q, *, *) = \{(q, \epsilon)\}$

Note that (a) and (b) come from rule (1) while the eight transitions of (c) come from rules

Scanned with CamScanner

Conversion from PDA to CFG

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ be a PDA. Then there is a context free grammar G such that $L(G) = N(P)$

we shall construct $G = (V, \Sigma, R, S)$ where the set of variables V consists of

1) The special symbol S, which is the start symbol.
2) All symbols of the form $[pXq]$, where p and q are states in Q, and X is a stack symbol in Γ.

The productions of G are as follows.

a) for all states p, G has the production $S \to [q_0 Z_0 p]$.
a symbol like $[q_0 Z_0 p]$ is intended to generate all those strings w that cause p to pop Z_0 from its stack while going from state q_0 to state p. That is $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$
If so, then these productions say that start symbol S will generate all strings w that cause p to empty its stack

b) Let $\delta(q, a, X)$ contain the pair $(r, Y_1 Y_2 \cdots Y_K)$ where
1) a is either a symbol in Σ or $a = \epsilon$
2) K can be any number, including 0, in which case the pair is (r, ϵ).

Then for all lists of states $r_1, r_2, \ldots r_K$, G has the production

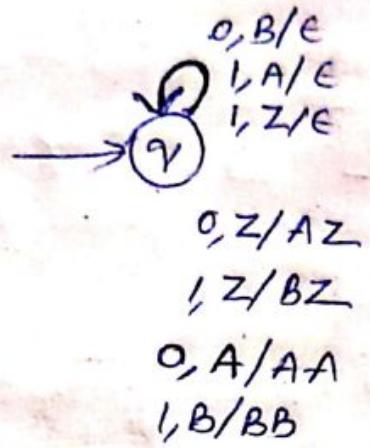
$$[qXr_K] \to a[rY_1r_1][r_1Y_2r_2]\ldots[r_{K-1}Y_Kr_K]$$

This production says that one way to pop X and go from state q_0 to state r_K is to read a (which may be ϵ) then use some input to pop Y_1 off the stack while going from state r_1, then read some more input that pops Y_2 off the stack and goes from state r_1 to r_2 and so on

Scanned with CamScanner

Design a PDA which processes strings of 0's and 1's and accepts if the number of 0's is equal to the number of 1's.

Here we use two stack symbols 'A' & 'B'



$P = (\{q\}, \{0,1\}, \{Z, A, B\}, \delta_N, q, Z)$

δ_N is defined by

1) $\delta_N(q, 0, Z) = (q, AZ)$
2) $\delta_N(q, 1, Z) = (q, BZ)$
3) $\delta_N(q, 0, A) = (q, AA)$
4) $\delta_N(q, 0, B) = (q, \epsilon)$
5) $\delta_N(q, 1, A) = (q, \epsilon)$
6) $\delta_N(q, 1, B) = (q, BB)$
7) $\delta_N(q, \epsilon, Z) = (q, \epsilon)$

Scanned with CamScanner

Now let us Convert the PDA to CFG.

$$P_N = (\{q\}, \{0,1\}, \{Z, A, B\}, \delta_N, q, Z)$$

a) S, the start symbol

b) $[qZq]$

c) $[qAq]$

d) $[qBq]$

The productions of grammar G is as follows.

① The only production for $S \rightarrow [qZq]$

If there are n states of the PDA, then there would be n productions of this type

② $[qZq] \rightarrow 0\,[qAq][qZq]$ results from $\delta(q,0,Z) \rightarrow \{(q,AZ)\}$

③ $[qZq] \rightarrow 1\,[qBq][qZq]$

④ $[qAq] \rightarrow 0\,[qAq][qAq]$

⑤ $[qBq] \rightarrow 1\,[qBq][qBq]$

⑥ $[qBq] \rightarrow 0$ results from $\delta_N(q,0,B) \rightarrow (q,\epsilon)$

⑦ $[qAq] \rightarrow 1$

⑧ $[qZq] \rightarrow \epsilon$

Scanned with CamScanner

We may for convenience replace the triple $[q_1 Z q_1]$ with X

$[q_1 A q_1]$ with A

$[q_1 B q_1]$ with B.

$S \rightarrow X$

$X \rightarrow 0AX$

$X \rightarrow 1BX$

$A \rightarrow 0AA$

$B \rightarrow 1BB$

$A \rightarrow 1$

$B \rightarrow 0$

$X \rightarrow \epsilon$

Scanned with CamScanner

Introduction to Turing Machine

There are specific problems we cannot solve using a computer. These problems are called "undecidable". Here comes the concept of Turing machine: Hence we use Turing machine to develop a theory of undecidable problems i.e. problems that no computer can solve.

Decidable and undecidable Problems in TOC

A problem is said to be Decidable if we can always construct a corresponding algorithm that can answer the problem correctly. If we want to compute all the prime numbers in the range 1000 to 2000. To find the solution of this problem, we can easily devise an algorithm that can enumerate all the prime numbers in this range.

Now talking about Decidability in terms of a Turing machine, a problem is said to be a Decidable Problem if there exists a corresponding Turing machine which halts on every input with an answer yes or no. These problems are termed as Turing Decidable since a Turing machine always halts on every input, accepting or rejecting it.

Scanned with CamScanner

Semi Decidable problems

For these types of problem turing machine halts on the input accepted by it. but it can either halt or loop forever on the input which is rejected by the T.M. Such problems are termed as Turing Recognisable.

undecidable problems

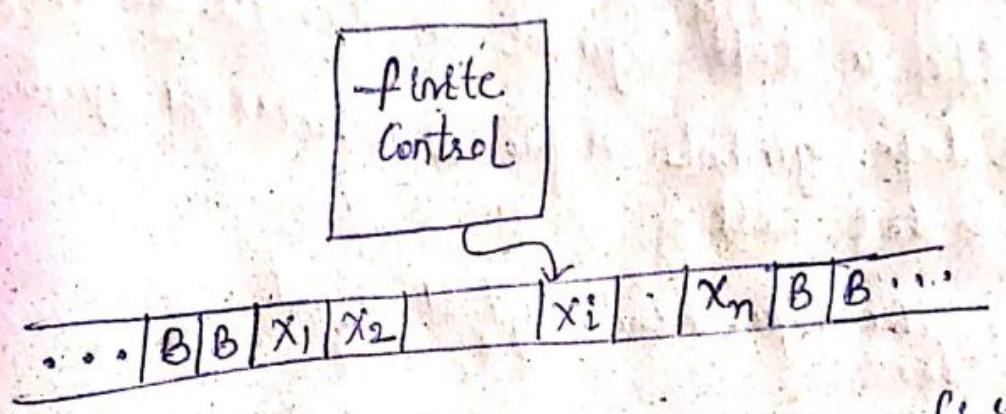
The problems for which we can't construct an algorithm that can answer the problem correctly infinite time are termed as undecidable problem. These problems may be partially decidable but they will never be decidable. That is there will always be a condition that will lead the T.M into an infinite loop without providing an answer at all.

Example: $a^n + b^n = c^n$ for any $n > 2$

Scanned with CamScanner

Notation for the Turing machine

The Turing Machine Consists of a finite Control which Can be in any of a finite set of states. There is a tape divided into Square or Cells, each cell can hold any one of a finite number of symbols



Initially, the input, which is a finite-length string of symbols chosen from the input alphabet, is placed on the tape. All other tape cells, extending infinitely to the left and right, initially hold a special symbol called the blank. The blank is a tape symbol, but not an input symbol and there may be other tape symbols besides the input symbols and the blank, as well

There is a tape head that is always positioned at one of the tape cells

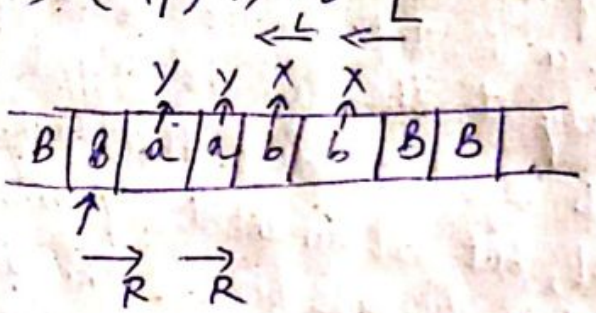
Scanned with CamScanner

A move of the Turing machine is a function of the state of the finite control and the tape symbol scanned. In one move, the Turing machine will

1) Change state. The next state optionally may be the same as the current state.
2) write a tape symbol in the cell scanned. This tape symbol replaces whatever symbol was in that cell.
3) Move the tape head left or right.

$$\delta : Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$$

$$\delta(q_0, a) \rightarrow (q_1, Y, R)$$



Turing machine is similar to that used for finite automata or PDA's. we describe a T.M by the 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

whose components have the following meanings.

Q : The finite set of states of the finite control

Σ : The finite set of input symbols

Scanned with CamScanner

Γ: The Complete set of tape symbols.

δ: The transition function. The arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$ if it is defined, is a triple (p, Y, D) where

1) p is the next state in Q.
2) Y is the symbol in Γ, written in the cell being scanned, replacing whatever symbol was there.
3) D is the direction, either L or R, standing for left or right, respectively and telling us the direction in which the head moves.

q_0: The start state, a member of Q, in which the finite control is found initially.

B: The blank symbol. This symbol is in Γ but not in Σ, ie., it is not an input symbol. The blank appears initially in all but the finite number of initial cells that hold input symbols.

F: The set of final or accepting states, a subset of Q.

Scanned with CamScanner

$a^n b^n \rightarrow PDA$

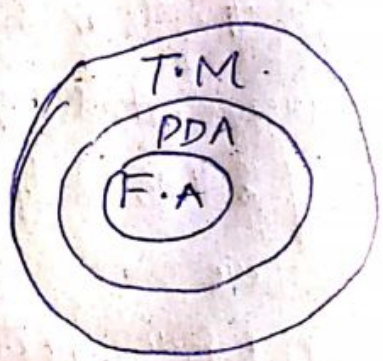
$a^n b^n c^n \rightarrow T.M$

B	a	b	B

Rw head

1) Read, write
2) Left, Right

Finite control



$\delta = Q \times \Gamma \rightarrow Q \times \Gamma \times (L, R)$

$(q_0, a) \longrightarrow (q_1, X, R)$

for Non deterministic.

$2^{Q \times \Gamma \times (L, R)}$

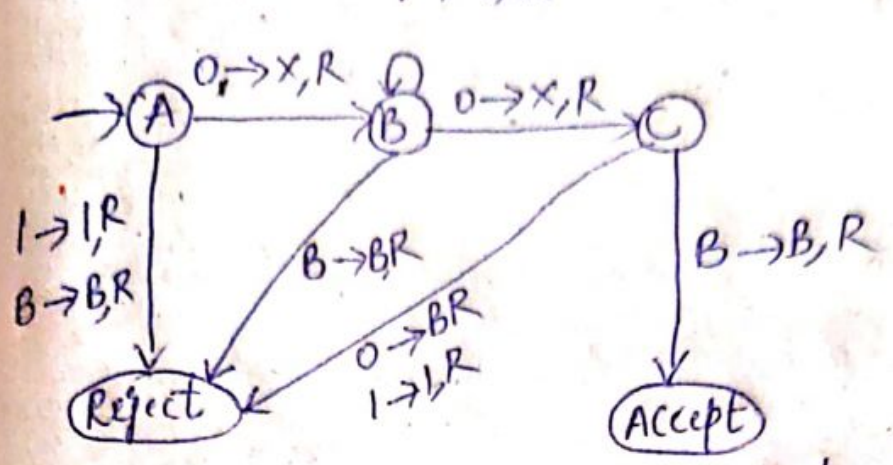
$(q_0, a) \rightarrow q_1, a, R$

q_0, X, L

Scanned with CamScanner

Design turing machine which recognizes the simple language

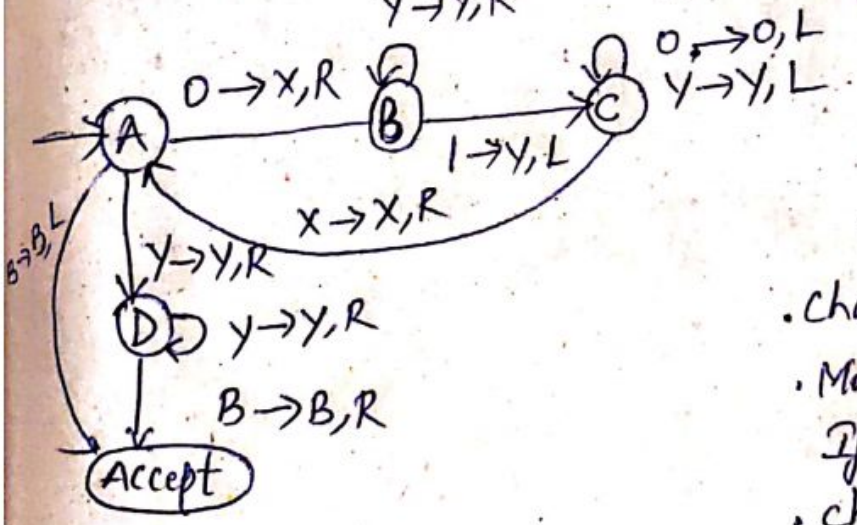
$L = 0 1^* 0$



0	1	1	0		
X	Y	Y	X	B	B

$L = 0^N 1^N$

0	0	0	0	1	1	1	1	B	B
X				Y					



Algorithm

- Change '0' to 'X'
- Move Right to first '1'
  If None: Reject
- change '1' to 'Y'
- Move Left to leftmost '0'
- Repeat the above steps until no more 0's
- Make sure no more 1's rem

Scanned with CamScanner

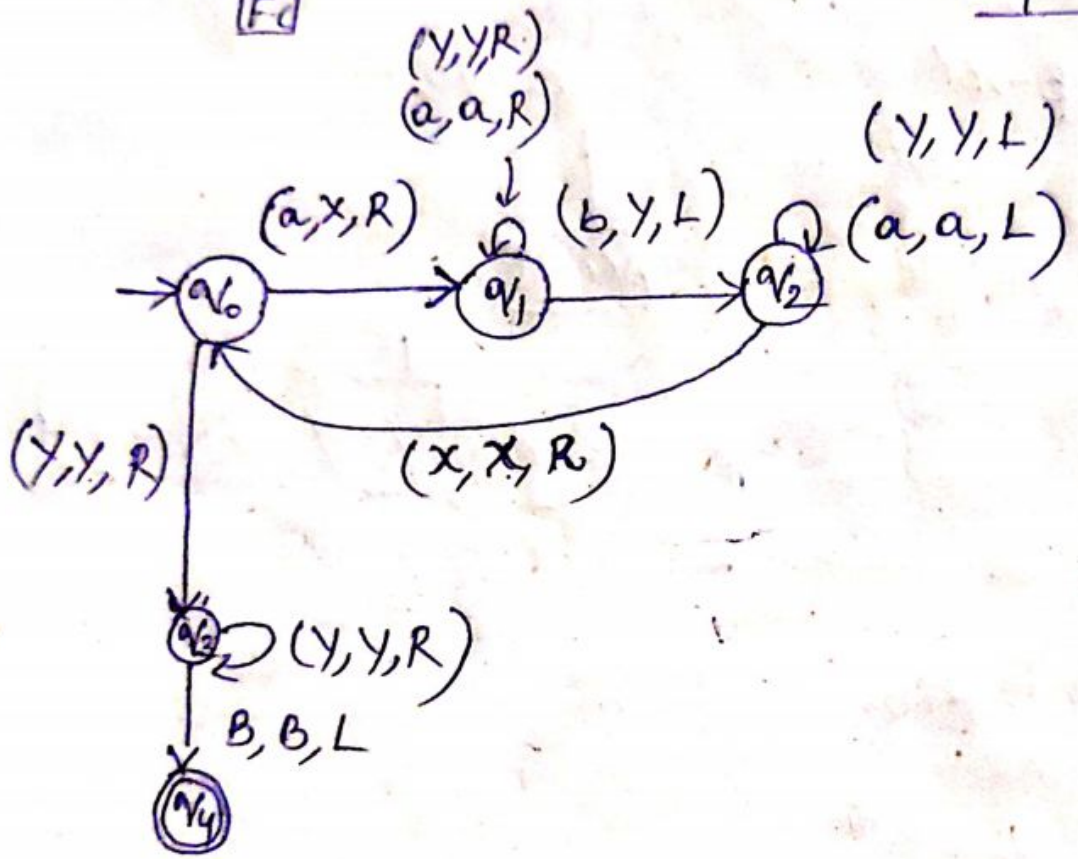
Design Turing machine for $\{a^n b^n \mid n \geq 1\}$

B	a	a	a	b	b	b	B

↑ R/W

FC

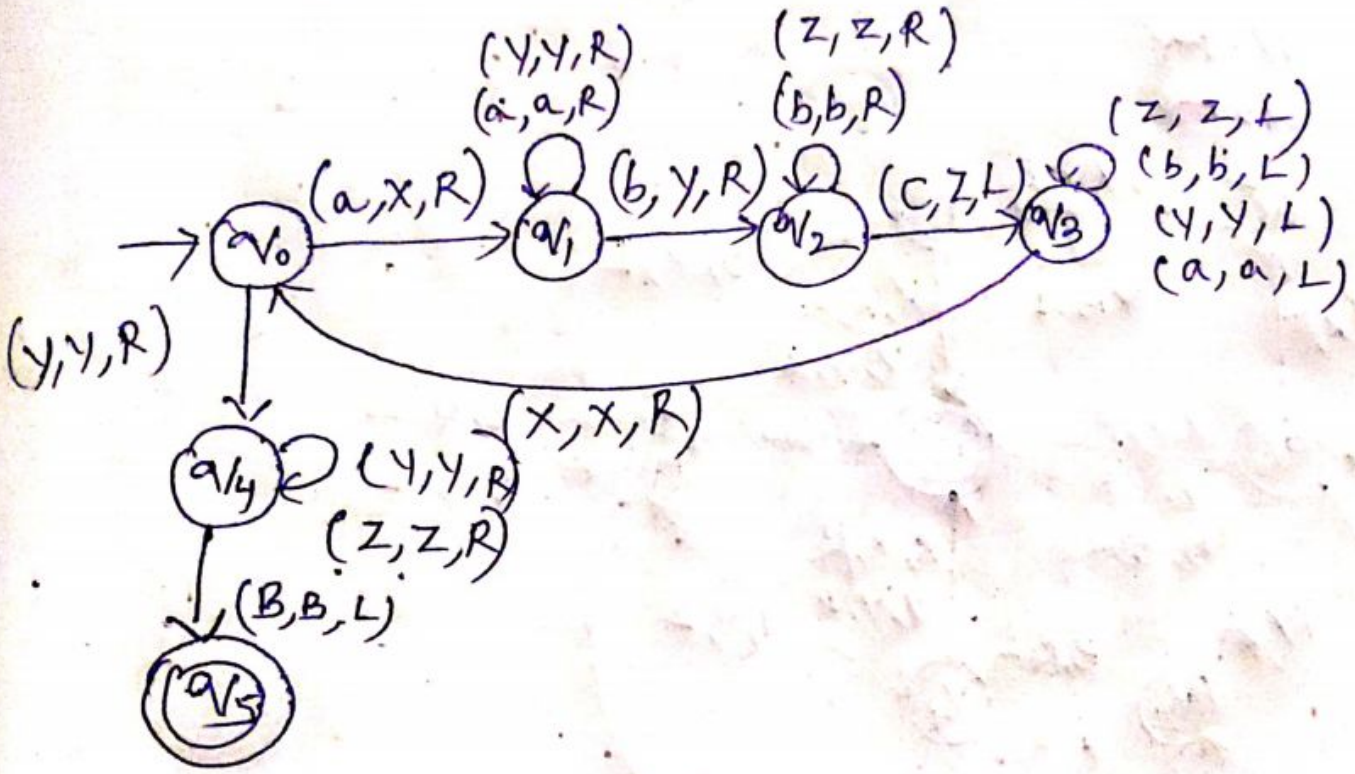
	X	X		Y	Y	
B	a	a	a	b	b	b



Scanned with CamScanner

Design Turing Machine for $\{a^n b^n c^n \mid n \geq 1\}$

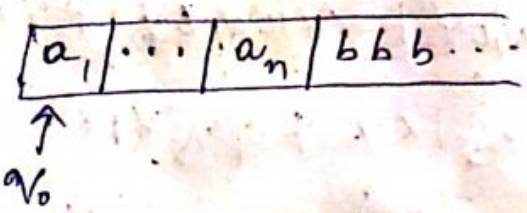
B	B	a	a	a	b	b	b	c	c	c	B	B



Scanned with CamScanner

Turing Machine as an Acceptor

The Turing Machine can be considered as an accepting device accepting sets of strings. Later we shall see that turing machines accept the family of languages generated by type 0 grammars. The set accepted by a Turing machine is called a recursively enumerable set. When we consider the Turing machine as an accepting device, we usually consider a one way infinite tape. The turing machine consists of a one way infinite read-write tape and a finite control



The input $a_1 \ldots a_n$ is placed at the left end of the tape. The rest of the cells contain the blank symbol b. Initially the tape head points to the leftmost cell in the initial state q_0. At any time the tape head will point to a cell and the machine will be in a particular state. Suppose the machine is in state q and pointing to a cell containing the symbol a, then depending upon the δ mapping (transition function) of the T.M it will change state to p and write a symbol X replacing a and move its tape head one cell to the left or to the right.

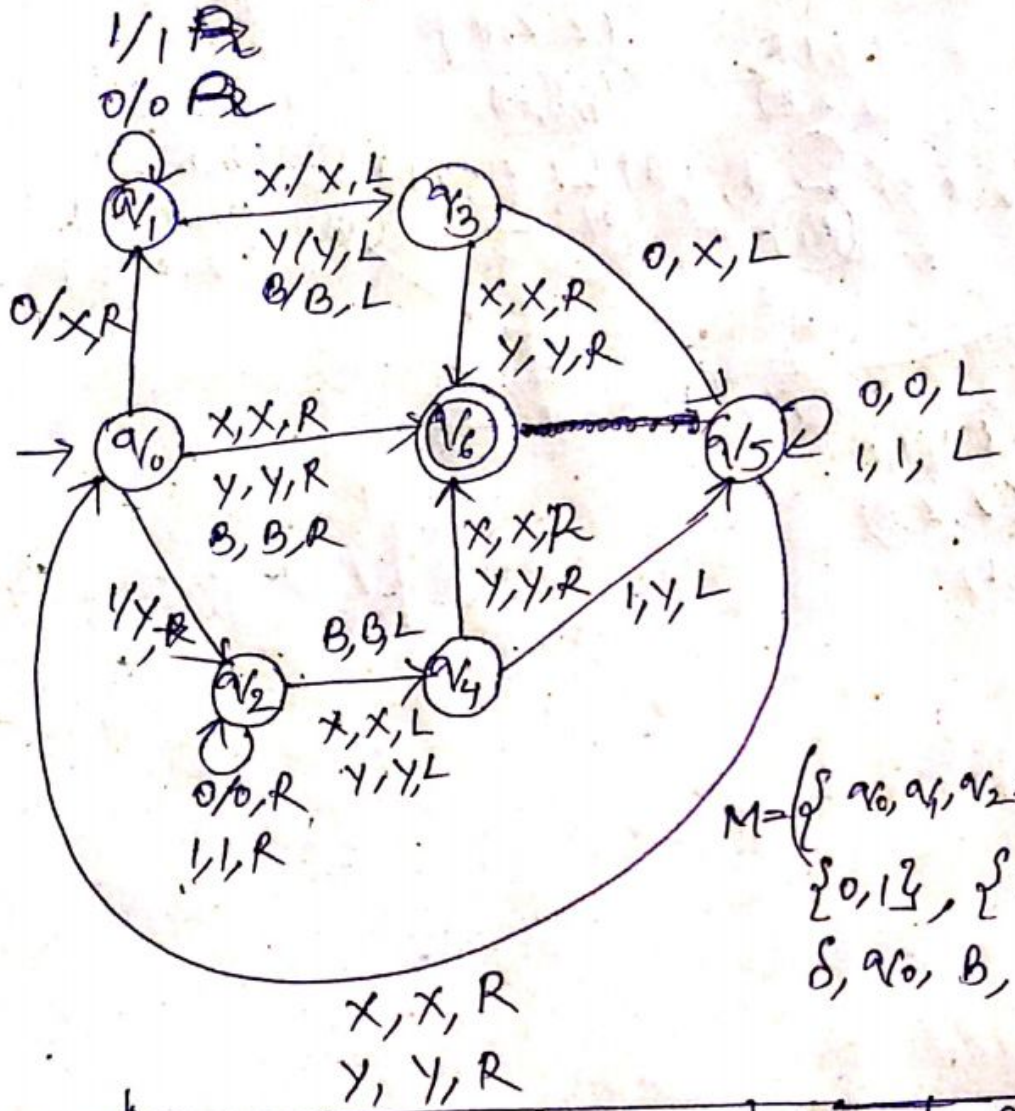
Scanned with CamScanner

The Turing Machine is not allowed to move off the left end of the tape. When it reaches a final state it accepts the input. ~~Now we consider~~

Language	Grammar	Automata
Recursive enumerable language	→ Type 0	Turing Machine
Context sensitive	→ Type 1	Linear Bounded Automata
Context free	→ Type 2	push Down Automata
Regular	→ Type 3	finite Automata

Scanned with CamScanner

Design T.M that will accept the language consisting of all palindromes of 0's and 1's



$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_6\})$.

State	0	1	X	Y	B
q_0	(q_1, X, R)	(q_2, X, R)	(q_6, X, R)	(q_6, Y, R)	(q_6, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, X, L)	(q_3, Y, L)	(q_3, B, L)
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	(q_4, X, L)	(q_4, Y, L)	(q_4, B, L)
q_3	(q_5, X, L)	—	(q_6, X, R)	(q_6, Y, R)	—
q_4	—	(q_5, Y, L)	(q_6, X, R)	(q_6, Y, R)	—
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, X, R)	(q_0, Y, R)	—

Scanned with CamScanner

If the input is 1001

$q_0 1001 \vdash Yq_2 001 \vdash Yoq_2 01 \vdash Yooq_2 1 \vdash Yoo1q_2$

$\vdash Yooq_4 1 \vdash Yooq_5 Y \vdash Yoq_5 oY \vdash Yq_5 ooY$

$\vdash q_5 YooY \vdash Yq_0 ooY \vdash YXq_1 oY \vdash YXoq_1 Y \vdash YXq_3 oY$

$\vdash YXq_5 XY \vdash Yq_5 XXY \vdash YXq_0 XY \vdash Yq_6 XXY$

Consider another Computation with input 10101

$q_0 10101 \vdash Yq_2 0101 \vdash Yoq_2 101 \vdash Yo1q_2 01 \vdash Yo10q_2 1$

$\vdash Yo101q_2 \vdash Yo10q_4 1 \vdash Yo10q_5 Y \vdash Yo1q_5 oY$

$\vdash Yoq_5 1oY \vdash Yq_5 o1oY \vdash q_5 Yo1oY \vdash Yq_0 o1oY$

$\vdash YXq_1 1oY \vdash YX1q_1 oY \vdash YX1oq_1 Y \vdash YX1q_3 oY$

$\vdash YX1q_5 XY \vdash YXq_5 1XY \vdash Yq_5 X1XY \vdash YXq_0 1XY$

$\vdash YXYq_2 XY \vdash YXq_4 YXY \vdash YXYq_6 XY$

for another example, Consider what M does on the input 1011, which is not in the language accepted.

$q_0 1011 \vdash Yq_2 011 \vdash Yoq_2 11 \vdash Yo1q_2 1 \vdash Yo11q_2 \vdash Yo1q_4 1$

$\vdash Yo1q_5 Y \vdash Yoq_5 1Y \vdash Yq_5 o1Y \vdash q_5 Y1oY \vdash Yq_0 o1Y$

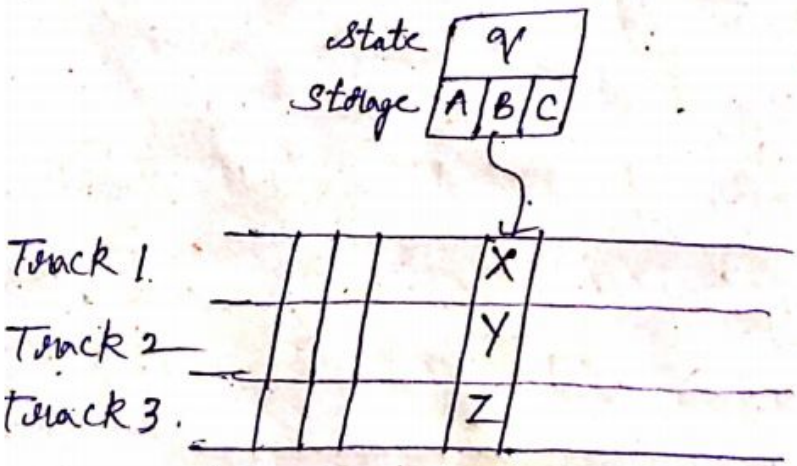
$\vdash YXq_1 1Y \vdash YX1q_1 Y \vdash YXq_3 1Y$.

Scanned with CamScanner

Types of Turing Machines

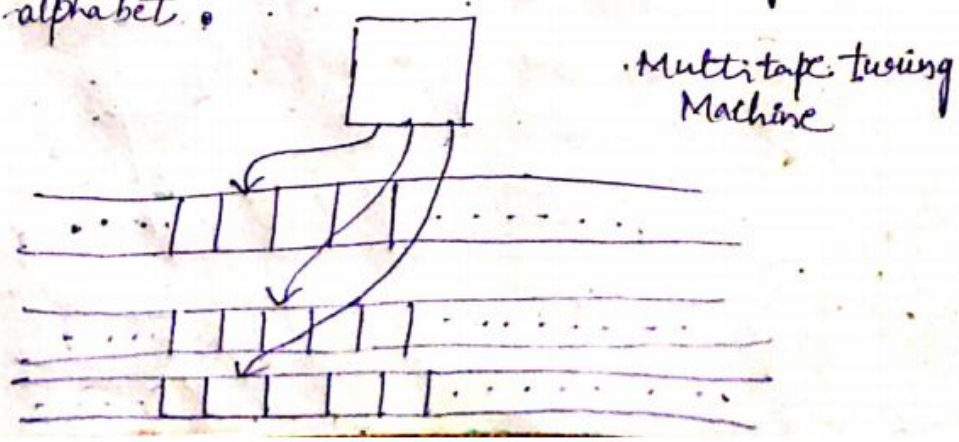
① Multiple Tracks

This type of Turing machine Consists of several tracks Each track can hold one symbol and the tape alphabet of the T.M. Consists of tuples with one component for each track. Thus for instance the cell scanned by the tape head in the below fig Contains the symbol [X, Y, Z]



② Multitape Turing Machine

A multitape TM has a finite Control and some finite number of tapes. Each tape is divided into cells and each cell can hold any symbol of the finite tape alphabet.



Scanned with CamScanner

(1) The input, a finite sequence of input symbols, is placed on the first tape.

(2) All other cells of all the tapes hold the blank.

(3) The finite control is in the initial state.

(4) The head of the first tape is at the left end of the input.

(5) All other tape heads are at some arbitrary cell. Since tapes other than the first tape are completely blank, its does not matter where the head is placed initially, all cells of these tapes "look" the same.

A move of the multitape TM depends on the state and the symbol scanned by each of the tape heads.
In one move, the multitape TM does the following.

(1) The control enters a new state, which could be the same as the previous state.

(2) On each tape, a new tape symbol is written on the cell scanned. Any of these symbols may be the same as the symbol previously there.

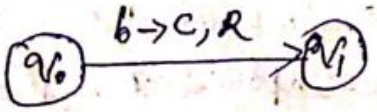
(3) Each of the tape heads makes a move, which can be either left, right or stationary. The heads move independently, so different heads may move in different directions and some may not move at all.

Scanned with CamScanner

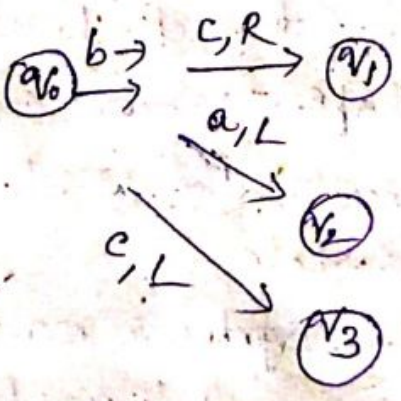
③ Non deterministic Turing Machines

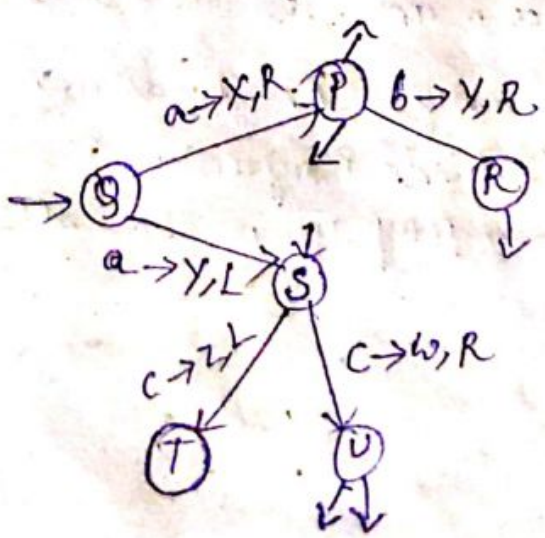
A non deterministic Turing Machine differs from the deterministic variety in

Deterministic



Non Deterministic





Scanned with CamScanner

(4) Semi infinite tape

The tape is semi-infinite, that is, there are no cells to the left of the initial head position.

The trick behind the construction is to use two tracks on the semi-infinite tape. The upper track represents the cells of the original T.M that are at or to the right of the initial head position. The lower track represents the positions left of the initial position, but in reverse order. Notice the * on the leftmost cell's bottom track. This symbol serves as an endmarker and prevents the head of the semi-infinite T.M from accidentally falling off the left end of the tape.

X_0	X_1	X_2	...
*	X_{-1}	X_{-2}	...

We shall make one more restriction to our turing machine: it never writes a blank. This simple restriction, coupled with the restriction that the tape is only semi-infinite means that the tape is at all times a prefix of nonblank symbols followed by an infinity of blanks. Further, the sequence of nonblanks always begins at the initial tape position.

Scanned with CamScanner

(5) Multistack turing Machine

A K-stack machine is a deterministic PDA with K-stacks. It obtains its input like the PDA does, from an input source. rather than having the input placed on a tape or stack as the T.M does.
Multistack machine has a finite control, which is in one of a finite set of states. It has a finite stack alphabet which it uses for all its stacks.
A move of the multistack machine is based on

(6)

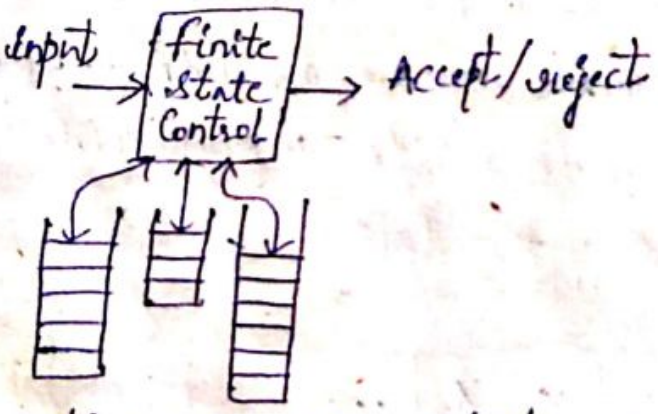


fig: A machine with three stacks

(1) The state of the finite Control.

(2) The input symbol read, which is chosen from the finite input alphabet. Alternatively, the multistack machine can make a move using ε input but to make the machine deterministic, there cannot be a choice of an ε-move or a non ε-move in any situation

(3) The top stack symbol on each of its stacks.

Scanned with CamScanner

In one move, the multistack machine can:

a) Change to a new state.

b) Replace the top symbol of each stack with a string of zero or more stack symbols. There may be different replacement string for each stack.

- Thus, a typical transition rule for a K-stack machine looks like.

$$\delta(q, a, X_1, X_2, \ldots, X_K) = (p, \gamma_1, \gamma_2, \ldots \gamma_K)$$

The interpretation of this rule is that in state q, with X_i on top of the ith stack, for $i = 1, 2, \ldots K$, the machine may consume a (or ϵ) from its input, go to state p, and replace X_i on top of the ith stack by string γ_i for each $i = 1, 2, \ldots K$. The multistack machine accepts by entering a final state.

We assume there is a special symbol $\$$, called the endmarker that appears only at the end of the input and is not part of that input. The presence of the end marker allows us to know when we have consumed all the available input.

Scanned with CamScanner

Counter Machines

A Counter machine may be thought of in one of two ways.

① The Counter machine has the same structure as the multistack machine. But in place of each stack is a Counter. Counters hold any non-negative integer, but we can only distinguish b/w zero and non zero Counters. That is, the move of the Counter machine depends on its state, input symbol and which if any of the Counters are zero. In one move, the Counter machine can:

(a) Change state

(b) Add or subtract 1 from any of its Counters, independently. However, a Counter is not allowed to become negative. So it cannot subtract 1 from a Counter that is currently

② A Counter machine may also be regarded as a restricted multistack machine. The restrictions are as follows:

(a) There are only two stack symbols, which we shall refer to as Z_0 (the bottom of stack marker) and X.

(b) Z_0 is initially on each stack.

(c) we may replace Z_0 only by a string of the form $X^i Z_0$ for some $i \geq 0$

(d) we may replace X only by X^i for some $i \geq 0$. That is, Z_0 appears only on the bottom of each stack and all other stack symbols are X.

Scanned with CamScanner

Turing Computable functions

Let f be any function which takes some input string and gives some answer (o/p string)

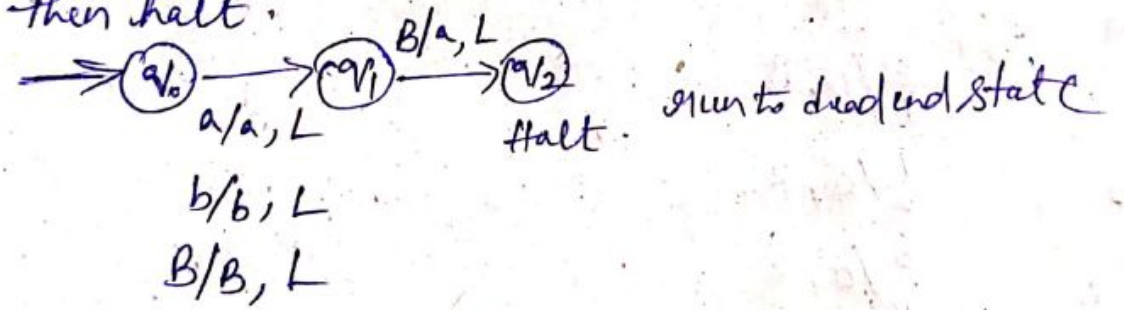
we say f is Turing Computable $\exists$ a T.M for which if we start with x on the tape, the T.M halts with $f(x)$ on the tape (don't care about accept/reject).

Eg: Let $\Sigma = \{a, b\}$, $f: \Sigma^* \to \Sigma^*$

$f(x) = ax$ (put a infront of the string)

$f(bbab) = abbab$

our turing machine must move left once then write 'a' then halt.



for functions of 2 parameters we could specify with a separator.

for inputs : 01101 & 11001

Separated : 01101 # 11001

stacked : $\frac{0}{1}\ \frac{1}{1}\ \frac{1}{0}\ \frac{0}{0}\ \frac{1}{1}$

Scanned with CamScanner

Bitwise and" is Turing Computable

- inputs are binary strings

$0 \rightarrow F$

$1 \rightarrow T$

and (00111 , 10101)

00101

Convenient to stack inputs

$\frac{0}{1}\ \frac{0}{0}\ \frac{1}{1}\ \frac{1}{0}\ \frac{1}{0}$ ← input string for the T.M

0 0 1 0 1

$\frac{0}{1}/0, R$

$\frac{1}{0}/0, R$

$\frac{0}{0}/0, R$

$\frac{1}{1}/1, R$

Scanned with CamScanner

"Bitwise add"

01001 + 01110

add:

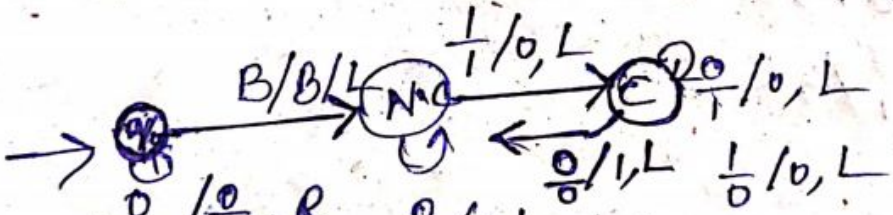
```
  01001   ← 9
  01110   ← 14
 -------
  10111   ← 23
```

0	1	0	0	1
0	1	1	1	0

move all the way to Right

Carry state

no Carry state



$\frac{0}{0} / \frac{0}{0}, R$ $\quad \frac{0}{0} / 0, L$

$\frac{1}{0} / \frac{1}{0}, R$ $\quad \frac{0}{1} / 1, L$

$\frac{1}{1} / \frac{1}{1}, R$ $\quad \frac{1}{0} / 1, L$

$\frac{0}{1} / \frac{0}{1}, R$

Scanned with CamScanner

Recursive and Recursive Enumerable Languages

Recursive Enumerable or Type-0 Language

R.E Languages or type-0 languages are generated by type-0 grammars. An R.E language can be accepted or recognized by T.M which means it will enter into final state for the strings of Language and may or may not enter into rejecting state for the strings which are not part of the Language. It means T.M can loop forever for the strings which are not a part of the Language. R.E languages are also called as Turing recognizable languages.

Recursive Language

A recursive language (subset of R.E) can be decided by T.M which means it will enter into final state for the strings of language and rejecting state for the strings which are not part of the Language. Eg: $L = \{a^n b^n c^n | n \geq 1\}$ is recursive because we can construct a turing machine which will move to final state if the string is of the form $a^n b^n c^n$ else move to non final state. So, the T.M will always halt in this case. REC languages are also called as Turing decidable languages.

Scanned with CamScanner

Closure properties of Recursive languages

union: If L_1 and L_2 are two recursive languages, their union $L_1 \cup L_2$ will also be recursive because if T.M halts for L_1 and halts for L_2, it will also halt for $L_1 \cup L_2$

Concatenation: If L_1 and L_2 are two recursive language their concatenation $L_1 \cdot L_2$ will also be recursive. for Example

$$L_1 = \{a^n b^n c^n \mid n >= 0\}$$
$$L_2 = \{d^m e^m f^m \mid m >= 0\}$$

$$L_3 = L_1 \cdot L_2$$
$$= \{a^n b^n c^n d^m e^m f^m \mid m >= 0 \text{ and } n >= 0\}$$

is also recursive.

L_1 says 'n' no: of a's followed by 'n' no: of b's followed by 'n' no: of c's. L_2 says 'm' no: of d's followed by 'm' no: of e's followed by 'm' no: of f's. Their Concatenation first matches no: of a's, b's and c's and then matches no: of d's, e's and f's. So it can be decided by T.M.

Scanned with CamScanner

Kleen closure

If L_1 is recursive, its Kleen closure L_1^* will also be recursive. for example

$$L_1 = \{a^n b^n c^n \mid n >= 0\}$$

$$L_1^* = \{a^n b^n c^n \mid n >= 0\}^* \text{ is also recursive}$$

Intersection and complement

If L_1 and if L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.

$$L_1 = \{a^n b^n c^n d^m \mid n >= 0 \text{ and } m >= 0\}$$

$$L_2 = \{a^m b^n c^n d^n \mid n >= 0 \text{ and } m >= 0\}$$

$$L_3 = L_1 \cap L_2$$

$$= \{a^n b^n c^n d^n \mid n >= 0\} \text{ will be recursive}$$

L_1 says 'n' no: of a's followed by 'n' no: of b's followed by 'n' no: of c's and then any no: of d's. L_2 says any no: of a's followed by 'n' no: of b's followed by 'n' no: of c's followed by 'n' no: of d's. Their intersection says 'n' no: of a's followed by 'n' no: of b's followed by 'n' no: of c's followed by 'n' no: of d's. So it can be decided by turing machine, hence recursive

Similarly, Complement of recursive Language L_1 which is $\Sigma^* - L_1$, will also be recursive.

Note: As opposed to REC Languages, R.E Languages are not closed under Complemenion which means complement of R.E Language need not be R.E.

Scanned with CamScanner

Recursive enumerable language	Recursive language
L is R.E if there is T.M	L is Recursive if there is Halting / Total T.M
Three states: Halt and Accept Halt and Reject Never Halt	Two states: Halt and Accept Halt and Reject
Closed under all except difference, Complement	Closed under all except Homomorphism and substitution

Scanned with CamScanner

Context sensitive languages - Type-1

CSL is generated by context sensitive grammer

A grammar is a quadruple (V, Σ, S, P) such that

V is a finite set of variable symbols.

Σ is the alphabet (of terminal symbols) of the grammar

It is required that $\Sigma \cap V = \phi$

$S \in V$ is the starting variable

P is a finite set of productions π, of the form

$\alpha \rightarrow \beta$, where $\alpha, \beta \in (\Sigma \cup V)^*$

Given a grammar $G = (V, \Sigma, S, P)$; G is Context-sensitive if every production in P is of the form.

$$\alpha A \beta \rightarrow \alpha \gamma \beta \qquad (\epsilon)$$

with $A \in V$, $\alpha, \beta \in (\Sigma \cup V)^*$ and $\gamma \in (V \cup \Sigma)^* - \lambda$

However $S \rightarrow \lambda$ is allowed provided that there is no rule in P with S on the right. A language $L \subseteq \Sigma^*$ is a Context-sensitive language if it is generated by some Context sensitive grammar $G = (V, \Sigma, S, P)$. This means that for every string $s \in L$ there is a derivation of s from S using the production P.

Scanned with CamScanner

(i) first of all it should be already unrestricted grammar

ie $\alpha \rightarrow \beta$

(ii) second condition is $|\alpha| \leq |\beta|$

Eg: $S \rightarrow AB$

$AB \rightarrow abC$

$B \rightarrow b$

Context sensitive languages are closed under

union

Concatenation

Reversal

Kleene star

Intersection.

Scanned with CamScanner

Linear Bounded automata

Linear Bounded Automata is a single tape Turing Machine with two special tape symbols call them left marker < and right marker >

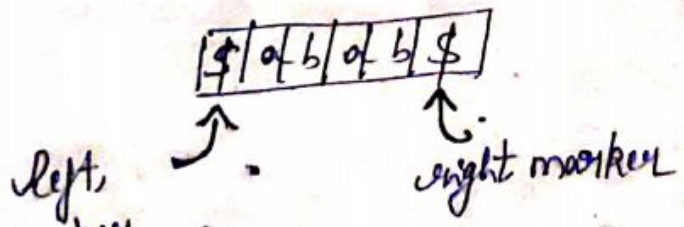
The transitions should satisfy these conditions

- It should not replace the marker symbols by any other symbol
- It should not write on cells beyond the marker symbols.

Thus the initial configuration will be

$$\langle q_0 a_1 a_2 a_3 \cdots a_n \rangle$$

LBA = T.M + Input size tape.



$$F.A < PDA < LBA < T.M$$

Standard examples

$L_1 = \{a^n b^n c^n, n \geq 1\}$

$L_2 = \{a^n : n \text{ is prime}\}$

$L_3 = \{a^n : n \text{ is nonprime}\}$

$L_4 = \{a^{n!}, n \geq 0\}$

$L_5 = \{ww : w \in (a,b)^+\}$

$L_6 = \{www^R : w \in (a,b)^+\}$

$L_7 = \{w^n : w \in \{a,b\}^+, n \geq 1\}$

Scanned with CamScanner

Formally Linear Bounded Automata is a non deterministic Turing Machine

$$M = (Q, \Sigma, \Gamma, \delta, \sqcup, q_0, <, >, t, r)$$

Q is set of all states

Σ is set of all terminals

Γ is set of all tape alphabets $\Sigma \subset \Gamma$

δ is set of transitions

$\sqcup$ is blank symbol

q_0 is the initial state

$<$ is left marker and $>$ is right marker

t is accept state.

r is reject state

Chomsky Hierarchy

Language	Automata
Recursively enumerable Types	Turing machine
Type 1 : Context Sensitive	Linear Bounded Automata
Context-free type 2	push Down Automata
Regular type 3	Finite Automata

Scanned with CamScanner

Chomsky Hierarchy

According to chomsky hierarchy, grammars are divided of 4 types

Type 0 known as unrestricted grammar.

Type 1 known as context sensitive grammar

Type 2 known as context free grammar

Type 3 Regular Grammar.

Type 0: unrestricted grammar

In type 0

Type 0 grammars include all formal grammars.

Type 0 grammar Language are recognized by turing machine. These languages are also known as the Recursively Enumerable languages.

Grammar production in the form of

$\alpha \rightarrow \beta$

V: Variables

T: Terminals

Where

α is $(V+T)^* V (V+T)^*$

β is $(V+T)^*$

In type 0 there must be atleast one variable on left side of production. Eg: $Sab \rightarrow ba$

$A \rightarrow S$

Here variables S, A and terminals a, b

Scanned with CamScanner

Type 1: Context sensitive Grammar

Type 1 grammars generate the Context sensitive languages. The language generated by the grammar are recognized by the Linear Bound Automata.

① First of all type 1 grammar should be Type 0

② Grammar production in the form of

$$\alpha \rightarrow \beta$$

$$|\alpha| <= |\beta|$$

ie Count of symbol in α is less than or equal to β.

for example

$S \rightarrow AB$

$AB \rightarrow abc$

$B \rightarrow b$

Scanned with CamScanner

Type 2: Context free Grammar

Type 2 grammars generate the context free languages. The language generated by the grammar is recognized by a push Down automata.

In Type 2,

1. first of all it should be Type 1
2. Left hand side of production can have only one variable.

$|\alpha| = 1$

Their is no restriction on β

for Example,

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

Type 3: Regular Grammar

Type 3: grammars generate regular languages. These languages are exactly all the languages that can be accepted by a finite state automaton.

Type 3 is most restricted form of grammar.

Type 3 should be in the given form.

$V \rightarrow VT / T$ (left regular grammar)

$V \rightarrow TV / T$ (right regular grammar)

so: $S \rightarrow a$

Scanned with CamScanner

The above form is called strictly regular grammar.

There is another form of regular grammar called extended regular grammar. In this form:

$V \rightarrow VT^* / T^*$ (Extended left regular grammar)

(or)

$V \rightarrow T^*V / T^*$ (Extended right regular grammar)

<u>Eg:</u>

$S \rightarrow ab$

Scanned with CamScanner

Post Correspondence problem

An instance of post Correspondence problem (PCP) Consists of two lists of strings over some alphabet Σ, the two lists must be of equal length. we generally refer to the A and B lists and write $A = w_1, w_2, \ldots, w_k$ and $B = x_1, x_2, \ldots x_k$ for some integer k. for each i, the pair (w_i, x_i) is said to be a corresponding pair.

we say this instance of PCP ~~(w_i, x_i) is said to be~~ has a solution, if there is a sequence of one or more integers $i_1, i_2, \ldots i_m$ that, when interpreted as indexes for strings in the A and B lists, yield the same string. That is $w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$. we say the sequence $i_1 i_2 \ldots i_m$ is a solution to this instance of PCP if so, The post correspondence problem is:

Given an instance of PCP, tell whether this instance has a solution.

	List A	List B
i	w_i	x_i
1	110	110110
2	0011	00
3	0110	110

An instance of PCP.

Scanned with CamScanner

In this case, PCP has a solution. for instance $i_1 = 2$ $i_2 = 3$ $i_3 = 1$ ie the solution is the list 2, 3, 1. we verify that this list is a solution by concatenating the corresponding strings in order for the two lists, that is

$$w_2 w_3 w_1 = x_2 x_3 x_1$$

$$\overbrace{001}^{w_2}\overbrace{10110}^{w_3}\overbrace{110}^{w_1} = \overbrace{001}^{x_2}\overbrace{10}^{x_3}\overbrace{110110}^{x_1}$$

Note this solution is not unique for instance.

Scanned with CamScanner

Dominos:

B	A	CA	ABC
CA	AB	A	C

We need to find a sequence of dominos such that the top and bottom strings are the same.

A	B	CA	A	ABC	→ ABCAAABC
AB	CA	A	AB	C	→ ABCAAABC

A	B		
1	111	→	$\frac{1}{111}$ ①
10111	10	→	$\frac{10111}{10}$ ②
10	0	→	$\frac{10}{0}$ ③

② ① ① ① ③

A: 10 111 1 1 0

B: 10 111 1 1 0

② ① ①

② ① ① ③

A: $\frac{10111}{10}$ $\frac{1}{1}$ $\frac{1}{1}$ $\frac{3}{0}$

B: $\frac{10111}{2}$ $\frac{1}{1}$ $\frac{1}{1}$ $\frac{0}{3}$

Scanned with CamScanner

Example

The below Example tells us that pcp is undecidable

	A	B	
①	10	101	→ 10 / 101
②	011	11	→ 011 / 11
③	101	011	→ 101 / 011

① ① ①
10 10
101 101 → This is not possible.

② ① ②
10 011
101 11 → This not possible

③ ① ③ ①
~~101~~ 10 101 10
101 011 101
① ③ ①

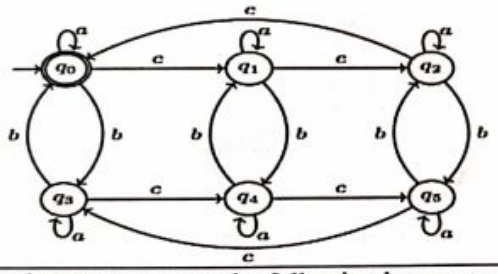
After ③ if we take ① they are not equal so avoid it.

① ③ ③
10 101 101
101 011 101 ① .....

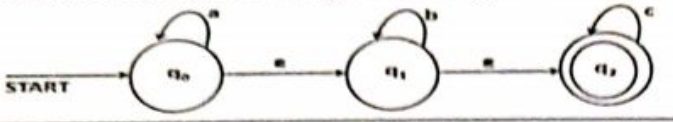
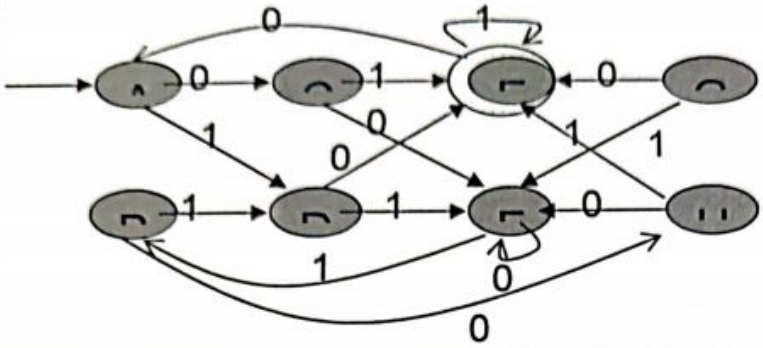
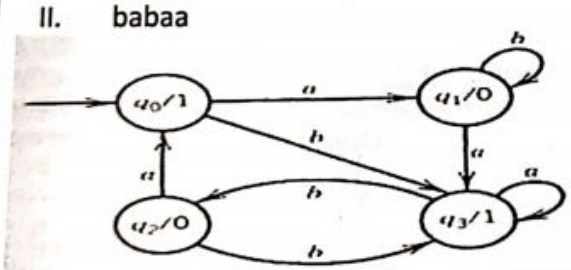
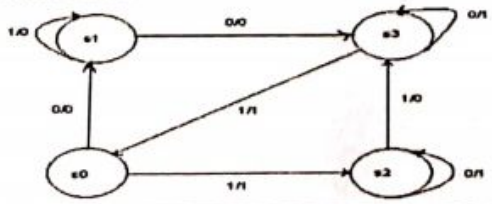
Here extra 1 is leftover and you will never get the solution hence you can say that it is undecidable.

Scanned with CamScanner

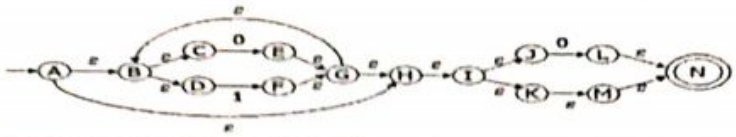
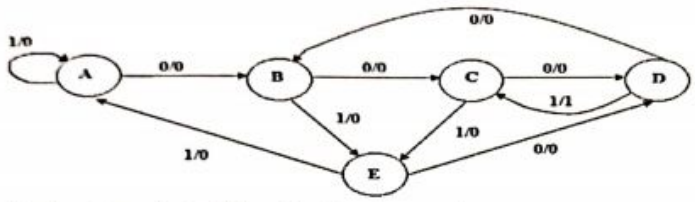
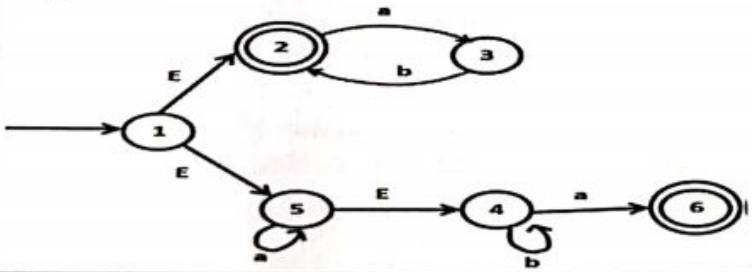
PROBLEM BASED LEARNING

No.	A] Problems on Finite Automata (1-25)
1.	Design Deterministic Finite Automata (DFA) accepts all strings without two consecutive 1's over an alphabet {0, 1}.
2.	Give the formal definition of Finite Automata? List limitations of finite automata.
3.	Design Deterministic Finite Automata (DFA) for the following languages, over the alphabet {0,1}: i. Set of all strings containing the substring 0110 ii. Set of all strings that do not contain the substring 1010
4.	Consider the Finite Automata and check the string acceptance for the following strings I. abcbaca II. abccacba 
5.	Obtain DFA to accept the following languages over input alphabet {a,b}. i. Every string starts with 'b' and ends with 'a'. ii. Every string contains odd number of a's and even number of

Scanned with CamScanner

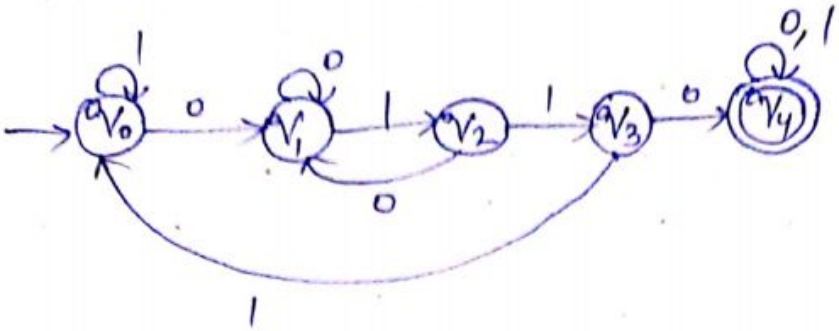
	b's and check the string acceptance for the string **"aabbabb"**
6.	Design Moore machine to print the **residual modulo 5** over the ternary string and find the output string for the given input sequence: **"120021"**
7.	Obtain a DFA from the following ε-NFA using ε-closures. 
8.	Construct the minimum state equivalent DFA using partition method. 
9.	Obtain a NFA to accept strings ending with "ab" where Σ={a,b}. Also obtain the equivalent DFA.
10.	Write any four differences between DFA and NFA and give the formal definition of DFA and NFA.
11.	Draw a NFA which accepting the set of all strings whose second last symbol is always 1 over the alphabet {0, 1} and check the string acceptance for "101010".
12.	Consider the Moore machine find the output string for following strings I. abbabb II. babaa 
13.	Obtain a DFA for accepting the set of all strings of 0's and 1's such that the 10th symbol from the right end is 1 and remaining nine symbols are 0's and check the string acceptance for the string **"01000000000"**.
14.	Define Moore Machine. Discuss the process for deriving equivalent Moore machine from the given Mealy machine. Construct equivalent Moore machine for the given Mealy Machine by considering S0 is initial state. 
15.	Consider the given NFA with ε-transitions and construct an equivalent DFA.

Scanned with CamScanner

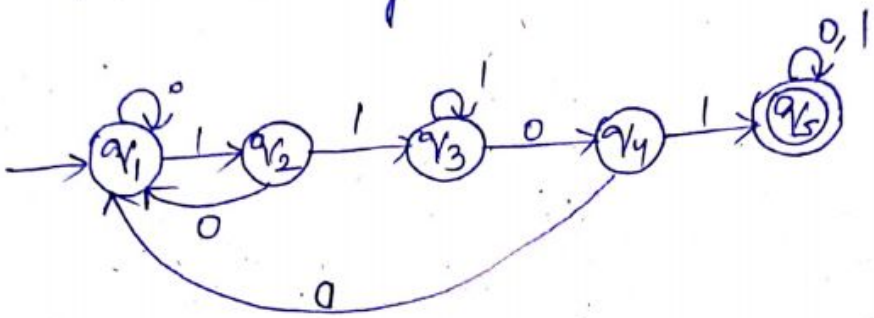
	
16.	Consider the following NFA M=({q0,q1,q2,q3,q4},{0,1},δ,q0,{q4}) where δ(q0,0)=q1, δ(q1,0)=q2, δ(q2,0)=q2, δ(q2,1)={q2,q3}, δ(q3,1)=q4 then construct transition table ,transition diagram and construct equivalent DFA and conclude which type of strings are accepted by resultant finite automata.
17.	Define the following terms with example I. E-closure II. Moore Machine III. Extended Transition Function IV. Klenn Closure & Positive Closure
18.	Design DFA for accepting set of strings not contains the substring "1010" over the alphabet {0, 1} and check the string acceptance for the sequence:"1110001"
19.	Consider the Mealy machine by considering A as initial state  Derive the output strings for the given strings I. 0101010 II. 10111
20.	Find Deterministic Finite Automata for the following languages on Σ={a,b} ,L ={w : n_a(w) mod 5>0} and show that string **"aaaa"** is accepted by the resultant automata.
21.	Construct Moore machine for the language if all strings contains substring **"aba"** then it prints output 'Y', or if it contains substring **"abba"** then it prints output 'N' otherwise it prints 'X' and derive the output string for the given input: **"abababbba".**
22.	Convert the given ε – NFA to DFA and check for the string acceptance by resultant DFA for the string:**"aabba"** 
23.	Minimize the finite automata

Scanned with CamScanner

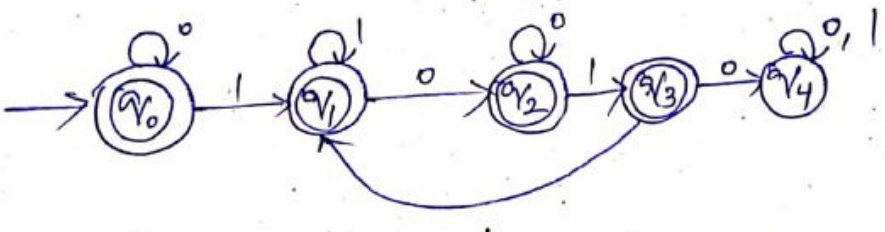
0110 is a substring.



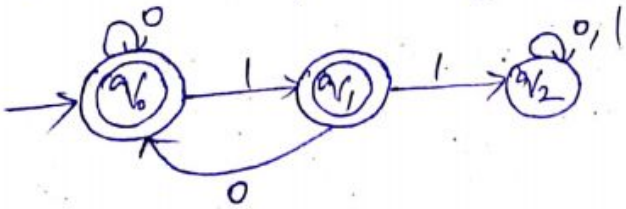
1101 is a substring



1010 not, a substring.



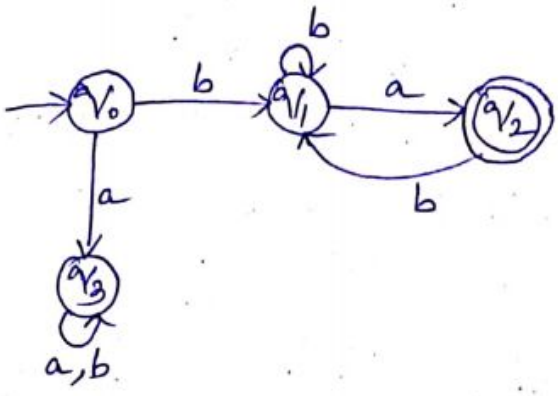
DFA accepting all strings without two consecutive 1's over an alphabet $\{0,1\}$



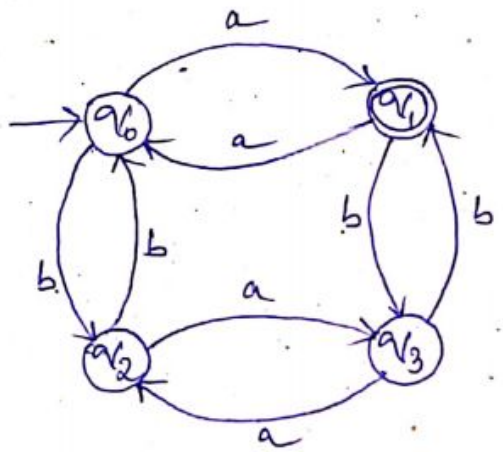
Scanned with CamScanner

(5)

(i)

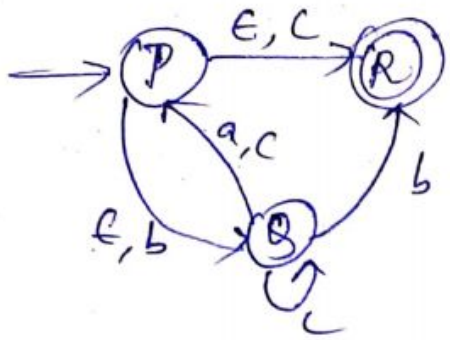


(ii) Every string that contains odd number of a's and even number of b's and check the string acceptance for the string "aabbabb".



Scanned with CamScanner

Convert the following ε- NFA to its Equivalent NFA.



Scanned with CamScanner