Define a context-free grammar (CFG) and explain its components.
A CFG is defined as G=(V,T,P,S)G = (V, T, P, S)G=(V,T,P,S), where:

- V: A finite set of variables (non-terminals).
- T: A finite set of terminal symbols.
- P: A finite set of production rules of the form A→αA \to \alphaA→α, where A∈VA \in VA∈V and α∈(V∪T)∗\alpha \in (V \cup T)^*α∈(V∪T)∗.
- S: The start symbol S∈VS \in VS∈V.

What is parse tree
A parse tree is a hierarchical tree representation of the derivation of a string from a CFG. It shows how the start symbol derives the string step-by-step through production rules.
Parse trees are used to check membership of a string in a language and for syntax analysis in compilers.

Left Most Derivation (LMD) and Derivation Tree : Leftmost derivation of a string from starting symbol S is done by replacing leftmost non-terminal symbol by RHS of corresponding production rule. For example, the leftmost derivation of string abab from grammar G above is done as :
<u>S</u> => a<u>S</u>b => ab<u>S</u>ab => abab

Right Most Derivation (RMD) : Rightmost derivation of a string from starting symbol S is done by replacing rightmost non-terminal symbol by RHS of corresponding production rule. e.g.; The rightmost derivation of string abab from grammar G above is done as :
<u>S</u> => S<u>S</u> => Sa<u>S</u>b => <u>S</u>ab => a<u>S</u>bab => abab

CFG vs. Regular Grammar:
Regular grammars allow only productions of the form
A→aBA \ aBA→aB or A→aA \ aA→a,
whereas CFGs allow broader production rules, making CFGs more expressive

Parameter	Context Free Grammar	Regular Grammar
Type	Type-2	Type-3
Recognizer	Push-down automata.	Finite State Automata
Rules	Productions are of the form: A->B; A∈N(Non-Terminal) B∈V^* (Any string)	Productions are of the form: V –> VT / T (left-linear grammar) (or) V –> TV /T (right-linear grammar)
Restriction	Less than Regular Grammar	More than any other grammar
Right-hand Side	The right-hand side of production has no restrictions.	The right-hand side of production should be either left linear or right linear.
Set Property	Super Set of Regular Grammar	Subset of Context Free Grammar
Intersection	Intersection of two CFL need not be a CFL	Intersection of two RG is a RG.
Complement	They are not closed under complement	Closed under complement
Range	The range of languages that come under CFG is wide.	The range of languages that come under RG is less than CFG.
Examples	S –> AB;A –> a;B –> b	S -> aS | bS | ∈

Applications of CFGs

Programming Languages:CFGs form the basis for the syntax of programming languages, where rules describe valid code structures (e.g., conditionals, loops).

Natural Language Processing (NLP):CFGs are used to model natural language syntax, though they are often augmented with additional constructs to handle ambiguity and context sensitivity.

XML Parsing:XML documents are validated against context-free grammars (via DTDs or XSDs) to ensure they conform to expected structure. OR

Context Free Grammars are used in Compilers (like GCC) for parsing. In this step, it takes a program (a set of strings).

Context Free Grammars are used to define the High Level Structure of a Programming Language.

Every Context Free Grammar can be converted to a Parser which is a component of a Compiler that identifies the structure of a Program and converts the Program into a Tree.

Document Type Definition in XML is a Context Free Grammar which describes the HTML tags and the rules to use the tags in a nested fashion.

Ambiguous Context Free Grammar : A context free grammar is called ambiguous if there exists more than one LMD or more than one RMD for a string which is generated by grammar. There will also be more than one derivation tree for a string in ambiguous grammar. The grammar described above is ambiguous because there are two derivation trees (Figure 1 and Figure 2). There can be more than one RMD for string abab which are:
S => SS => SaSb => Sab => aSbab => abab
S => aSb => abSab => abab

Ambiguous Context Free Languages : A context free language is called ambiguous if there is no unambiguous grammar to define that language and it is also called inherently ambiguous Context Free Languages.
eg- L={anbncm} U {anbmcm}

Note :

- If a context free grammar G is ambiguous, language generated by grammar L(G) may or may not be ambiguous.
- It is not always possible to convert ambiguous CFG to unambiguous CFG. Only some ambiguous CFG can be converted to unambiguous CFG.
- There is no algorithm to convert ambiguous CFG to unambiguous CFG.
- There always exists a unambiguous CFG corresponding to unambiguous CFL.
- Deterministic CFL are always unambiguous and are parsed by LR parsers.

What are the steps to simplify a CFG?
Remove Useless Symbols:

- Identify symbols (non-terminals or terminals) that do not lead to terminal strings (non-generating symbols) and remove them.Eliminate symbols that are not reachable from the start symbol.

Remove ϵ\epsilonϵ-Productions:

- Identify nullable variables (those that derive ϵ\epsilonϵ) and remove them, except when ϵ\epsilonϵ is part of the language.

Remove Unit Productions:

- Eliminate productions of the form A→BA \to BA→B, where both AAA and BBB are non-terminals.

Remove Redundant Rules:

- Remove duplicate rules or simplify overly complex derivations.

Why is CFG simplification important?

- Simplified CFGs are easier to analyze, parse, and convert to other forms (e.g., normal forms).
- Simplified grammars reduce computational complexity and improve efficiency in applications like parsing and language recognition.

Steps for converting CFG into GNF

- Step 1: Convert the grammar into CNF. :If the given grammar is not in CNF, convert it into CNF. You can refer the following topic to convert the CFG into CNF: Chomsky normal form
- Step 2: If the grammar exists left recursion, eliminate it. :If the context free grammar contains left recursion, eliminate it. You can refer the following topic to eliminate left recursion: Left Recursion
- Step 3: In the grammar, convert the given production rule into GNF form.

If any production rule in the grammar is not in GNF form, convert it.

What is Greibach Normal Form (GNF)?

he Greibach normal form is referred to as GNF. A context-free grammar (CFG) is in Greibach normal form (GNF) if and only all of its production rules meet one of the criteria listed below:

- A non-terminal that generates a terminal. For instance, X → x.
- A start symbol that generates ε. For instance, S → ε.
- A non-terminal that generates a terminal followed by any number of non-terminals. For instance, S → xXSY.

What is Chomsky Normal Form (CNF)? How is a CFG converted into CNF?

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

- Start symbol generating ε. For example, A → ε.
- A non-terminal generating two non-terminals. For example, S → AB.
- A non-terminal generating a terminal. For example, S → a.

: What is a regular grammar?

- A regular grammar is a CFG where all production rules are of one of the following forms:
    1. A→aBA \to aBA→aB, where A,B∈VA, B \in VA,B∈V and a∈Ta \in Ta∈T,
    2. A→aA \to aA→a, where a∈Ta \in Ta∈T,
    3. A→ϵA \to \epsilonA→ϵ (for right-linear grammars).
- Regular grammars can be right-linear or left-linear.

RLG vs LLG

The main difference between right linear grammar (RLG) and left linear grammar (LLG) in Theory of Computation (TOC) is the position of the non-terminal symbol in relation to the terminal symbol:

- **RLG**: The non-terminal symbol is on the right of the terminal symbol.
- **LLG**: The non-terminal symbol is on the left of the terminal symbol.

Production rules in RLG are written in reverse order of how they are used to generate strings, while in LLG they are written in the same order. To convert an RLG to an LLG, you can reverse the order of the production rules

Define Useless productions – The productions that can never take part in derivation of any string , are called useless productions. Similarly , a variable that can never take part in derivation of any string is called a useless variable

Defince Epislon production -a production in a context-free grammar (CFG) where the right-hand side is the empty string, represented by the symbol epsilon.

Define Unit production-

Pushdown Automata

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

Instantaneous Description (ID)

Instantaneous Description (ID) is an informal notation of how a PDA "computes" an input string and makes a decision whether that string is accepted or rejected.

An ID is a triple (q, w, α), where:

1. q is the current state.
2. w is the remaining input.
3. α is the stack contents, top at the left.

Why NPDA is more powerful than DPDA?

NDPA is more powerful than DPDA because we can add more transitions to it. It is possible for every language to add a transition. For some languages, we can construct DPDA there exist an NPDA but there are some languages that are accepted by NPDA but are not by DPDA. This is said to be powerful when it accepts more sets of languages than other automata.

In fact, it is more powerful than DFA(Deterministic finite automata) and NFA(Non-deterministic finite automata) also because, In the case of DFA and NFA, they are equivalent in power. So for every language accepted by DFA there exist an NFA and Vice-Versa. There is not any language for which we construct NFA but not DFA. Hence, we can't convert NPDA to DPDA always and we can convert NFA to equivalent DFA always.

PDA Acceptance

A language can be accepted by Pushdown automata using two approaches:

1. Acceptance by Final State: The PDA is said to accept its input by the final state if it enters any final state in zero or more moves after reading the entire input.

Let $P = (Q, \sum, \Gamma, \delta, q0, Z, F)$ be a PDA. The language acceptable by the final state can be defined as:

$$L(PDA) = \{w \mid (q0, w, Z) \vdash^* (p, \varepsilon, \varepsilon), q \in F\}$$

2. Acceptance by Empty Stack: On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty.

Let $P = (Q, \sum, \Gamma, \delta, q0, Z, F)$ be a PDA. The language acceptable by empty stack can be defined as:

$$N(PDA) = \{w \mid (q0, w, Z) \vdash^* (p, \varepsilon, \varepsilon), q \in Q\}$$

Equivalence of Acceptance by Final State and Empty Stack

- If L = N(P1) for some PDA P1, then there is a PDA P2 such that L = L(P2). That means the language accepted by empty stack PDA will also be accepted by final state PDA.
- If there is a language L = L (P1) for some PDA P1 then there is a PDA P2 such that L = N(P2). That means language accepted by final state PDA is also acceptable by empty stack PDA.

Difference between NPDA and DPDA:

S. No	DPDA(Deterministic Pushdown Automata)	NPDA(Non-deterministic Pushdown Automata)
1.	It is less powerful than NPDA. Example: We can only construct DPDA for odd-length palindromes and not for even length palindromes.	It is more powerful than DPDA. Example: NPDA can be constructed for both even-length and odd-length palindromes.
2.	It is possible to convert every DPDA to a corresponding NPDA.	It is not possible to convert every NPDA to a corresponding DPDA.
3.	The language accepted by DPDA is a subset of the language accepted by NDPA.	The language accepted by NPDA is not a subset of the language accepted by DPDA.
4.	The language accepted by DPDA is called DCFL(Deterministic Context-free Language) which is a subset of NCFL(Non-deterministic Context-free Language) accepted by NPDA.	The language accepted by NPDA is called NCFL(Non-deterministic Context-free Language).
5.	There is only one state transition from one state to another state for an input symbol.	There may or maynot be more than one state transition from one state to another state for same input symbol.

CFG to PDA Conversion

CFG → PDA

Algorithm

1) Let q_0 be the start state and Z_0 be the initial symbol in stack. In state q_0 without consuming any input, change the state to q_1 and push the start symbol S on the stack using the transition,

$$\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$$

2) (i) For every $A \in V$, introduce the transition $\delta(q_1, \epsilon, A) = (q_1, \alpha)$ for all the productions of the form $A \to \alpha$ in G.

(ii) For every $A \in \Sigma$, introduce the transition, $\delta(q_1, a, a) = (q_1, \epsilon)$

3) Finally in state q_1, without consuming any i/p, change the state to q_f.

$$\delta(q_1, \epsilon, Z_0) = (q_f, Z_0)$$

PDA to CFG Conversion

PDA → CFG

Algorithm

1) Assume there is a transition from q_i to q_j on i/p symbol 'a' and top of stack 'Z'. The non terminal corresponding to this transition is given by, $(q_i Z q_j)$

→ single variable

2) If there is a transition of the form $\delta(q_i, a, Z) = (q_j, AB)$, then introduce the production of the form : $(q_i Z q_k) \to a(q_j A q_l)(q_l B q_k)$, where q_l, q_k are all possible states of Q

3) If there is a transition of the form $\delta(q_i, a, Z) = (q_j, \epsilon)$, then introduce the production of the form $(q_i Z q_j) \to a$.

Steps for converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. If the start symbol T is at the right-hand side of any production, create a new production as:

S1 → S

Step 2: In the grammar, remove the null, unit and useless productions. You can refer to the Simplification of CFG.

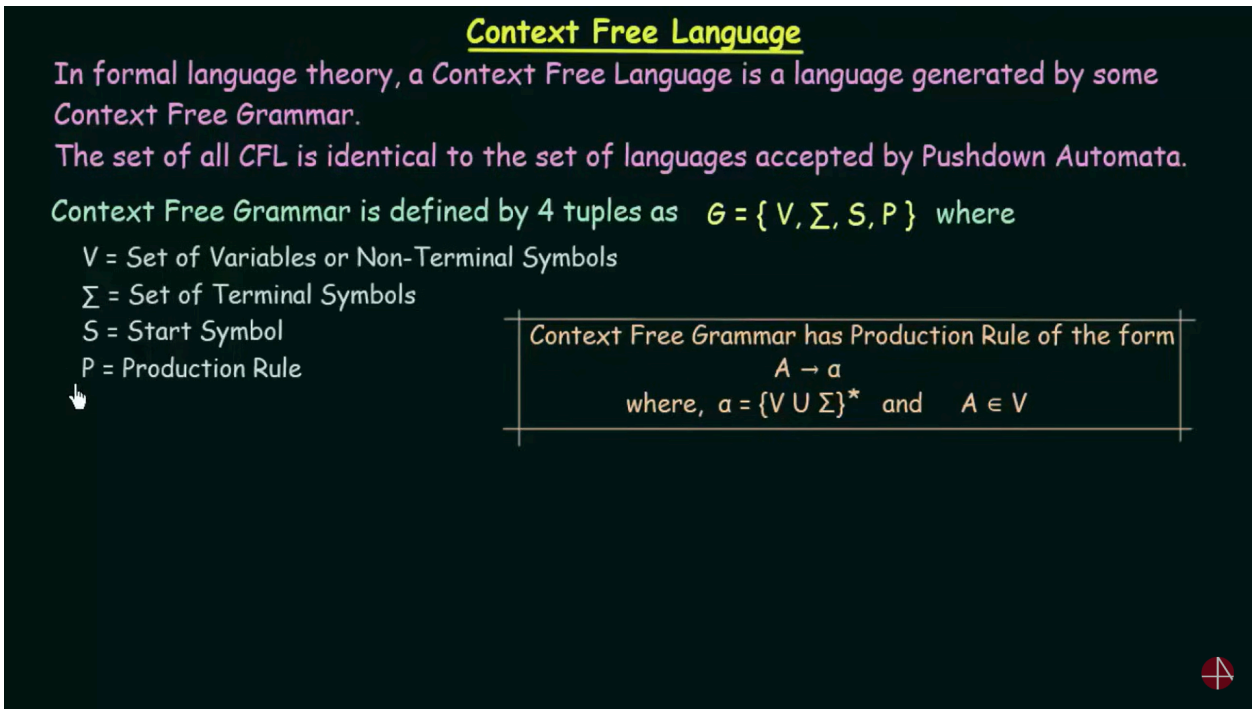
Step 3: Eliminate terminals from the RHS of the production if they exist with other non-terminals or terminals. For example, production S → aA can be decomposed as:

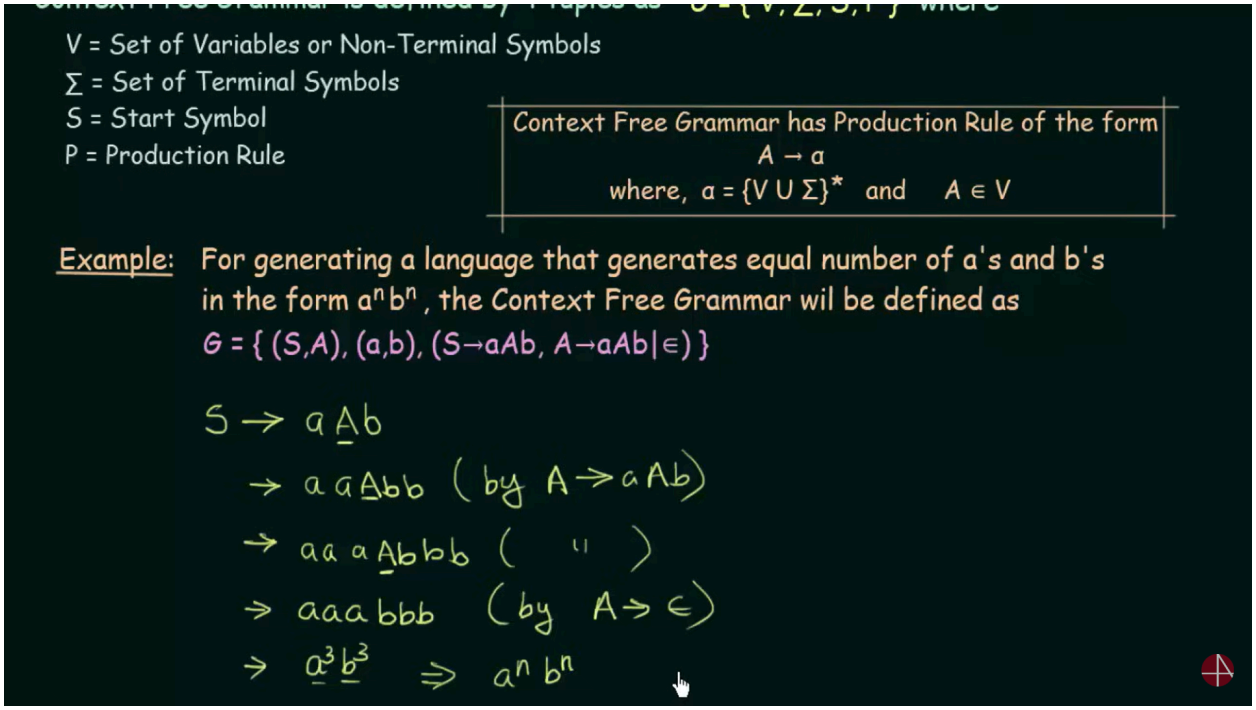
S → RA
R → a

Step 4: Eliminate RHS with more than two non-terminals. For example, S → ASB can be decomposed as:

S → RS
R → AS

Language of CFG -





The language of a context-free grammar (CFG) is a context-free language (CFL).

Language of PDA-The language of a pushdown automaton (PDA) can be defined in two ways:

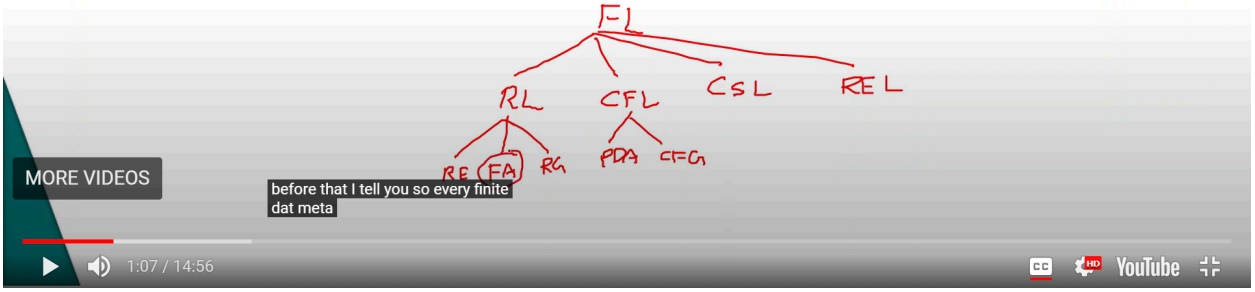
- **Acceptance by final state: The PDA accepts an input string if it can read the entire string and enter a final state.**
-
- **Acceptance by empty stack: The PDA accepts an input string if it can empty its stack.**

Both methods define the same class of languages, and a PDA that uses one method can be converted to use the other.

2.6 The Languages of PDA
Watch later Share

Topic: The Languages of PDA

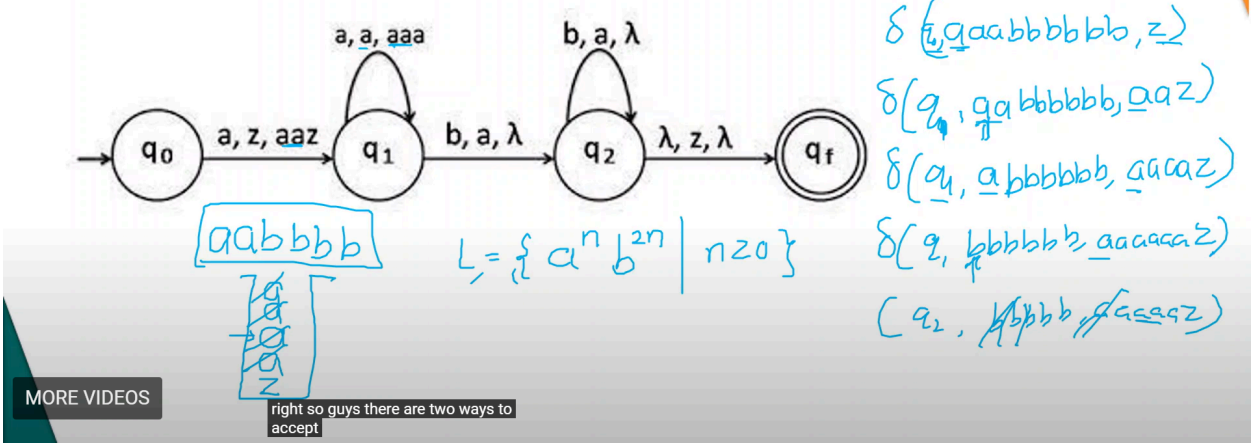
Previous Topic: PDA Examples



2.6 The Languages of PDA
To exit full screen, press Esc
Watch later Share

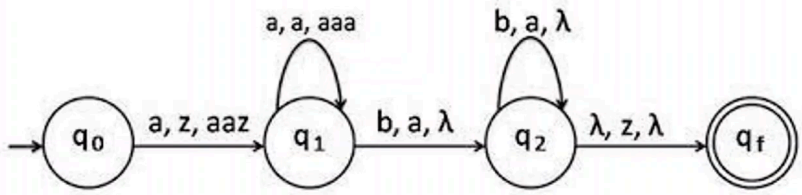
Push Down Automata: Language

Find the Language accepted by the PDA given Below



Push Down Automata: Language

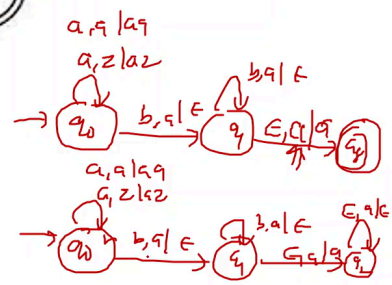
Find the Language accepted by the PDA given Below



$L = \{a^n b^m \mid n > m\}$

$L = \{a, aab, aaaabb, aaaaab\}$

push pop



are not worrying about how many a in the stack but we are saying it is accepted

2.6 The Languages of PDA

To exit full screen, press Esc

PDA: Acceptance by Final State

1. We say that a run

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

is accepted via final state if $q_n \in F$ and $w_n = \varepsilon$.

2. We say that a word w is accepted via final states if there exists a run of P over w that is accepted via final state.

3. We write $L(P)$ for the set of words accepted via final states.

4. In other words,

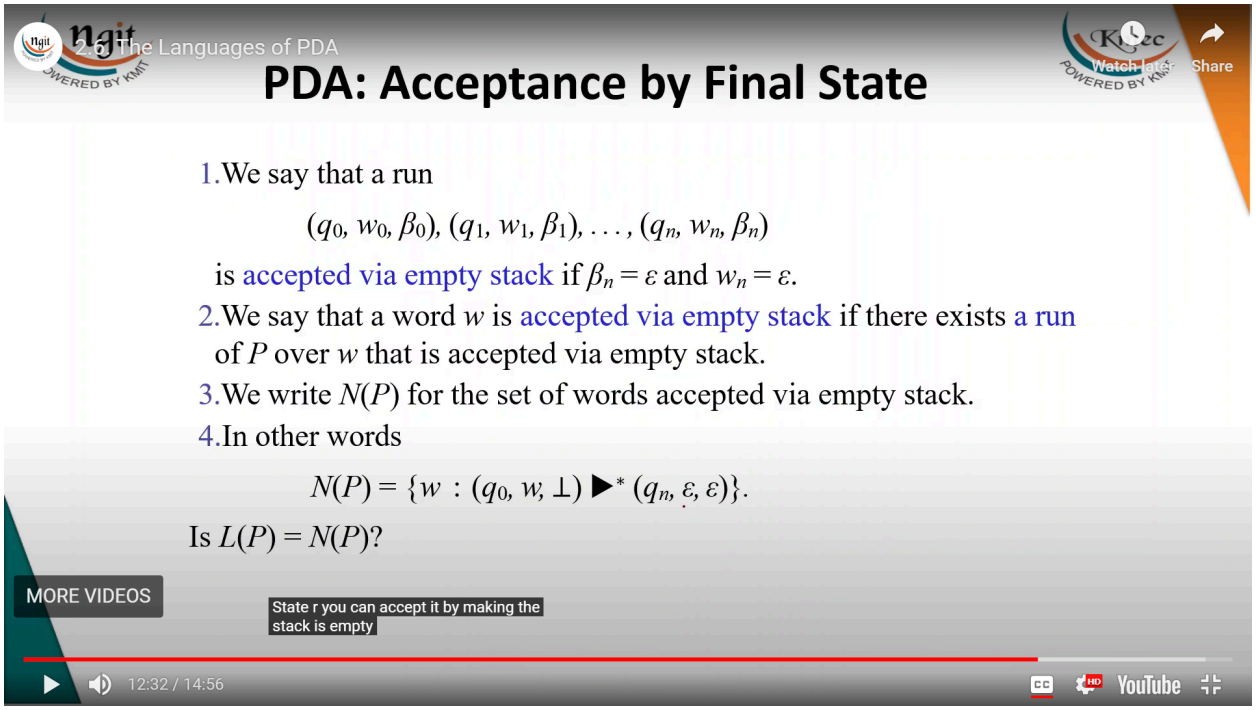
$$L(P) = \{w : (q_0, w, \perp) \blacktriangleright^* (q_n, \varepsilon, \beta) \text{ and } q_n \in F\}.$$

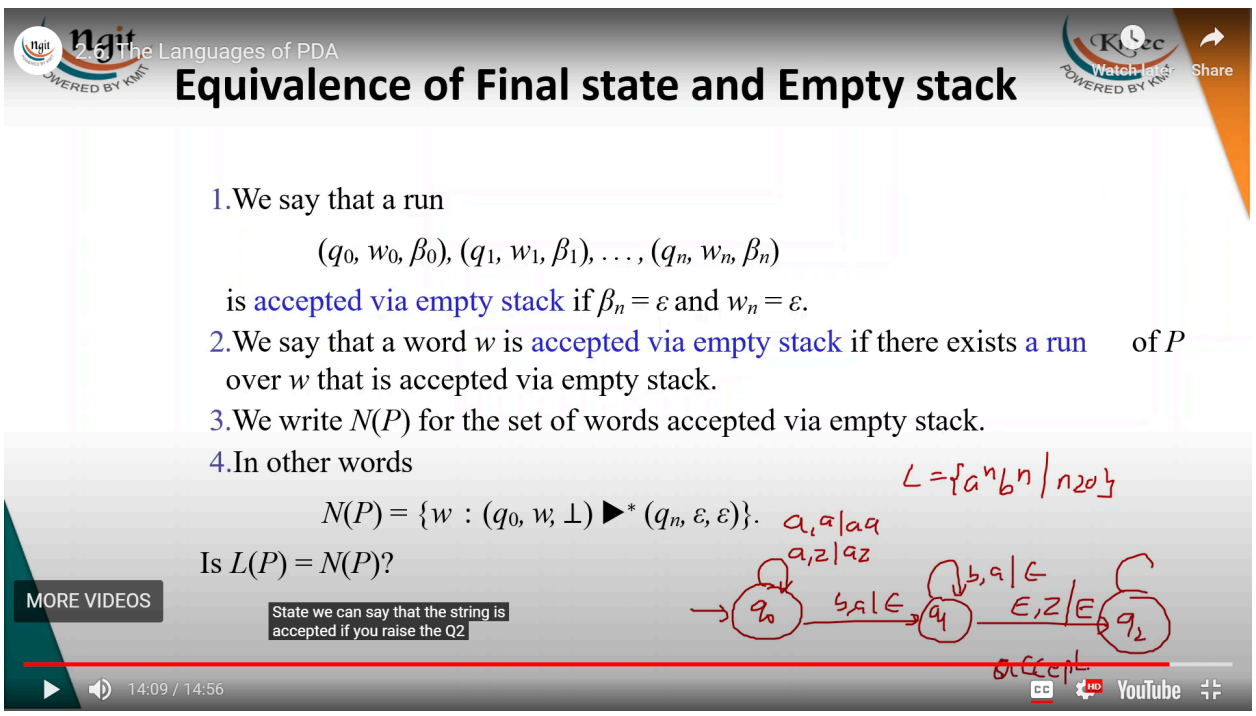
MORE VIDEOS

two ways one is acceptance by final state so

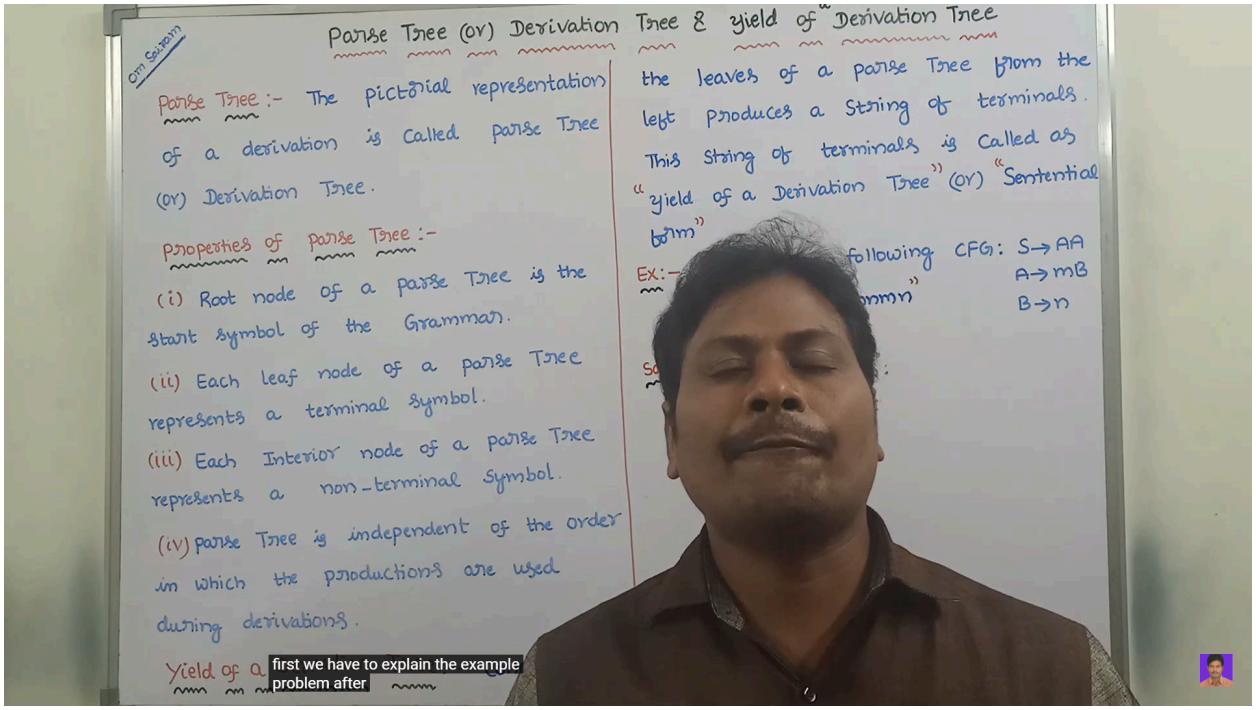
Example $L = \{ww : w \in \{0, 1\}^*\}$ with the notion of configuration, computation, run, and acceptance.

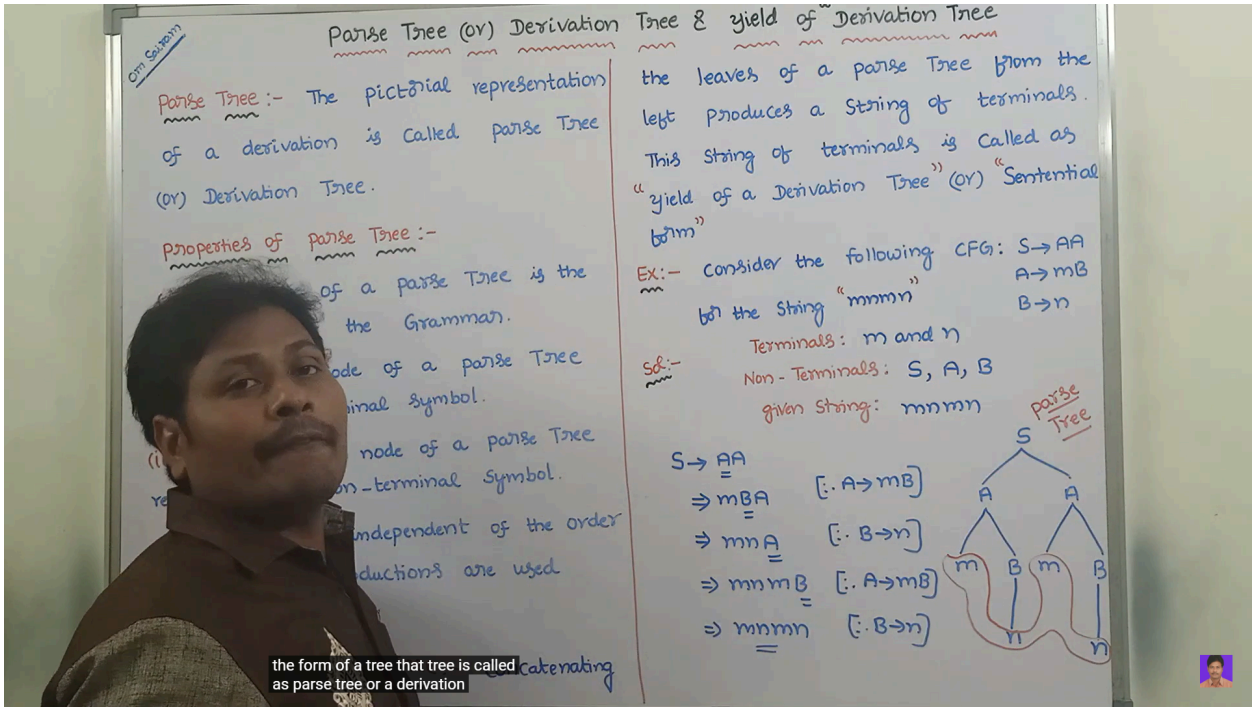
12:20 / 14:56





The Yield of Parse tree-


Om Saravana
Parse Tree (or) Derivation Tree & yield of Derivation Tree
Parse Tree :- The pictorial representation of a derivation is called parse tree (or) Derivation Tree.
Properties of Parse Tree :-
(i) Root node of a parse tree is the start symbol of the Grammar.
(ii) Each leaf node of a parse tree represents a terminal symbol.
(iii) Each interior node of a parse tree represents a non-terminal symbol.
(iv) parse tree is independent of the order in which the productions are used during derivations.
Yield of a
the leaves of a parse tree from the left produces a string of terminals. This string of terminals is called as "yield of a Derivation Tree" (or) "Sentential form"
Ex:-
following CFG: S→AA
A→mB
B→n
first we have to explain the example problem after


Om Saravana
Parse Tree (or) Derivation Tree & yield of Derivation Tree
Parse Tree :- The pictorial representation of a derivation is called parse tree (or) Derivation Tree.
Properties of Parse Tree :-
the leaves of a parse tree from the left produces a string of terminals. This string of terminals is called as "yield of a Derivation Tree" (or) "Sentential form"
Ex:- Consider the following CFG: S→AA
A→mB
B→n
for the string "mnmn"
Sol:-
Terminals: m and n
Non-Terminals: S, A, B
given string: mnmn
S→AA
⇒ mBA [∵ A→mB]
⇒ mnA [∵ B→n]
⇒ mnmB [∵ A→mB]
⇒ mnmn [∵ B→n]
parse Tree
the form of a tree that tree is called as parse tree or a derivation