

Automata Theory :-

Problems are a set of problem instances. A problem instance is a specific instance of a problem. We predominantly deal with yes/no problems which have 2 sets, the accept/yes and the reject/no sets.

A problem is computable only if all the instances are computable/solvable by the model.

A computational model solves a problem P if (and only if?)

i) If inputs belonging to the Yes instance of P , then the device outputs YES / ACCEPT

ii) If inputs belonging to the No instance of P , then the device outputs NO / REJECT

If i) & ii) hold, P is computable.

Alphabet:- Any finite, non-empty set of symbols

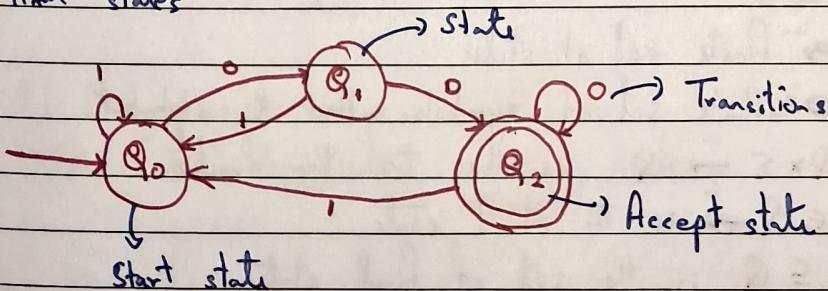
Strings / Words:- Finite sequence of symbols from an alphabet

Language:- Set of words/strings from the current alphabet

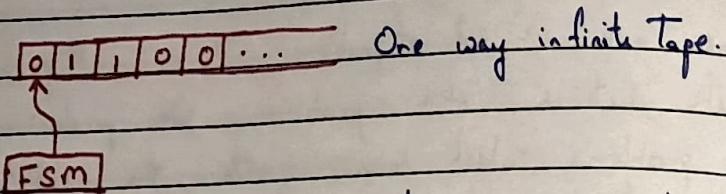
Generally, ϵ denotes the empty string

Deterministic Finite State Automata Model [DFA]

Has a single start state, unique transitions and zero or more final states



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



After the input is read entirely, and the machine is in the final/accept state, then it returns YES / ACCEPT ; else it outputs NO / REJECT

Run:- The total computation done during the reading and executing of the entire input string. The string is accepted or rejected after the complete run.

Definition:- Let the DFA be M . Then the language that M accepts is :-

$$L(M) = \{w \mid w \text{ results in an accepting run}\}$$

$\{w \mid \text{set of all strings } w \text{ such that } M(w) \text{ accepts}\}$

This is also known as the language of the automaton.

Definition:- For any language L , we say M solves/decides L if

- i) $\forall w \in L, M(w) \text{ accepts}$
- ii) $\forall w \notin L, M(w) \text{ rejects}$

Definition:- A DFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of states

Σ is a finite set of symbols called the alphabet

$\delta: Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of final states

11
110
001
111

classmate

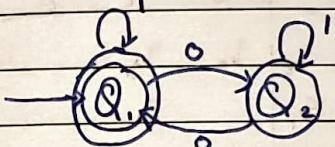
Date _____
Page _____

Characteristics of a DFA:-

- (i) Single Start State
- (ii) Unique Transitions
- (iii) Zero or more final states

Language of a DFA is the set of all binary strings ^{that result in} ending in 00 an accept state at the end of the computation.

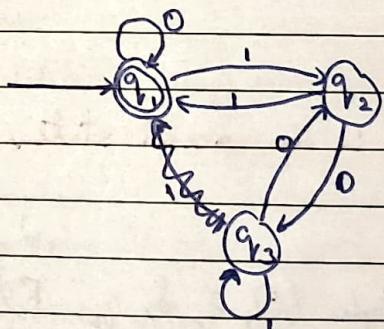
Example:- i) $\Sigma = \{0, 1\}$, $L(m) = \{\omega \mid \omega \text{ has an even no of } 0s\}$



ii) $\Sigma = \{0, 1\}$, $L(m) = \{\omega \mid \omega \text{ is divisible by } 3\}$

Any input string would leave remainders 0, 1, 2
DFA will have 3 states

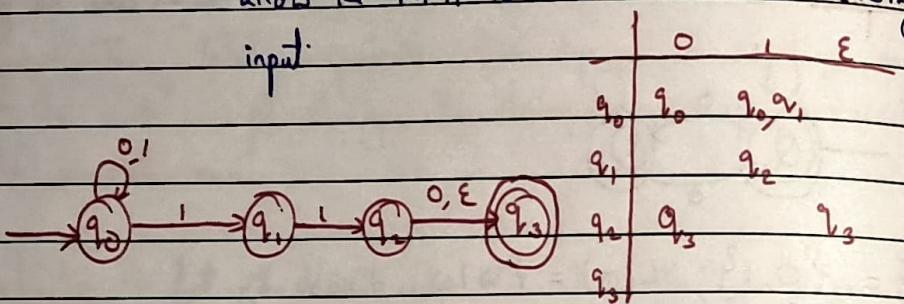
If ω is divisible by 3, then after reading a 0 it should stay in the final state, and so on.



If L is solvable by a DFA, then $\neg L$ is also solvable by a DFA. [Just toggle the accepting states]

Non-deterministic finite state Automata (NFA)

- Characteristics:-**
- i Single start state
 - ii Zero or more final states
 - iii Multiple transitions are possible on the same input state
 - iv Some transitions might be missing.
 - v ϵ -transitions or empty transitions exist which allow the NFA to transition without reading any input.



A given input can have multiple runs.

Take 10110 as the input string. It could accept, reject or crash in this case.

Crashes occur when the machine has no transition for an input

For the 10110 example, a possible crash run is -

$q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \xrightarrow{\epsilon} q_3 \xrightarrow{0} \text{crash}$

Crashes are rejecting runs.

If \exists at least one run results in an accept state, the input is accepted, else rejects

Definition :- An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

$Q \Rightarrow$ finite set of states

$\Sigma \Rightarrow$ finite set of alphabets (contains ϵ)

$\delta \Rightarrow Q \times \Sigma \rightarrow P(Q)$, is the transition function

$q_0 \in Q \Rightarrow$ start state

$F \subseteq Q \Rightarrow$ set of accepting states

Intuitively NFAs are more powerful than DFAs because of non-determinism.

$\rightarrow L_2$

The languages accepted by DFAs are a subset of the languages accepted by NFAs.

\downarrow

L_1

$L_2 \subseteq L_1$

$L_1 \subseteq L_2$ (Proof upcoming)

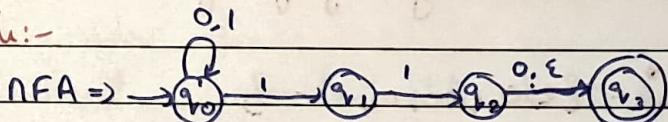
Hence $L_1 = L_2 \rightarrow$ Both are equally powerful.

We prove this by showing that we can construct a DFA to accept the same languages as ~~NFA~~ ^{an}. These DFAs are called Remembering DFAs,

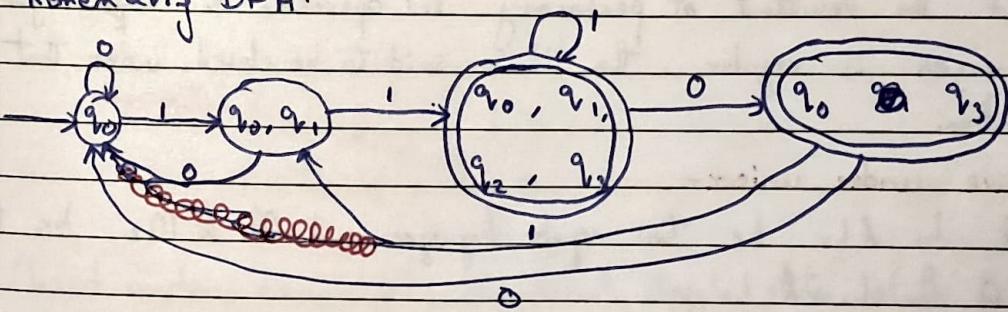
Steps :-

- i) Let R be a remembering DFA
- ii) R on an input enters a state labelled by all possible states that NFA can enter on that input.
- iii) This trims away the non-determinism of the NFA without losing its language
- iv) Also, if NFA has k states, R has at most 2^k states [Power state]

Example:-



Remembering DFA:



Fact-1:-

Regular Languages:- A language is called regular if \exists a finite state automaton that decides it.

Let M be a finite state automaton and,

$$L(M) = \{w \mid w \text{ is accepted by } M\}$$

then $L(M)$ is regular.

Any language has a set of operations that can be performed on it associated with it.

Regular Operations:- Let L_1, L_2 be languages

Binary \rightarrow i) Union :- $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$

ii) Concatenation :- $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$

Unary \rightarrow iii) Star :- $L_1^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in L_1\}$

Countably $\infty \leftarrow$ \hookrightarrow L, concat any no. of strings from L.

④ Σ :- $\Sigma = \{a\} ; \Sigma^* = \{\epsilon, a, aa, \dots\}$

$\Sigma = \{\emptyset\} ; \Sigma^* = \{\emptyset\}$

ϵ belongs to the * of any language by convention

$$L = \{0, 1\} \rightarrow L^* = \{\epsilon, 0, 1, 00, 01, \dots\}$$

Closure of Regular Languages

If the resultant of performing set operation on ~~two~~ regular languages is regular, then it is said to be closed under that operation.

i) Closure under union:-

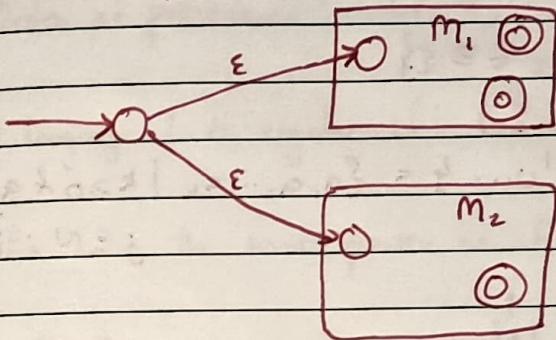
Let L_1 & L_2 be two regular languages and M_1 & M_2 be the FSA for L_1 & L_2 .

Construct an NFA by having a common start state with 2^Σ

transition to the start states of M_1 & M_2 . This decides ~~not~~ L, UL.

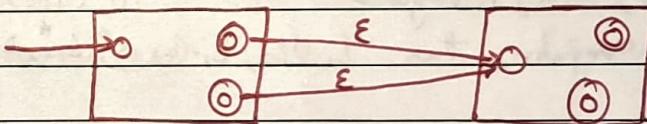
Hence Union is closed

Math proof in slides [Lec-3]



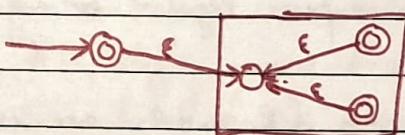
(ii) Closure under concatenation :-

Construct an NFA with start state as that of M_1 and ϵ transitions b/w final states of M_1 to the start state of M_2 .



(iii) Closure under star :-

Construct a new start state which is an accepting state, have an ϵ transition to the old start state, and have ϵ -loop backs from the final states to the old start state.



(iv) Closed under complement:- [Just toggle states of the DFA] Not for NFA ↴

v) Closure under intersection:- Yes, proof using De Morgan's Law with Union and Complement.

Notation:- Σ is an alphabet

i) $\Sigma^0 = \{\epsilon\}$

ii) $\Sigma^2 = \{a_1 a_2 \mid a_1, a_2 \in \Sigma\}$

iii) $\Sigma^k = \{a_1 \dots a_k \mid a_i \in \Sigma\}$

iv) $\Sigma^+ = \bigcup_{i \geq 0} \Sigma^i = \{\Sigma^0 \cup \Sigma^1 \cup \dots\} = \{a_1 a_2 \dots a_n \mid k \geq 0 \text{ & } a_j \in \Sigma \forall j \in \mathbb{N}; i \leq k\}$

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

- First Principles:- [Ab initio]

Re Lang alt def:- Let Σ be an alphabet. Then the following are Re langs over Σ

i) \emptyset is regular

ii) for each $a \in \Sigma$, $\{a\}$ is regular

iii) If L_1 & L_2 are regular, then $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^* , L_2^* are regular.

Syntax for Regular Expressions :- R is said to be a regular expression if it has one of the following forms:-

i) \emptyset is a regular expression. $L(\emptyset) = \emptyset$

ii) ϵ is a regular expression. $L(\epsilon) = \{\epsilon\}$

iii) Any $a \in \Sigma$ is a regular expression, $L(a) = \{a\}$

Regular Expressions are ways to express regular languages

iv) $R_1 + R_2$ is a regular expression if R_1 and R_2 are regular expressions, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$

v) R^* is a regular expression if R is a regular expression, $L(R^*)$

vi) $R_1 R_2$ is a regular expression if R_1, R_2 are regular expressions, $L(R_1 R_2) = L(R_1) \cdot L(R_2)$

regular expression:-

Regular Expressions are like the alphabet of the Regular Language

vii) (R) is a regular expression if R is a regular expression. $L((R)) = L$

Order of precedence :- $(), +, \cdot, ^$

A language L is regular iff for some regular expression R , $L(R) = L$

REs have the same power as FSA.

Reg Expr	$L(R)$
01	$\{01\}$
$01 + 1$	$\{01, 1\}$
$(0+1)^*$	$\{\epsilon, 0, 1, 00, 01, \dots\}$
$(01+E)^\epsilon$	$\{01, \epsilon\}$
$(0+1)^*01$	$\{01, 001, 101, 0001, \dots\}$
$(0+10)^*(E+1)$	$\{\epsilon, 0, 10, 00, 001, 010, 0101, \dots\}$

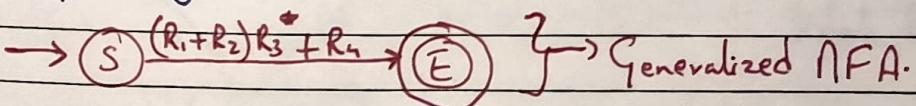
Check slides for example - Lec 3. I'm ~~deccpy~~ ^{high} ;)

DFA to RE:-

If a lang is regular, it accepts an RE.

Given a DFA, we recursively construct a 2 state generalized NFA with

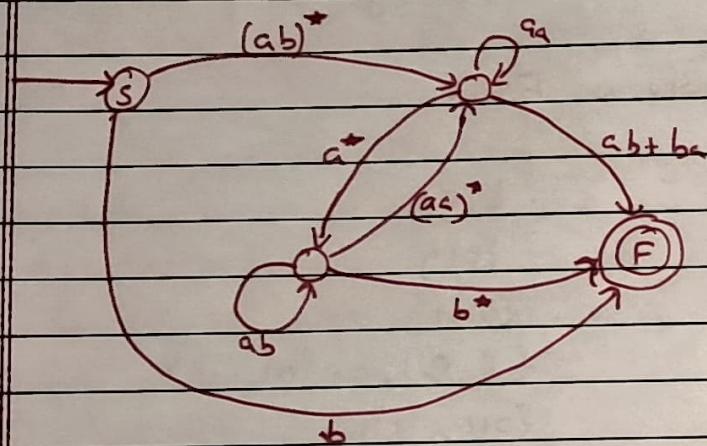
- i) a start and end state [Single; no incoming arrows to start, outgoing from end]
- ii) A single arrow from start to end
- iii) The label of the arrow is the RE corresponding to the language accepted by M.



hello cold world hello h ε / /

Generalized NFAs

- Transitions may have REs
- Unique start state with only outgoing arrows
- Unique final state with only incoming arrows
- During a run in a GNFA, blocks of symbols might be read instead of single symbols. [These correspond to REs]



This GNFA has multiple accepting strings like
b, abababab, aacbbag, etc.

↳ Uses ϵ transitions [think!]

"All transitions can be ϵ ←
due to it being RE!"

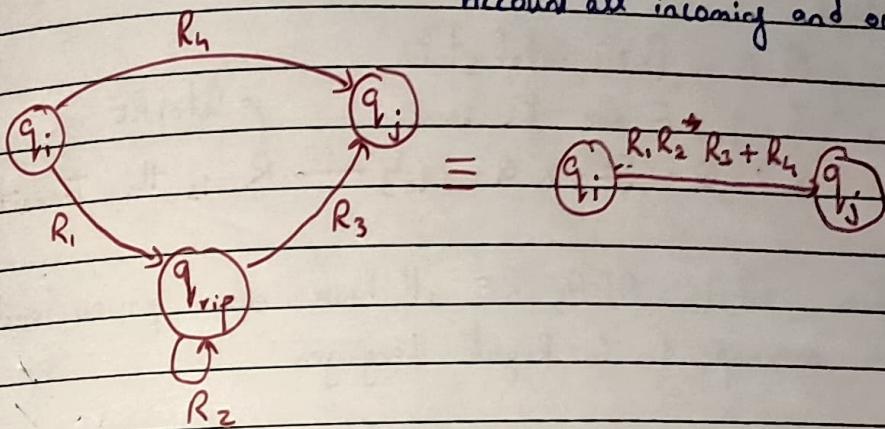
Starting from a DFA, we will begin by constructing a GNFA on k states. We then follow a recursive procedure to keep reducing the no. of states by 1 until we reach a GNFA with 1 state.

i) Add a new start state with same transitions to the old start state.

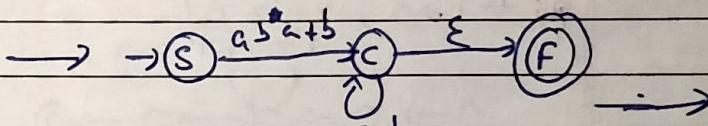
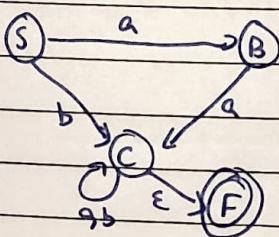
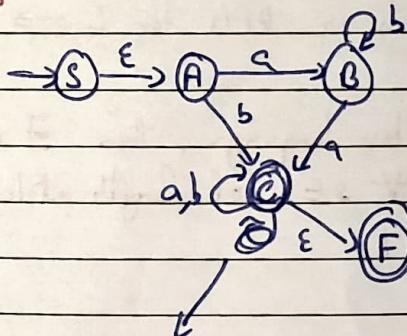
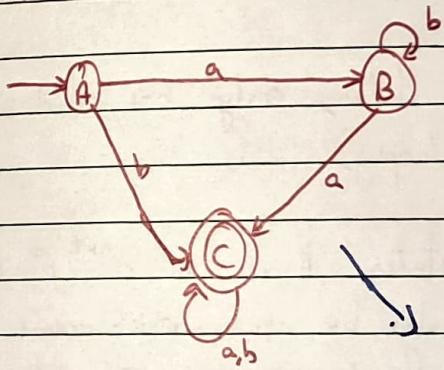
Add a new final state with ϵ transitions from all the old final states.

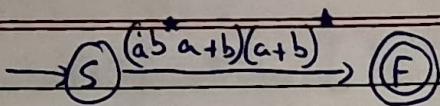
8 Pick any state [$i = \text{start/finish}$] of the GNFA
 Call it q_{rip} .

iii Rip out q_{rip} and adjust the states of the GNFA.
 Account all incoming and outgoing edges



Ex:-

DFA \rightarrow 



Formally, a GFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$
where

Q is a finite set of states

Σ, q_0, F are the same \uparrow Set of RE

$\delta : Q - \{F\} \times Q - \{q_0\} \rightarrow R$ is the transition fn.

Thus DFAs, NFAs, RE all have equal power and all of them correspond to Regular Languages

Pumping Lemma:-

Let $\Sigma = \{0, 1\}$. Consider $L = \{0^n, n \geq 0\}$.

For a DFA of n -distinct states, only $n-1$ sized strings can be accepted (without looping)

If there is a loop of t states, then even $0^{n+t} 1^n$ or (or perhaps $0^n 1^{n+t}$) will also be read, which means that \exists there exists no DFA for $L \rightarrow L$ is not regular.

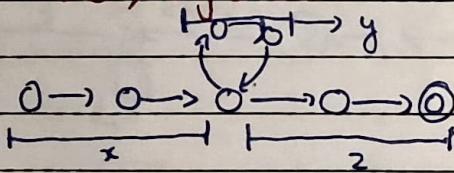
If L is a regular language, then \exists a number p (pumping length) where $\forall s \in L$ of length at least p , $\exists x, y, z$ s.t

$s = xyz$, s.t

i) $|y| \leq p$

ii) $|y| \geq 1$

iii) $\forall i \geq 0, xy^i z \in L$



We usually prove a language is not regular by contraposition.

Contraposition = If p s.t. $s \in L$ with $|s| \geq p$, $\nexists (x, y, z)$, with $s = xyz$ s.t. $\neg (|xyz| \leq p) \vee \neg (|y|^i \geq 1) \vee \neg (\forall i \geq 0, xy^i z \in L)$

Proof:- Say the DFA M has p states.

By the pigeonhole principle, any string $\geq p$ would encounter a loop.

Let $s = s_1 s_2 \dots s_n$ with $n \geq p$ and suppose $v_1 v_2 \dots v_m$ be the sequence of states encountered while implemting a run of s in M .

There would be two states $v_i = v_j$ where $i \neq j$. We can hence divide s into three parts $\rightarrow \underbrace{s_1 s_2 \dots s_i}_{x}, \underbrace{s_{i+1} \dots s_j}_{y}$ and $\underbrace{s_{j+1} \dots s_n}_{z}$

i) We can traverse the loop bit any number of times and so $\forall i \geq 0$, $xv^i z \in L$

ii) Also, as $j \neq i$, $|y| \geq 1$.

iii) The DFA reads $|xyz|$ before the end of the first loop, hence $|xyz| \leq p$

Context Free Grammars

Grammars - provide a way to generate strings belonging to a language
 \hookrightarrow set of rules \rightarrow systematic procedures

Contain variables and terminals

\hookrightarrow consist of strings over the alphabet corresponding to the language that the grammar generates

Formally, Grammars are 4-tuples (V, Σ, P, S) such that

- i) V is the set of variables
- ii) Σ is the set of terminals (disjoint from V)
- iii) P is the set of production rules
- iv) S is the start variable

$$P: (V \cup \Sigma)^* V (V \cup \Sigma)^* \rightarrow (V \cup \Sigma)^*$$

\hookrightarrow Has at least one variable on the left hand side

S is generally the variable on the LHS of the first rule.

Ex:- $G: X \rightarrow 1X$

$$X \rightarrow 0Y \quad S = X$$

$$Y \rightarrow 0X \quad V = [X, Y]$$

$$Y \rightarrow 1Y \quad \Sigma = [0, 1]$$

$$Y \rightarrow \epsilon$$

The process of substitutions using the rules of G required to obtain a certain string is called a derivation.

The generated string is said to be derived by the grammar

$$X \rightarrow 1X \rightarrow 11X \rightarrow 110Y \rightarrow 1101Y \rightarrow 1101$$

is a valid derivation.

A string belongs to the language of a Grammar if \exists derivation for it.

$$L(G) \text{ is } \{ w \in \Sigma^* \mid s \Rightarrow^* w \}$$

\hookrightarrow derivation

Regular Grammar :- If the rules of the Grammar G are of the form

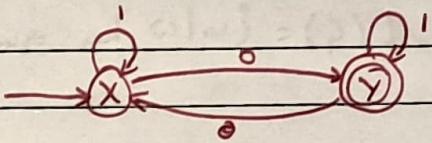
$\text{Var} \rightarrow \text{Ter Var}$

$\text{Var} \rightarrow \text{Ter}$

$\text{Var} \rightarrow \epsilon$

then the language of the grammar is regular. These are also called Right-linear grammar (a single var to the right of the terminals in the RHS)

DFA:-



Regular Grammar

A run in the DFA are analogous to the derivations in CFGs

Left linear grammar:- $\text{Var} \rightarrow \text{Var Ter}$

$\text{Var} \rightarrow \text{Ter}$

$\text{Var} \rightarrow \epsilon$

Mixing Left and Right Linear Grammars does not result in Regular Langs

Context Free Grammars:- If the rules of the Grammar is of the form $V \rightarrow (\cancel{V} \cancel{V} \cancel{S}) (V \cup T)^*$, then such a Grammar is Context Free.

Regular Grammars \subset CFG

Ex:- $S \rightarrow 0S1 \quad | \quad S \rightarrow \epsilon$ $\Rightarrow L(G) = \{\omega \mid \omega = 0^n 1^n, n \geq 0\}$

CFGs are more powerful than Regular Grammars / DFA / NFA

$S \rightarrow 0S1 \quad | \quad S \epsilon \quad | \quad \epsilon$ \Rightarrow If $0S1$ are $\{\}\}$, then it generates balanced parenthesis

Tips:-

1) Check if the CFL is a union of simpler languages

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow \dots$$

$$S_2 \rightarrow \dots$$

2) Grammars with rules like $S \rightarrow aSb$ help construct machines that might need unbounded memory.

Q) Construct a CFG to s.t $L(G) = \{w \mid w \text{ has equal no. of } 1s\}$

$$S \rightarrow SS$$

$$S \rightarrow OS1$$

$$S \rightarrow 1SO$$

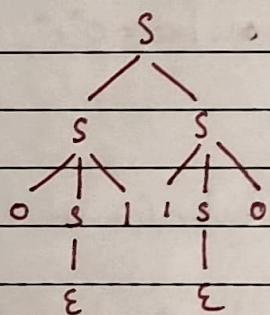
$$S \rightarrow \epsilon$$

Parse Trees for CFGs

Ordered trees that provide alternate representations of the derivations of a grammar.

Parsing is used in compilers.

Root of the tree is the start node, leaves are the terminals and the leaves are read from left to right. We move from a node to its children by applying the rules of the CFG.



Any string can have multiple derivations, but the parse trees are the same.

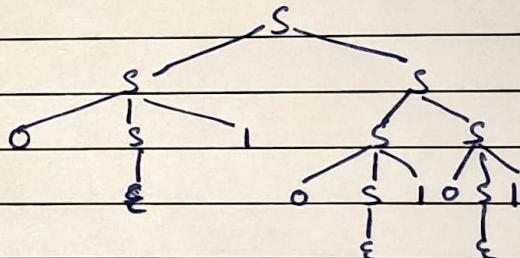
Definition:- If a string is derived by always replacing the leftmost var, it is called the leftmost derivation.

Definition:- Similar for rightmost derivation.

Derivations need not be leftmost or rightmost.

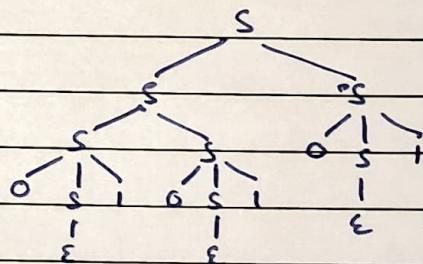
Ambiguous Grammars:- A CFG is called ambiguous if $\exists w \in L(G)$ such that there are two or more parse trees/ left derivations /right derivations for w .

Consider $w = 010101$ with $S \rightarrow 0S1 \mid SS \mid \epsilon$



Leftmost :- $S \rightarrow SS \rightarrow OS1S \rightarrow O1S \rightarrow OISS \rightarrow O10S1S \rightarrow O101S \rightarrow O10101$

$O10101 \leftarrow O1010S1 \leftarrow$



Ambiguity is not desirable, esp in compilers, since there would be different interpretations and hence different outcomes.

We can remove parenthesis or assign precedence while branching and change of rules. add vars