Definitions

finite automata

Fintite automata

Finite Automata(FA) is the simplest machine to recognize patterns.The finite automata or finite state machine is an abstract machine which have five elements or tuple. It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol. Basically it is an abstract model of digital computer.

Alphabet.

Definition – An **alphabet** is any finite set of symbols.

Kleen closure -> * (according to wiki and book)

Kleene Star

kleen

clean

- **Definition** – The Kleene star, $\sum^*$, is a unary operator on a set of symbols or strings, $\sum$, that gives the infinite set of all possible strings of all possible lengths over $\sum$ including λ.
- **Representation** – $\sum^* = \sum_0 \cup \sum_1 \cup \sum_2 \cup \ldots\ldots$ where $\sum_p$ is the set of all possible strings of length p.

Kleene Closure / Plus

- **Definition** – The set $\sum^+$ is the infinite set of all possible strings of all possible lengths over $\sum$ excluding λ.
- **Representation** – $\sum^+ = \sum_1 \cup \sum_2 \cup \sum_3 \cup \ldots\ldots$

  $\sum^+ = \sum^* - \{\lambda\}$

Language

- **Definition** – A language is a subset of $\sum^*$ for some alphabet $\sum$. It can be finite or infinite.

language

Scanned with CamScanner

dfa

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton**. As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where –

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

Regular language

regular language

A language is said to be regular is and only if some fsm recognises it

Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

regular expression

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language $\{\varepsilon\}$.
- If $a \in \Sigma$ (Σ represents the input alphabet), a is regular expression with language {a}.
- If a and b are regular expression, a + b is also a regular expression with language {a,b}.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a* (0 or more times a) is also regular.

regular grammar

Regular Grammar : A grammar is regular if it has rules of form A -> a or A -> aB or A -> ε where ε is a special symbol called NULL.

Scanned with CamScanner

closure properties

Closure Properties of Regular Languages

union

- **Union :** If L1 and If L2 are two regular languages, their union L1 ∪ L2 will also be regular. For example, L1 = {a^n | n ≥ 0} and L2 = {b^n | n ≥ 0}
  L3 = L1 ∪ L2 = {a^n ∪ b^n | n ≥ 0} is also regular.

intersection

- **Intersection :** If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,
  L1= {a^m b^n | n ≥ 0 and m ≥ 0} and L2= {a^m b^n ∪ b^n a^m | n ≥ 0 and m ≥ 0}
  L3 = L1 ∩ L2 = {a^m b^n | n ≥ 0 and m ≥ 0} is also regular.

- **Concatenation :** If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,
  L1 = {a^n | n ≥ 0} and L2 = {b^n | n ≥ 0}
  L3 = L1.L2 = {a^m . b^n | m ≥ 0 and n ≥ 0} is also regular.

kleen closure

- **Kleene Closure :** If L1 is a regular language, its Kleene closure L1* will also be regular. For example,
  L1 = (a ∪ b)
  L1* = (a ∪ b)*

- **Complement :** If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example,
  L(G) = {a^n | n > 3}
  L'(G) = {a^n | n <= 3}

equivalent

Two regular **expressions are equivalent** if languages generated by them are same. For example, (a+b*)* and (a+b)* generate same language. Every string which is generated by (a+b*)* is also generated by (a+b)* and vice versa.

NDFA/NFA

nfa

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

Formal Definition of an NDFA

An NDFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where –

- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the alphabets.
- **δ** is the transition function where δ: Q × ∑ → 2^Q

Scanned with CamScanner

(Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)

- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

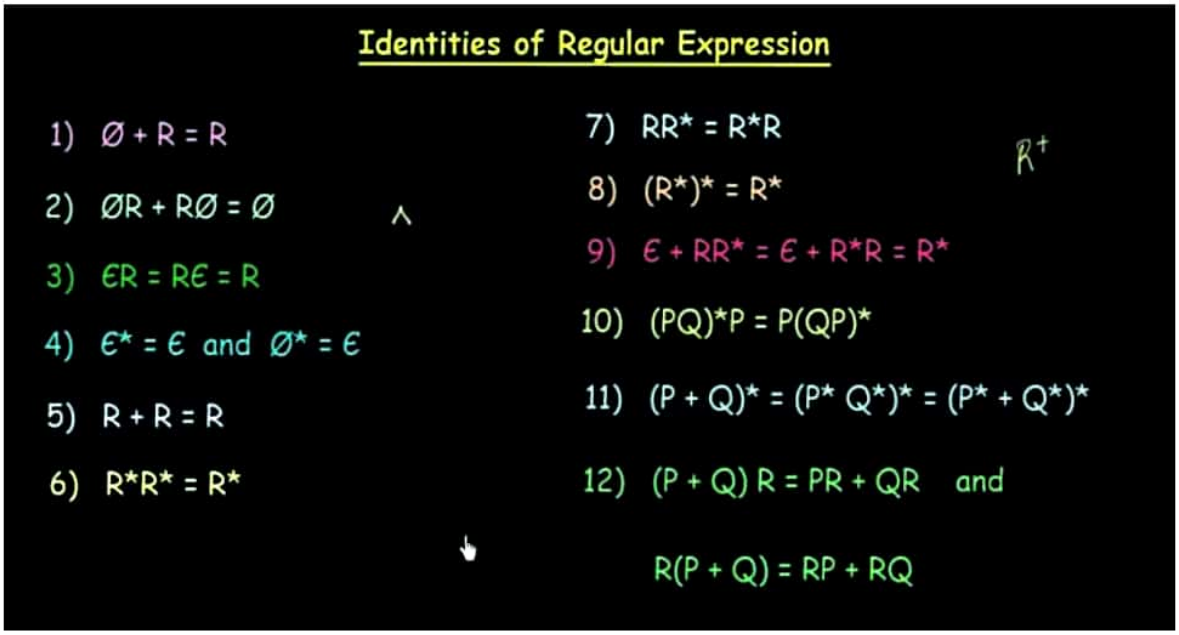
Unreachable state: a state is said to be unreachable if there is no way it can be reached from initial state.

Epsilon nfa

enfa

The NFA with epsilon-transition is a finite state machine in which the transition from one state to another state is allowed without any input symbol i.e. empty string ε. Adding the transition for the empty string doesn't increase the computing power of the finite automata but adds some flexibility to construct then DFA and NFA.

Identities of regular expression



regx

regex

Arden's theorem state that:
"If P and Q are two regular expressions over sigma, and if P does not contain epsilon, then the following equation in R given by R = Q + RP has an unique solution i.e., R = QP*."

ardens theorm

Scanned with CamScanner

Equivalence of 2 finite automata

The two finite automata (FA) are said to be equivalent if both the automata accept the same set of strings over an input set Σ.

When two FA's are equivalent then, there is some string x over Σ. On acceptance of that string, if one FA reaches to the final state, the other FA also reaches to the final state.

equivalence fsm

Method

The method for comparing two FA's is explained below –

Let M and M1 be the two FA's and Σ be a set of input strings.

Step 1 – Construct a transition table that has pairwise entries (q, q^1) where $q \in M$ and $q^1 \in M^1$ for each input symbol.

Step 2 – If we get in a pair as one final state and other non-final state then we terminate the construction of transition table declaring that two FA's are not equivalent

Step 3 – The construction of the transition table gets terminated when there is no new pair appearing in the transition table.

Mealy Machine

A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

mealy

It can be described by a 6 tuple $(Q, \sum, O, \delta, X, q_0)$ where –

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the input alphabet.
- **O** is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where $\delta: Q \times \sum \rightarrow Q$
- **X** is the output transition function where $X: Q \times \sum \rightarrow O$
- $\mathbf{q_0}$ is the initial state from where any input is processed $(q_0 \in Q)$.

Moore Machine

Moore machine is an FSM whose outputs depend on only the present state.

A Moore machine can be described by a 6 tuple $(Q, \sum, O, \delta, X, q_0)$ where –

- **Q** is a finite set of states.

moore

- $\sum$ is a finite set of symbols called the input alphabet.

tuples

Scanned with CamScanner

- **O** is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where δ: Q × ∑ → Q
- **X** is the output transition function where X: Q → O
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).

Mealy Machine	Moore Machine
Output depends both upon the present state and the present input	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Mealy Machine.
The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done.	The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur.
Mealy machines react faster to inputs. They generally react in the same clock cycle.	In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later.

Type text here

table1

Both have same power.

Moore -> mealy = no. of states same

mealy moore

Mealy -> moore = no. of states increases(states*output at max)

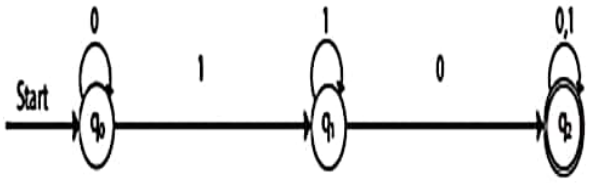
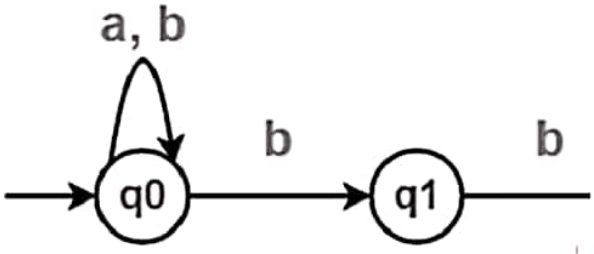
states

Scanned with CamScanner

Difference between dfa and nfa(both have same power)

Deterministic Finite Automata	Non-Deterministic Finite Automata
Each transition leads to exactly one state called as deterministic	A transition leads to a subset of states i.e. some transitions can be non-deterministic.
Accepts input if the last state is in Final	Accepts input if one of the last states is in Final.
Backtracking is allowed in DFA.	Backtracking is not always possible.
Requires more space. table2	Requires less space.
Empty string transitions are not seen in DFA.	Permits empty string transition.
For a given state, on a given input we reach a deterministic and unique state.	For a given state, on a given input we reach more than one state.
DFA is a subset of NFA.	Need to convert NFA to DFA in the design of a compiler.
$\delta : Q \times \Sigma \rightarrow Q$ For example – $\delta(q0,a)=\{q1\}$	$\delta : Q \times \Sigma \rightarrow 2^Q$ For example – $\delta(q0,a)=\{q1,q2\}$
DFA is more difficult to construct.	NFA is easier to construct.

Scanned with CamScanner

Deterministic Finite Automata	Non-Deterministic Finite Automata
DFA is understood as one machine.	NFA is understood as multiple small machines computing at the same time.
	

Kleene theorem

Kleene's Theorem-I :

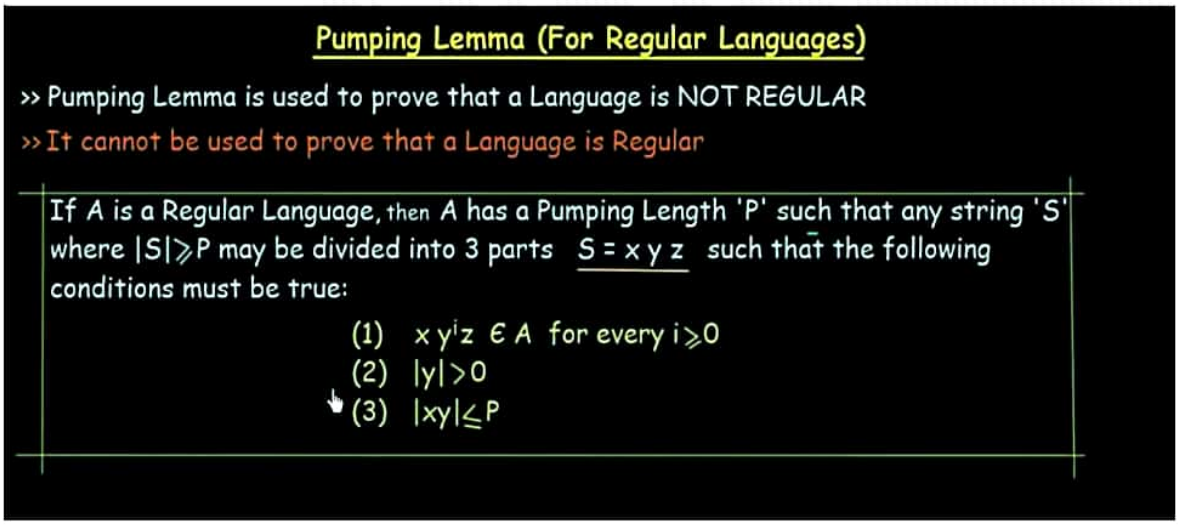
kleene theorm

For any Regular Expression r that represents Language L(r), there is a Finite Autom ata that accepts same language.

clean

pumping lemma

Pumping lemma for regular language



Scanned with CamScanner

noam chomsky hierarchy chomsky hierarchy TYPE

Noam Chomsky gave a Mathematical model of Grammar which is effective for writing computer languages

The four types of Grammar according to Noam Chomsky are:

Grammar Type	Grammar Accepted	Language Accepted	Automaton
TYPE-0	Unrestricted Grammar	Recursively Enumerable Language	Turing Machine
TYPE-1	Context Sensitive Grammar	Context Sensitive Language	Linear Bounded Automaton
TYPE-2	Context Free Grammar	Context Free Language	Pushdown Automata
TYPE-3	Regular Grammar	Regular Language	Finite State Automaton

Grammar :
It is a finite set of formal rules for generating syntactically correct sentences or meaningful correct sentences.

Constitute Of Grammar :
Grammar is basically composed of two basic elements –

1. **Terminal Symbols –**
   Terminal symbols are those which are the components of the sentences generated using a grammar and are represented using small case letter like a, b, c etc.

2. **Non-Terminal Symbols –**
   Non-Terminal Symbols are those symbols which take part in the generation of the sentence but are not the component of the sentence. Non-Terminal Symbols are also called Auxiliary Symbols and Variables. These symbols are represented using a capital letter like A, B, C, etc.

grammar 4tuples

Formal Definition of Grammar :
Any Grammar can be represented by 4 tuples – <N, T, P, S>

- **N –** Finite Non-Empty Set of Non-Terminal Symbols.
- **T –** Finite Set of Terminal Symbols.
- **P –** Finite Non-Empty Set of Production Rules.
- **S –** Start Symbol (Symbol from where we start producing our sentences or strings).

Scanned with CamScanner

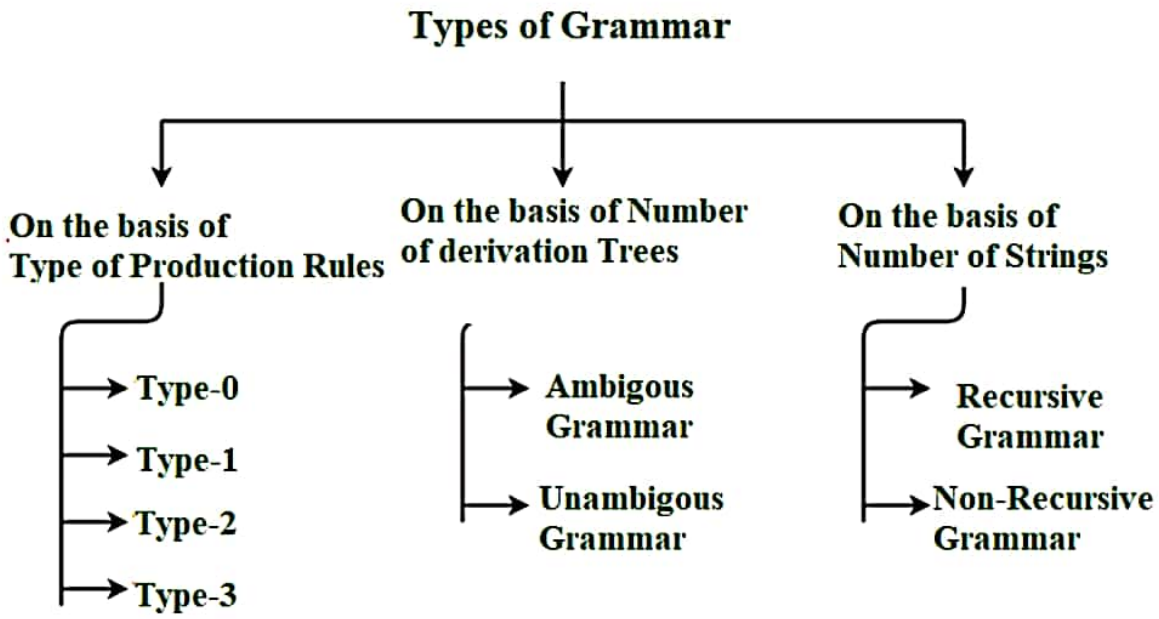
production rules

Production Rules :
A production or production rule in computer science is a rewrite rule specifying a symbol substitution that can be recursively performed to generate new symbol sequences. It is of the form alpha-> beta where alpha is a Non-Terminal Symbol which can be replaced by beta which is a string of Terminal Symbols or Non-Terminal Symbols.

types equivalent

Equivalent Grammars :
Grammars are said to be equivalent id they produce the same language.

Types of grammar



regular grammar

Regular Grammar : A grammar is regular if it has rules of form A -> a or A -> aB or A -> ε where ε is a special symbol called NULL.

Right linear grammar: A-> xB or A->x

linear

Left linear grammar: A->Bx or A->x

The set of all strings generated from a is said to be language of that grammar.

Context free grammar cfg

Definition – A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple **(N, T, P, S)** where

- **N** is a set of non-terminal symbols.

Scanned with CamScanner

- **T** is a set of terminals where **N ∩ T = NULL.**

tuples

- **P** is a set of rules, **P: N → (N ∪ T)***, i.e., the left-hand side of the production rule **P** does have any right context or left context.
- **S** is the start symbol.

Derivation tree

derivation tree

A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a context-free grammar.

Representation Technique

- **Root vertex** – Must be labeled by the start symbol.
- **Vertex** – Labeled by a non-terminal symbol.
- **Leaves** – Labeled by a terminal symbol or ε.

Sentential Form and Partial Derivation Tree

sentential

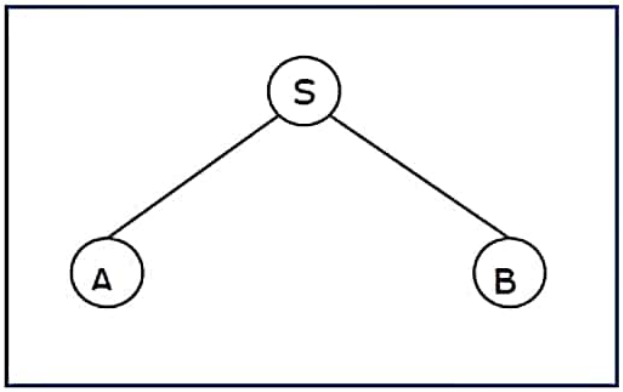
A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

Example

If in any CFG the productions are –

S → AB, A → aaA | ε, B → Bb| ε

the partial derivation tree can be the following –



partial

If a partial derivation tree contains the root S, it is called a **sentential form**. The above sub-tree is also in sentential form.

Scanned with CamScanner

tree

leftmost

rightmost

derivation

Leftmost and Rightmost Derivation of a String

- **Leftmost derivation** – A leftmost derivation is obtained by applying production to the leftmost variable in each step.
- **Rightmost derivation** – A rightmost derivation is obtained by applying production to the rightmost variable in each step.

Ambiguous grammar

ambiguos

A CFG is said to ambiguous if there exists more than one derivation tree for the given input string i.e., more than one **L**eft**M**ost **D**erivation **T**ree (LMDT) or **R**ight**M**ost **D**erivation **T**ree (RMDT).

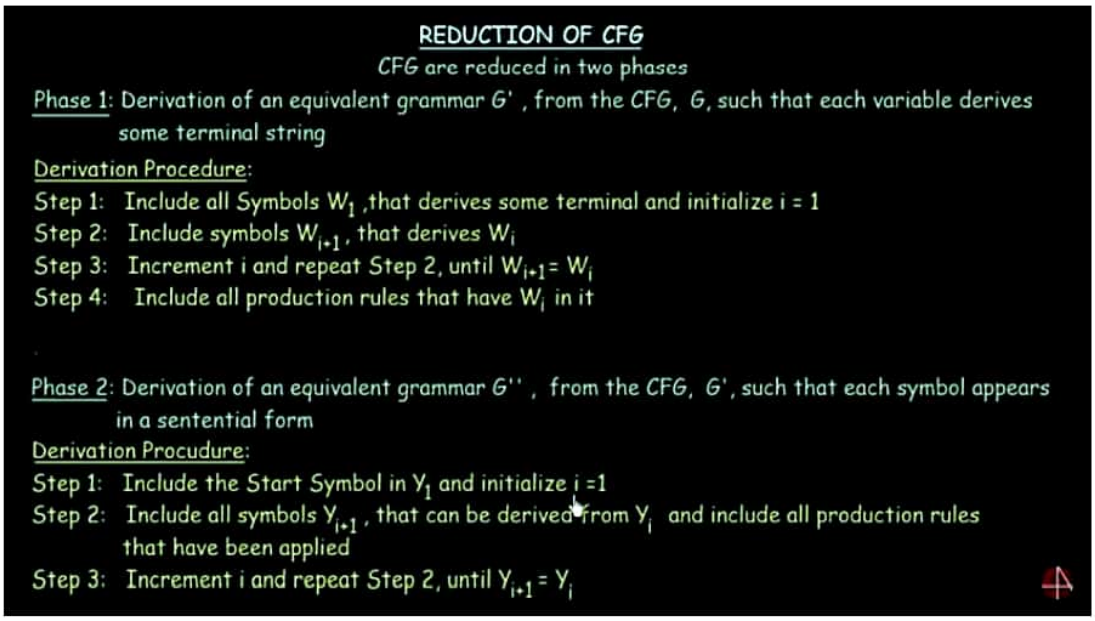
Simplification of cfg

simplification of cfg

in a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Besides, there may be some null productions and unit productions. Elimination of these productions and symbols is called **simplification of CFGs**. Simplification essentially comprises of the following steps –

- Reduction of CFG
- Removal of Unit Productions
- Removal of Null Productions

Reduction of cfg



reduction

Scanned with CamScanner

Removal of unit productions removal of unit

Simplification of Context Free Grammar

Removal of Unit Productions

Any Production Rule of the form A→B where A, B ∈ Non Terminals is called Unit Production

Procedure for Removal

Step 1: To remove A→B, add production A→x to the grammar rule whenever B→x occurs in the grammar. [x ∈ Terminal, x can be Null]

Step 2: Delete A→B from the grammar.

Step 3: Repeat from Step 1 until all Unit Productions are removed.

Example: Remove Unit Productions from the Grammar whose production rule is given by
P: S→XY, X→a, Y→Z|b, Z→M, M→N, N→a

Y→Z, Z→M, M→N

1) Since N→a, we add M→a

P: S→XY, X→a, Y→Z|b, Z→M, M→a, N→a

Removal of null productions removal null

Simplification of Context Free Grammar

Removal of Null Productions

In a CFG, a Non-Terminal Symbol 'A' is a nullable variable if there is a production A→∈ or there is a derivation that starts at 'A' and leads to ∈. (Like A→.....→∈)

Procedure for Removal:

Step 1: To remove A→∈, look for all productions whose right side contains A

Step2: Replace each occurences of 'A' in each of these productions with ∈

Step 3: Add the resultant productions to the Grammar

Example: Remove Null Productions from the following Grammar
S→ABAC, A→aA|∈, B→bB|∈, C→c

A→∈ , B→∈

1) To eliminate A→∈

S

Scanned with CamScanner

Chomsky normal form

cnf

In Chomsky Normal Form (CNF) we have a restriction on the length of RHS; which is; elements in RHS should either be two variables or a Terminal.
A CFG is in Chomsky Normal Form if the productions are in the following forms:

$A \rightarrow a$
$A \rightarrow BC$

where A, B and C are non-terminals and a is a terminal

Steps

Steps to convert a given CFG to Chomsky Normal Form:

Step 1: If the Start Symbol S occurs on some right side, create a new Start Symbol S' and a new Production $S' \rightarrow S$.

Step 2: Remove Null Productions. (Using the Null Production Removal discussed in previous Lecture)

Step 3: Remove Unit Productions. (Using the Unit Production Removal discussed in previous Lecture)

Step 4: Replace each Production $A \rightarrow B_1 \ldots\ldots B_n$ where $n > 2$, with $A \rightarrow B_1 C$ where $C \rightarrow B_2 \ldots\ldots B_n$ Repeat this step for all Productions having two or more Symbols on the right side.

Step 5: If the right side of any Production is in the form $A \rightarrow aB$ where 'a' is a terminal and A and B are non-terminals, then the Production is replaced by $A \rightarrow XB$ and $X \rightarrow a$. Repeat this step for every Production which is of the form $A \rightarrow aB$

Gnf greibach normal form

gnf

Greibach Normal Form

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

$A \rightarrow b$
$A \rightarrow bC_1C_2 \ldots\ldots C_n$

where $A, C_1, \ldots\ldots, C_n$ are Non-Terminals and b is a Terminal

Steps to convert a given CFG to GNF:

Step 1: Check if the given CFG has any Unit Productions or Null Productions and Remove if there are any (using the Unit & Null Productions removal techniques discussed in the previous lecture)

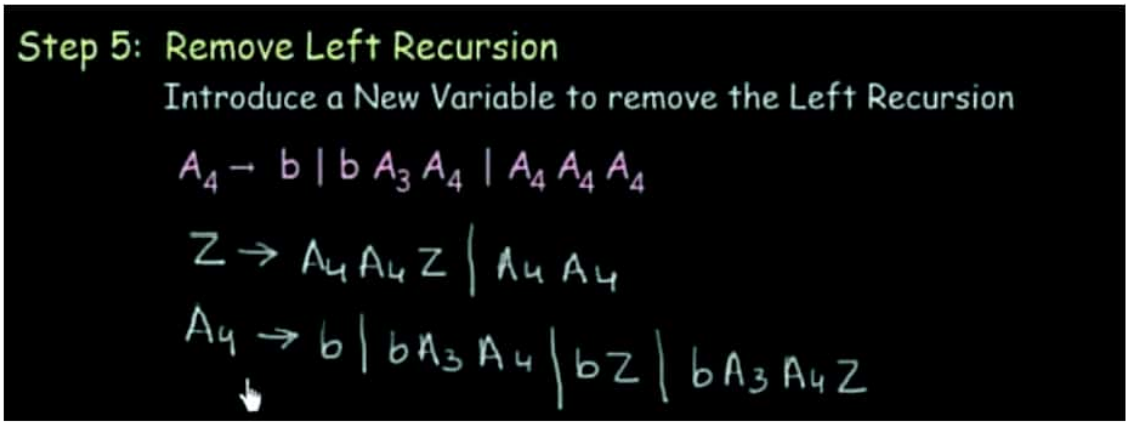
Step 2: Check whether the CFG is already in Chomsky Normal Form (CNF) and convert it to CNF if it is not. (using the CFG to CNF conversion technique discussed in the previous lecture)

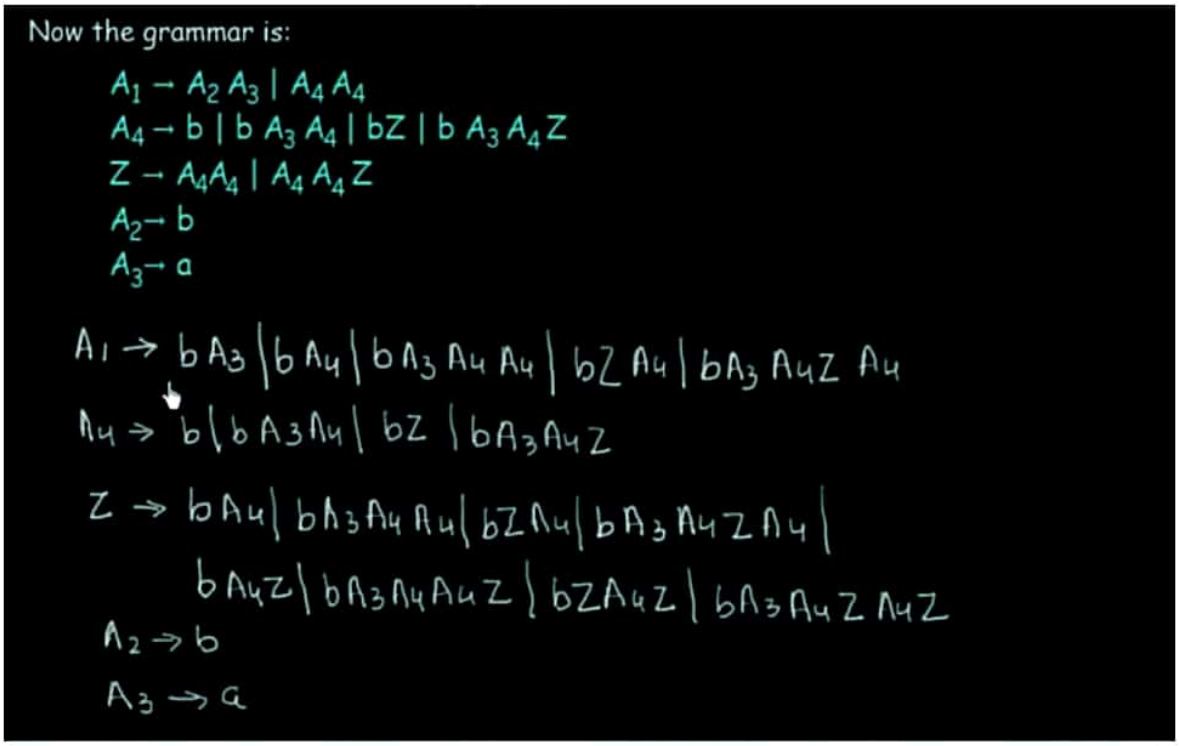
Step 3: Change the names of the Non-Terminal Symbols into some A_i in ascending order of i

Scanned with CamScanner

Removal of left recursion in gnf

left recusion





regular grammar or context free grammar me kya difference hai?

Regular Expressions	Context-free grammar

table3

Lexical rules are quite simple in case of Regular Expressions.	Lexical rules are difficult in case of Context free grammar.
Notations in regular expressions are easy to understand.	Notations in Context free grammar are quite complex.

Scanned with CamScanner

Regular Expressions	Context-free grammar
A set of string is defined in case of Regular Expressions.	In Context free grammar the language is defined by the collection of productions.
It is easy to construct efficient recognizer from Regular Expressions.	By using the context free grammar, it is very difficult to construct the recognizer.
There is proper procedure for lexical and syntactical analysis in case of Regular Expressions.	There is no specific guideline for lexical and syntactic analysis in case of Context free grammar.
Regular Expressions are most useful for describing the structure of lexical construct such as identifiers, constant etc.	Context free grammars are most useful in describing the nested chain structure or syntactic structure such as balanced parenthesis, if else etc. and these can't be define by Regular Expression.

pumping lemma

Pumping Lemma for Context-free Languages (CFL)

Pumping Lemma for CFL states that for any Context Free Language L, it is possible to find two substrings that can be 'pumped' any number of times and still be in the same language. For any language L, we break its strings into five parts and pump second and fourth substring.

Pumping Lemma, here also, is used as a tool to prove that a language is not CFL. Because, if any one string does not satisfy its conditions, then the language is not CFL.

Thus, if L is a CFL, there exists an integer n, such that for all $x \in L$ with $|x| \geq n$, there exists u, v, w, x, y $\in \Sigma^*$, such that x = uvwxy, and

(1) $|vwx| \leq n$

(2) $|vx| \geq 1$

(3) for all $i \geq 0$: $uv^iwx^iy \in L$

Scanned with CamScanner

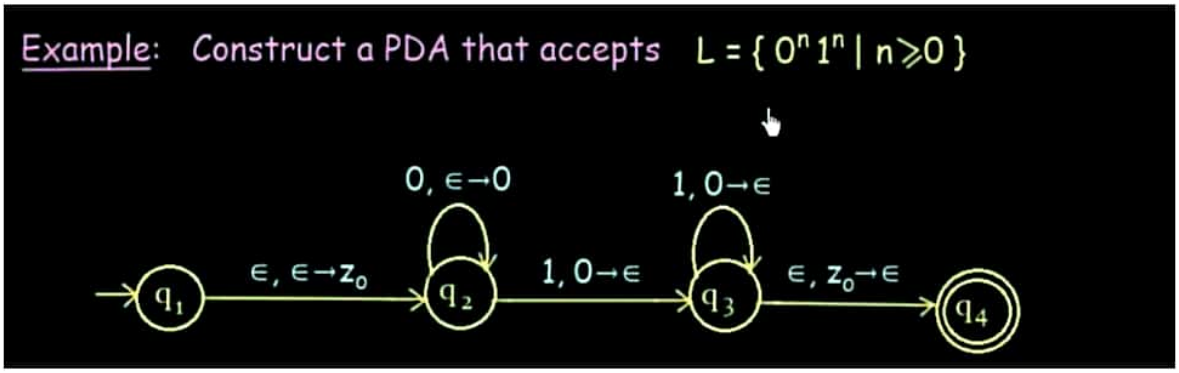
pushdown automata pda pushdown automata

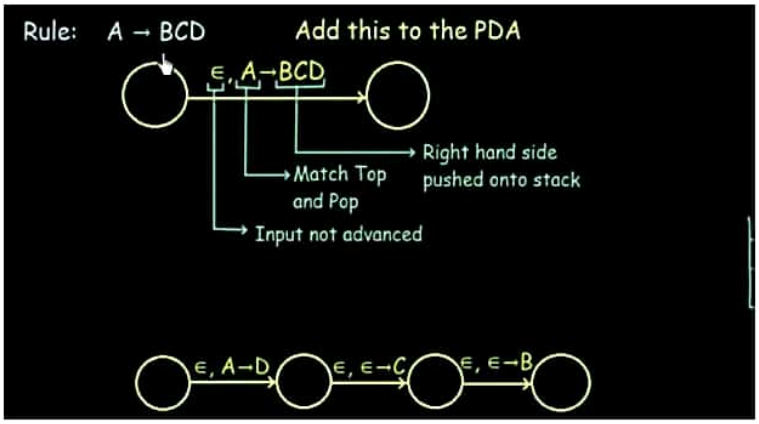
Pushdown Automata is a finite automata with extra memory called stack which helps Pushdown automata to recognize Context Free Languages.

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from stack) gamma
- q0 is the initial state tuples
- Z is the initial pushdown symbol (which is initially present in stack)
- F is the set of final states
- δ is a transition function which maps Q x {Σ ∪ ∈} x Γ into Q x Γ*. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

Pda example





Scanned with CamScanner

pda

stack

δ = The Transition Function

q_0 = The Start State

z_0 = The Start Stack Symbol

F = The set of Final / Accepting States

Press Esc to exit full screen

δ takes as argument a triple δ (q, a, X) where:

(i) q is a State in Q

(ii) a is either an Input Symbol in Σ or a = ∈

(iii) X is a Stack Symbol, that is a member of Γ

The output of δ is finite set of pairs (p, γ) where:

p is a new state

γ is a string of stack symbols that replaces X at the top of the stack

Eg. If γ = ∈ then the stack is popped

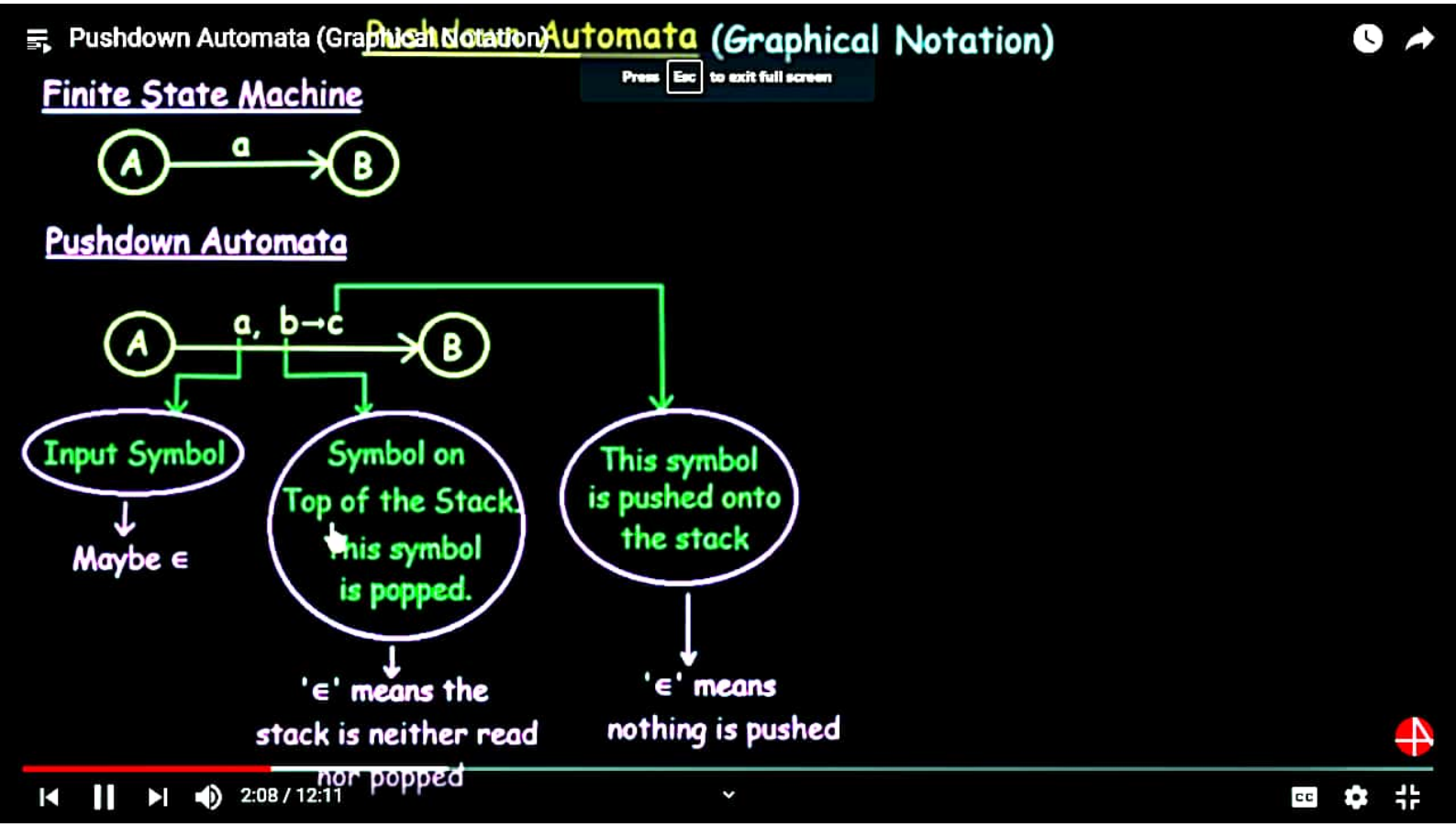
If γ = X then the stack is unchanged

If γ = YZ then X is replaced by Z and Y is pushed onto the stack

Scanned with CamScanner

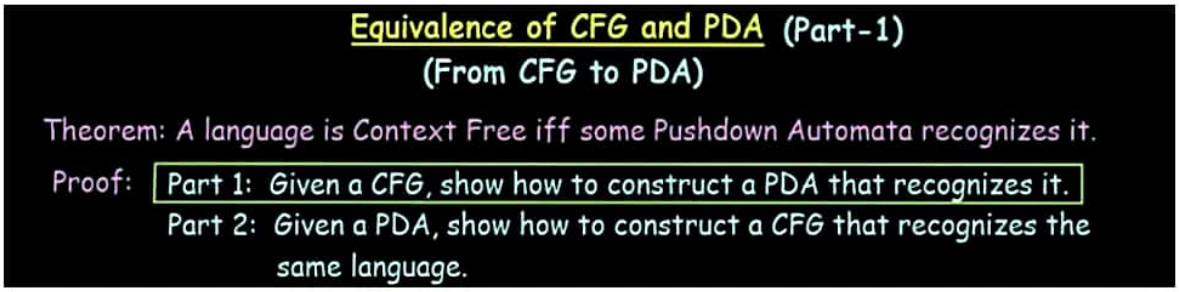
stack

transition



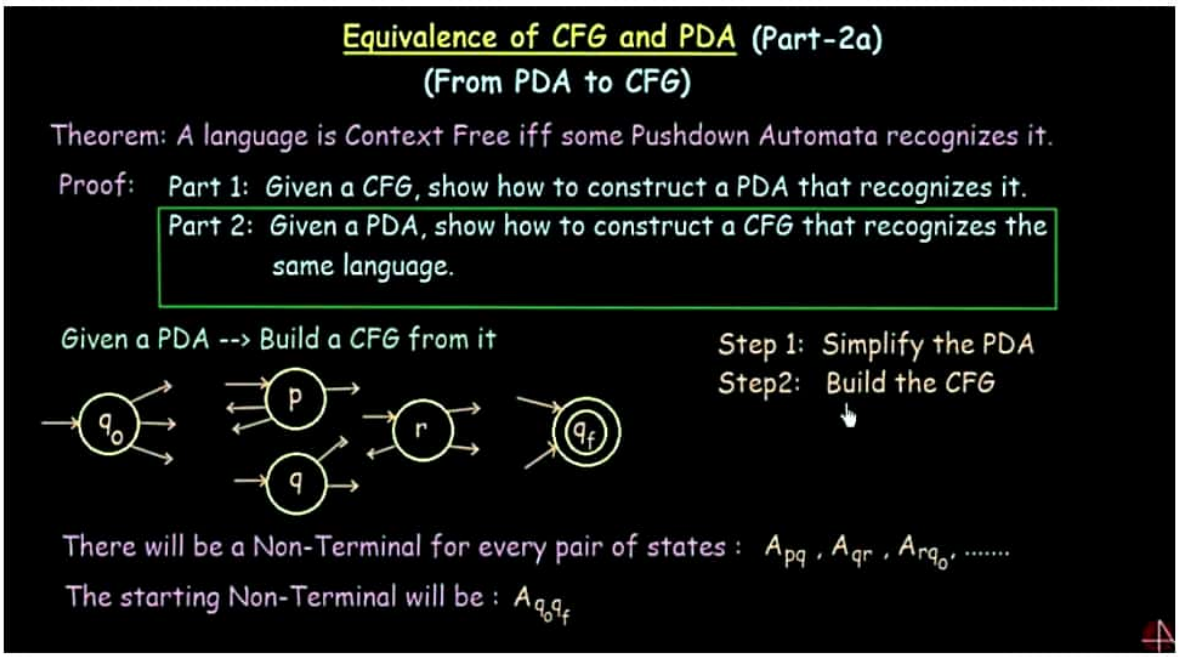
Scanned with CamScanner

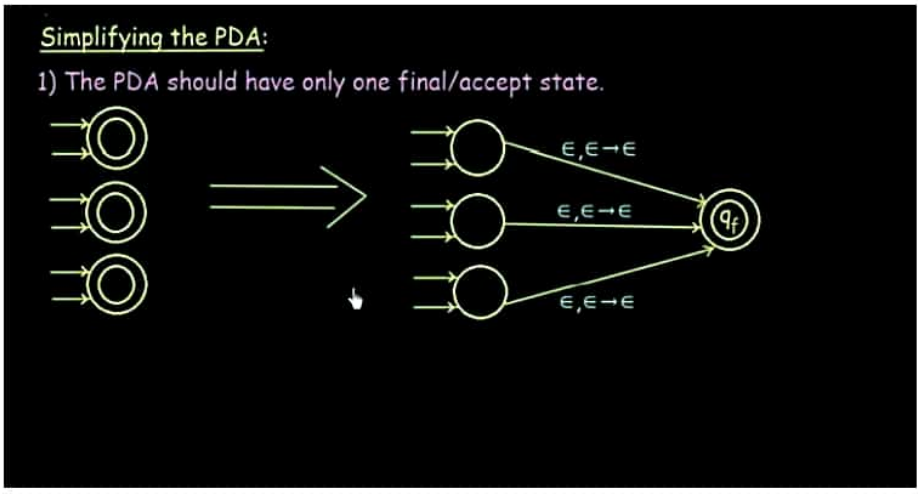
Equivalence of pda and cfg



equivalence pda to cfg cfg

Pda to cfg





Scanned with CamScanner

Misc

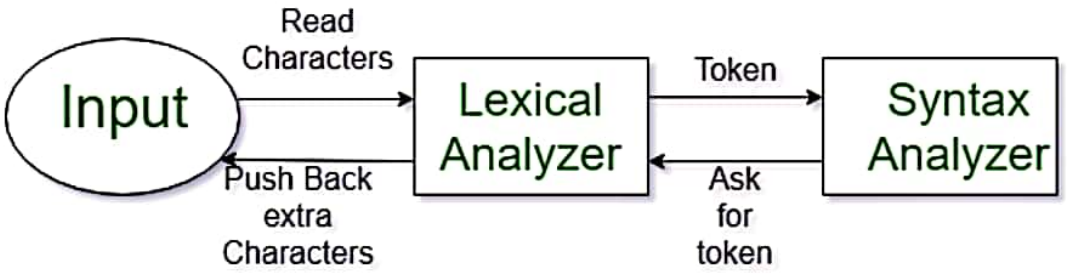
lexical analysis

Lexical analysis

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of **Tokens**.

tokens

- Lexical Analysis can be implemented with the Deterministic finite Automata.
- The output is a sequence of tokens that is sent to the parser for syntax analysis



practical applications of finite automata

The **Applications** of these Automata are given as follows:

1. Finite Automata (FA) –

practical application

- For the designing of lexical analysis of a compiler.
- For recognizing the pattern using regular expressions.
- For the designing of the combination and sequential circuits using Mealy and Moore Machines.
- Used in text editors.
- For the implementation of spell checkers.

2. Push Down Automata (PDA) –

- For designing the parsing phase of a compiler (Syntax Analysis).
- For implementation of stack applications.
- For evaluating the arithmetic expressions.

Scanned with CamScanner

- For solving the Tower of Hanoi Problem.

3. Linear Bounded Automata (LBA) –

- For implementation of genetic programming.
- For constructing syntactic parse trees for semantic analysis of the compiler.

4. Turing Machine (TM) **–**

- For solving any recursively enumerable problem.
- For understanding complexity theory.
- For implementation of neural networks.
- For implementation of Robotics Applications.
- For implementation of artificial intelligence.

re

recursive enumerable

Recursive Enumerable (RE) or Type -0 Language

RE languages or type-0 languages are generated by type-0 grammars. An RE language can be accepted or recognized by Turing machine 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 TM can loop forever for the strings which are not a part of the language. RE languages are also called as Turing recognizable languages.

recursive language rec

Recursive Language (REC)

A recursive language (subset of RE) can be decided by Turing machine 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. e.g.; L= $\{a^n b^n c^n | n>=1\}$ 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 TM will always halt in this case. REC languages are also called as Turing decidable languages. The relationship between RE and REC languages can be shown in Figure 1.

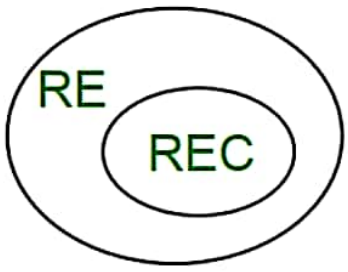


Figure 1

Scanned with CamScanner

True or false ?

Ki humare paas ek automata h A^P where P is prime

Yes using turing machine

prime

Prove ye cfl h ya nhi with pumping lemma

Nhi hai

use of NFA in UNIX commands

unix

example: ls *.java gives all files with .java

union of two dfas

union

is possible, as the regular expression is closed under union.

cellular automata

cellular automata

different normal form(dekh lena bs cnf ya gnf hai ya kuch aur bhi)

cfg pr union wagerah se kaunsi grammar bnti hai

union, concat, kleene closure of cfg = cfg,

a^p where p is prime is cfg or not using pumping lemma

Recursive enummerable set k subsets saare recursive honge ?

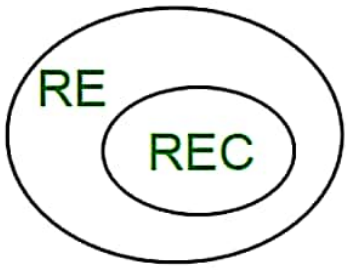


Figure 1

recursive enumerable

True or false ? (false ig)

Scanned with CamScanner

emptiness in dfa

Grammar for palindrome (cfg imo)

wwr wawr

palindrome

is dfa valid without final states

yes

Computable properties of regular language

1) How many states will be there in a DFA that accepts 3 or 5

2

What is membership problem?

Given a word w and a language L, we want to check if w∈L. This is called the membership problem.

membership problem

3) How to solve it (CYK)

4) Explain it in detail (CYK)

cyk

CYK algorithm is a parsing algorithm for context free grammar.

In order to apply CYK algorithm to a grammar, it must be in Chomsky Normal Form. It uses a dynamic programming algorithm to tell whether a string is in the language of a grammar.

Time and Space Complexity :

Time Complexity –

O(n3.|G|)

Where |G| is the number of rules in the given grammar.

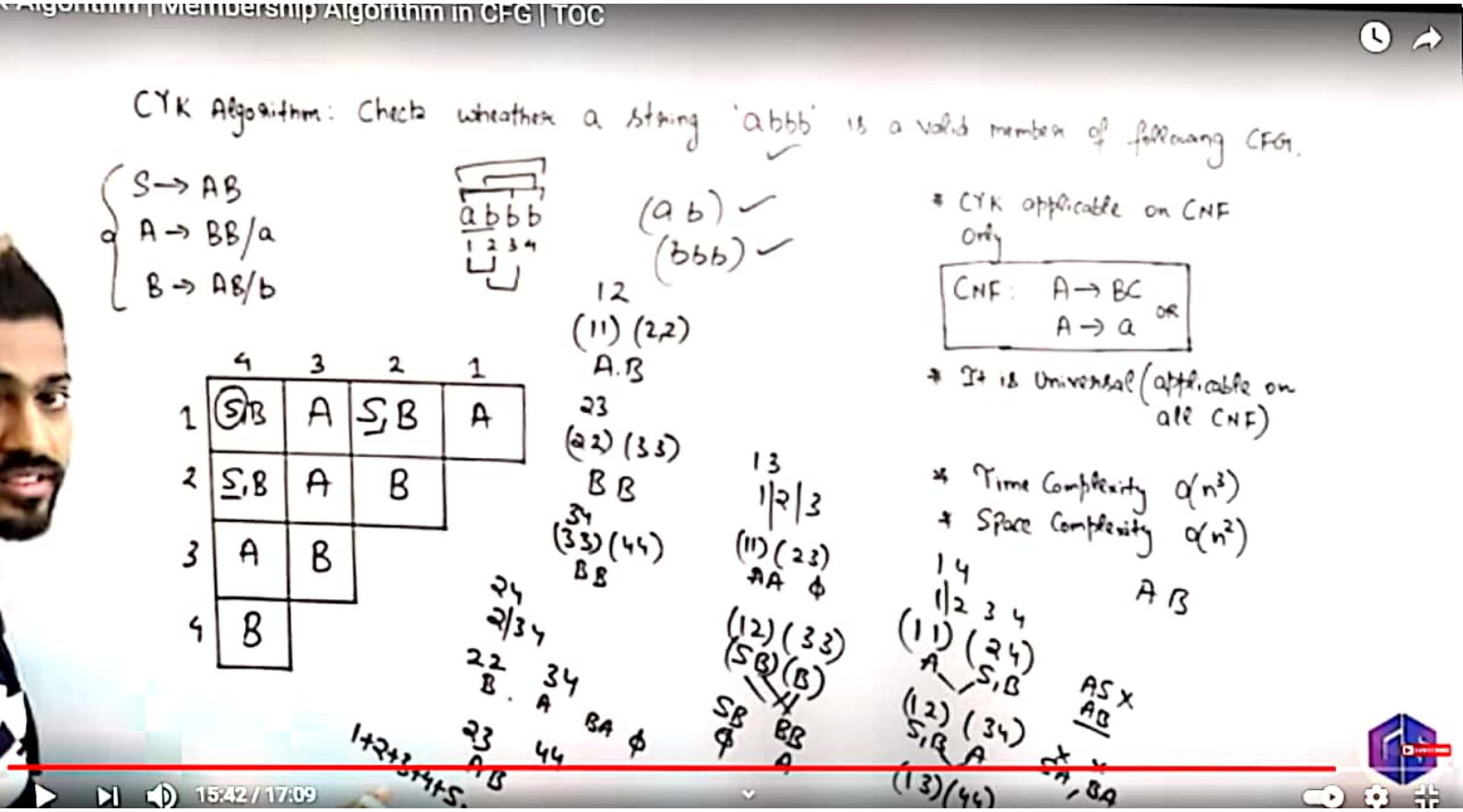
Space Complexity –

O(n2)

CYK Algorithm for Context Free Grammar - GeeksforGeeks

Scanned with CamScanner

similar to MCM DP



Scanned with CamScanner