Cairo University
FCI

Theory of Computation

Context Free Grammar (CFG)

Manar Elkady, Ph.D.

This lecture is prepared by Dr. Hussein Sharaf and Neso Academy

p.434: Regular Languages and non-regular languages

Language	Language Defined by	Corresponding Accepting Machine
Regular Languages (Type3 languages)	RE	Finite automaton, transition graph
Context-free Lang **(Type2 languages)**	**Context-free grammar**	**Deterministic Pushdown automaton**
Type 0 language (Recursively enumerable) And Type1 (context sensitive lang)	Type 0 grammar (Recursively enumerable) And Type1 (context sensitive grammar)	Turing machines

CFG

- A *context-free grammar* is a notation for defining context free languages.
- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.
- Basic idea is to use "variables" to stand for sets of strings.
- These variables are defined recursively, in terms of one another.

CFG formal definition

- C =(V, Σ, R, S)
- V: is a finite set of variables.
- Σ: symbols called terminals of the alphabet of the language being defined.
- S ∈ V: a special start symbol.
- R: is a finite set of production rules of the form $A \rightarrow \alpha$ where $A \in V$, $\alpha \in V \cup \Sigma$

CFG - 1

- Define the language $\{ a^n b^n \mid n \geq 1 \}$.
- Terminals/symbols = {a, b}.
- Variables = {S}.
- Start symbol = S.
- Productions ={

  $S \rightarrow ab$,

  $S \rightarrow aSb$ }

Summary

$$S \rightarrow ab$$

$$S \rightarrow aSb$$

Derivation

- Derivation example for "aabb"
- Using S→ aSb

  ➔generates uncompleted string that still has a non-terminal S.

- Then using S→ ab to replace the inner S

  ➔Generates "aabb"

➔S ➔aSb ➔aabb ……[Successful derivation of aabb]

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.
  - That is, the "productions for A" are those that have A on the left side of the .

Derivations – Formalism

- We say $\alpha A \beta \longrightarrow \alpha\gamma\beta$ if $A \longrightarrow \gamma$ is a production.
- Example: $S \longrightarrow 01$; $S \longrightarrow 0S1$.
- S => 0S1 => 00S11 => 000111.

$$\begin{tikzcd} S \arrow[rr, bend right] & => & 0S1 \arrow[rr, bend right] & => & 00S11 \arrow[rr, bend right] & => & 000111. \end{tikzcd}$$

Balanced-parentheses

Prod1 $S \rightarrow (S)$

Prod2 $S \rightarrow ()$

- Derive the string ((())).

$S \rightarrow (S)$	.....[by prod1]
$\rightarrow ((S))$	.....[by prod1]
$\rightarrow ((()))$	.....[by prod2]

Palindrome

- Describe palindrome of a's and b's using CGF
- 1] $S \rightarrow aSa$ 2] $S \rightarrow bSb$
- 3] $S \rightarrow b$ 4] $S \rightarrow a$
- 5] $S \rightarrow \Lambda$

- Derive "baab" from the above grammar.
- $S \rightarrow \underline{bSb}$ [by 2]

  $\rightarrow b\underline{aSa}b$ [by 1]

  $\rightarrow baab$ [by 5]

CFG – 2.1

- Describe anything (a+b)* using CGF

1] $S \rightarrow \Lambda$	2] $S \rightarrow Y$	3] $Y \rightarrow aY$
4] $Y \rightarrow bY$	5] $Y \rightarrow a$	6] $Y \rightarrow b$

- Derive "aab" from the above grammar.
- $S \rightarrow \underline{Y}$ [by 2]

  $S \rightarrow \underline{aY}$ [by 3]

  $Y \rightarrow a\underline{aY}$ [by 3]

  $Y \rightarrow aa\underline{b}$ [by 6]

CFG – 2.2

1] $S \rightarrow \Lambda$	2] $S \rightarrow Y$	3] $Y \rightarrow aY$
4] $Y \rightarrow bY$	5] $Y \rightarrow a$	6] $Y \rightarrow b$

- Derive “aa” from the above grammar.
- $S \rightarrow \underline{Y}$ [by 2]

  $Y \rightarrow \underline{aY}$ [by 3]

  $Y \rightarrow a\underline{a}$ [by 5]

Remember CFG is about categorizing the grammar of a language

CFG – 3

- Describe anything (a+b)* using CGF

1] $S \rightarrow aS$ 2] $S \rightarrow bS$ 3] $S \rightarrow \Lambda$

- Derive “aab” from the above grammar.
- $S \rightarrow \underline{aS}$ [by 1]

  $S \rightarrow a\underline{aS}$ [by 1]

  $S \rightarrow aa\underline{bS}$ [by 2]

  $S \rightarrow aab\underline{\Lambda}$ [by 3] $\rightarrow aab$

CFG – 4 Anything aa Anything

Describe anything (a+b)*aa(a+b)* using CGF

1] $S \rightarrow XaaX$ 2] $X \rightarrow aX$ 3] $X \rightarrow bX$

4] $X \rightarrow \Lambda$

Note: rules 2, 3, 4 represents anything (a+b)*

- Derive “baabaab” from the above grammar.
- $S \rightarrow \underline{XaaX}$ [by 1]

$X \rightarrow b\underline{\Lambda}aaX$ [by 4]

$X \rightarrow baab\underline{aX}$ [by 2]

$X \rightarrow baabaa\underline{bX}$ [by 3]

$X \rightarrow \underline{bX}aaX$ [by 3]

$X \rightarrow baa\underline{bX}$ [by 3]

$X \rightarrow baaba\underline{aX}$ [by 2]

$X \rightarrow baabaab\underline{\Lambda}$ [by 4]

CFG – 5 Even-Even grammar

Describe a language that has even number of a’s and even number of b’s using CGF.

i.e. aababbab, aaabaababb

1] $S \rightarrow S\ S$

2] $S \rightarrow aa$

3] $S \rightarrow bb$

4] $S \rightarrow \Lambda$

5] $S \rightarrow \underline{\text{UNBALANCED}}\ \boxed{S}\ \underline{\text{UNBALANCED}}$

6] $\text{UNBALANCED} \rightarrow ab$

7] $\text{UNBALANCED} \rightarrow ba$

CFG – 5 Even-Even grammar

- Derive "aababbbab" from the above grammar.

- S →SS

→ aa UNBALANCED S UNBALANCED

→ aa UNBALANCED $\underline{\text{S}}$ UNBALANCED

→ aa ba $\underline{\text{S}}$ UNBALANCED

→ aa ba $\underline{\text{bb}}$ UNBALANCED

→ aa ba $\underline{\text{bb}}$ ab

CFG – 6 Balanced a-b grammar

- i.e. {Λ, ab, aaabbb,… }
- $S \rightarrow aSb \mid \Lambda$

- Derive "aaaabbbb"
- $S \rightarrow aSb \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb$

  $\rightarrow \underline{aaaaSbbbb} \rightarrow \underline{aaaabbbb}$

CFG – 7 Even-Plaindrome grammar

- i.e. {Λ, ab, abbaabba,… }
- S → aSa | bSb | Λ

- Derive "abbaabba" "abba abba"
- S →aSa→abSba→abbSbba →abbaSabba→ abba abba

→ abbaabba

CFG – 8 Plaindrome grammar

- i.e. {Λ, abb, ababa,… }
- S → aSa | bSb | a | b | Λ

- Derive "ababa" "ab a ba"
- S →aSa→abSba→abSba →abSba→ ababa

- Note: $a^n b a^n$ can be represented by S → aSa | b
- But $a^n b a^n b^{n+1}$ can not be represented by CFG !

$S \to aSab \mid b$

$X \to b$

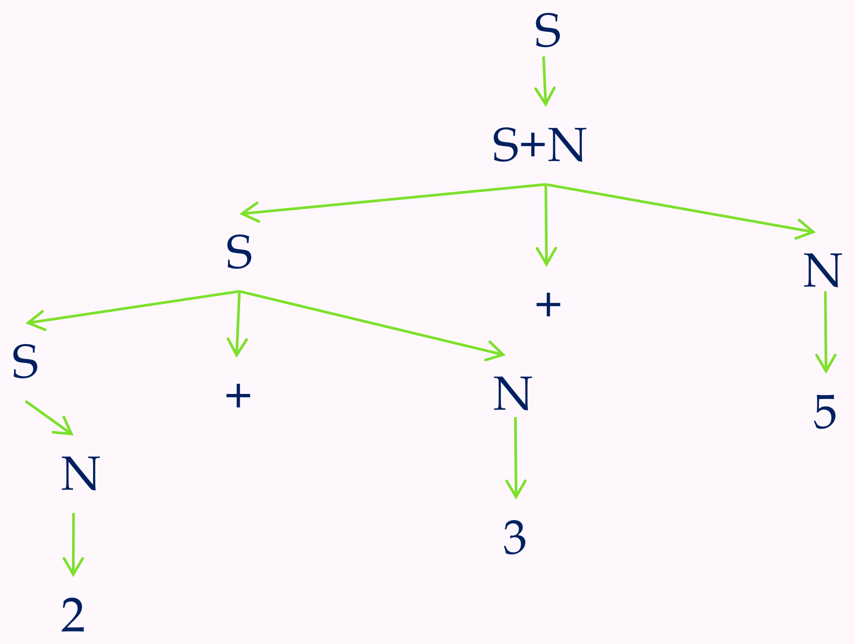
CFG – 9 Even-Plaindrome grammar

- i.e. {Λ, ab, abbaabba,… }
- S → aSa | bSb | Λ   Derive abaaba

$$\begin{tikzcd}
& S \arrow[dl] \arrow[d] \arrow[dr] & \\
a & S \arrow[dl] \arrow[d] \arrow[dr] & a \\
b & S \arrow[dl] \arrow[d] \arrow[dr] & b \\
a & S \arrow[d] & a \\
& \Lambda &
\end{tikzcd}$$

CFG – 10

- Deduce CFG of addition and parse the following expression 2+3+5
- 1] S→S+S | N
- 2] N→1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | NN



Can u make another parsing tree ?

CFG – 11.1

- Deduce CFG of a addition/multiplication and parse the following expression 2+3*5
- 1] S→S+S|S*S|N
- 2] N→1|2|3|4|5|6|7|8|9|0

N1|N2|N3|N4|N5|N6|N7|N8|N9|N0

$$\begin{tikzcd}
 & & & & S \arrow[d] & \\
 & & & & S*S \arrow[dll] \arrow[d] \arrow[dr] & \\
 & & S \arrow[dll] \arrow[d] \arrow[dr] & & * & N \arrow[d] \\
S \arrow[dr] & & + & N \arrow[d] & & 5 \\
 & N \arrow[d] & & 3 & & \\
 & 2 & & & &
\end{tikzcd}$$

Can u make another parsing tree ?

S ->S + S
S → N

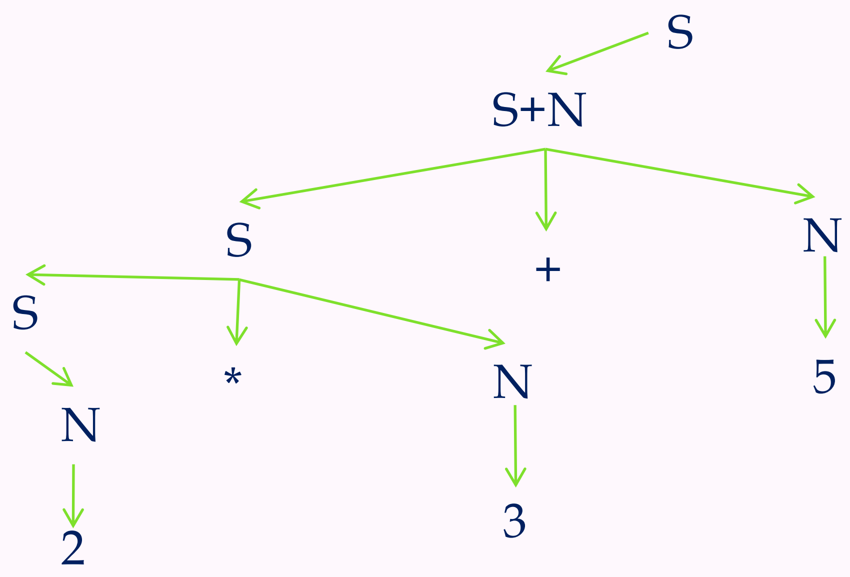
CFG -11.2 without ambiguity

- Deduce CFG of a addition/multiplication and parse the following expression 2*3+5

1] S→ Term | Term + S

2] Term → N | N * Term

- 3] N→1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
  N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N0



Can u make another parsing tree ?

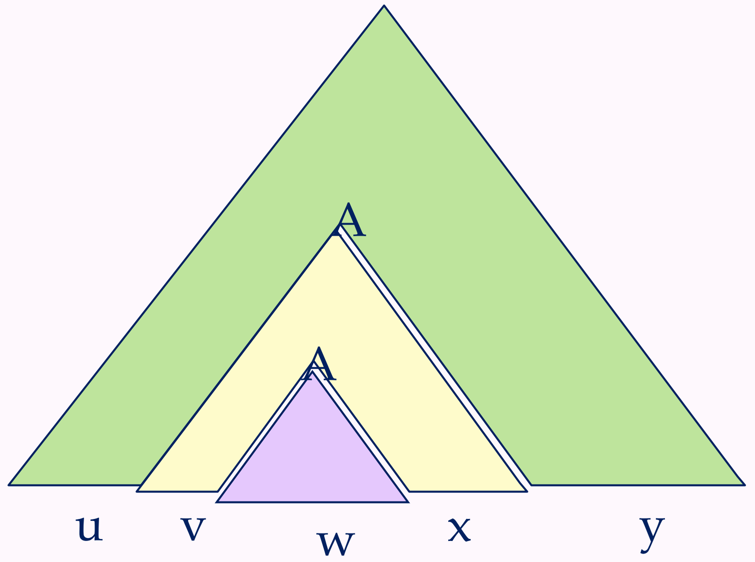
CFL Pumping Lemma

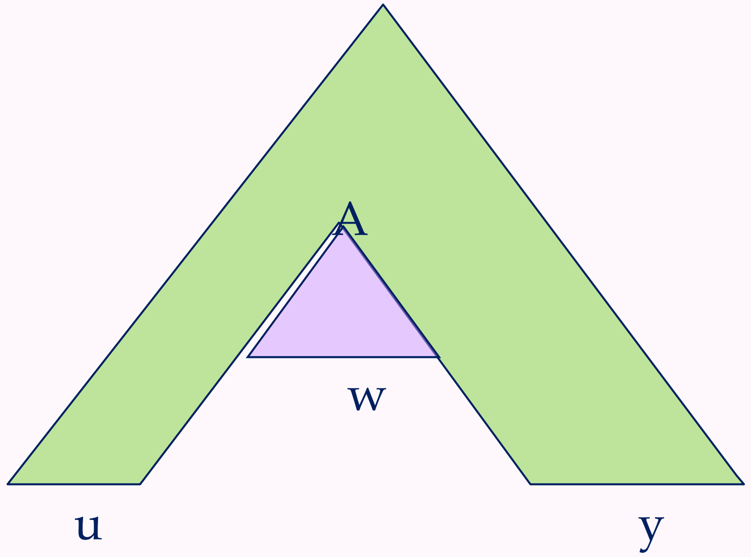
- Recall the pumping lemma for regular languages.
- It told us that if there was a string long enough to cause a cycle in the DFA for the language, then we could "pump" the cycle and discover an infinite sequence of strings that had to be in the language.

CFL Pumping Lemma (2)

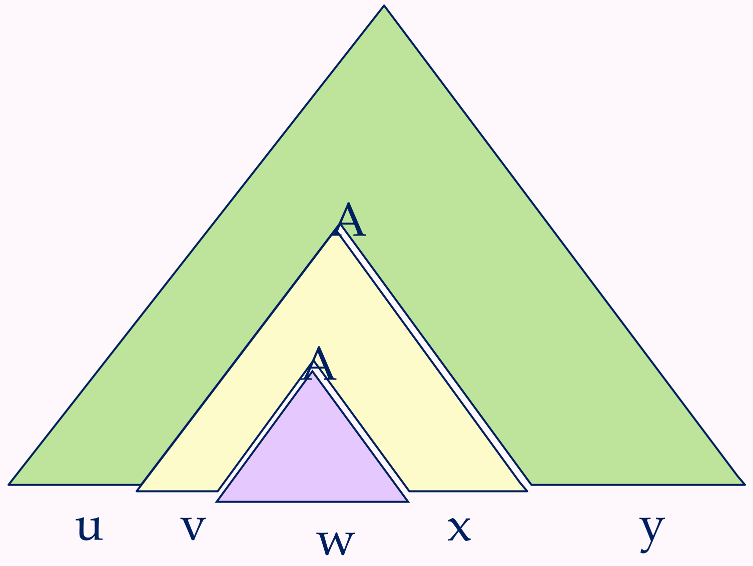
- For CFL's the situation is a little more complicated.
- We can always find two pieces of any sufficiently long string to "pump" in tandem.
    - That is: if we repeat each of the two pieces the same number of times, we get another string in the language.

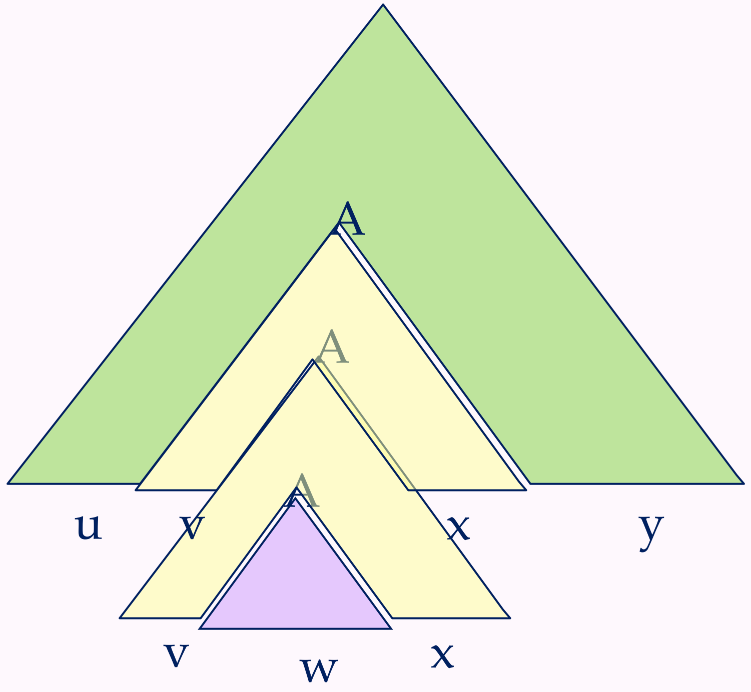
Pump Zero Times



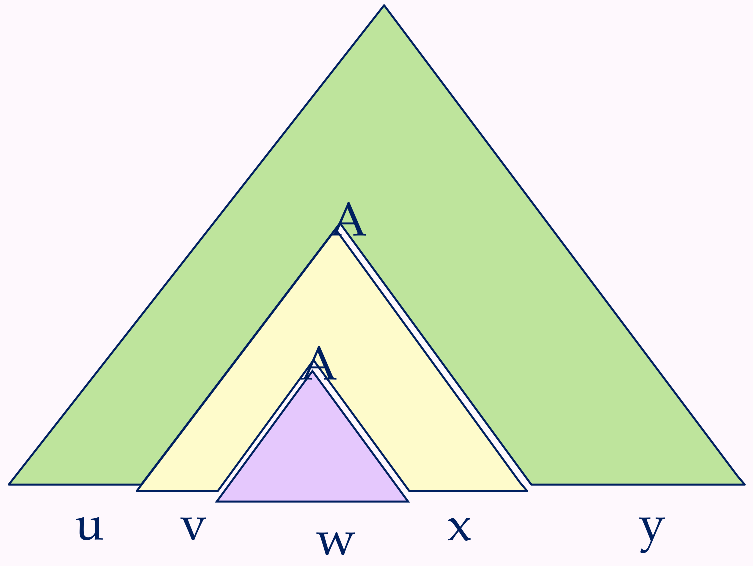


Pump Twice

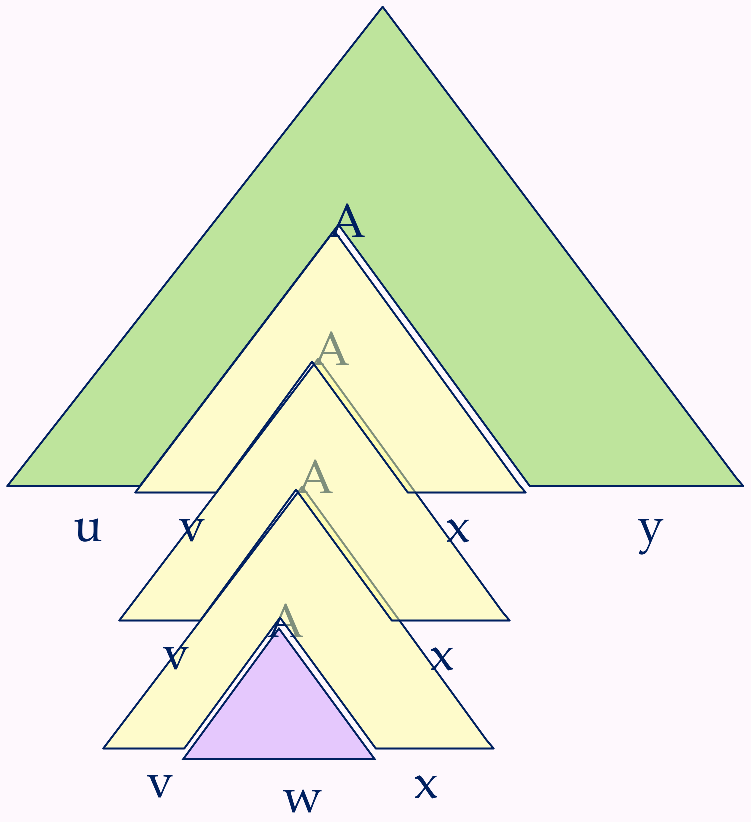


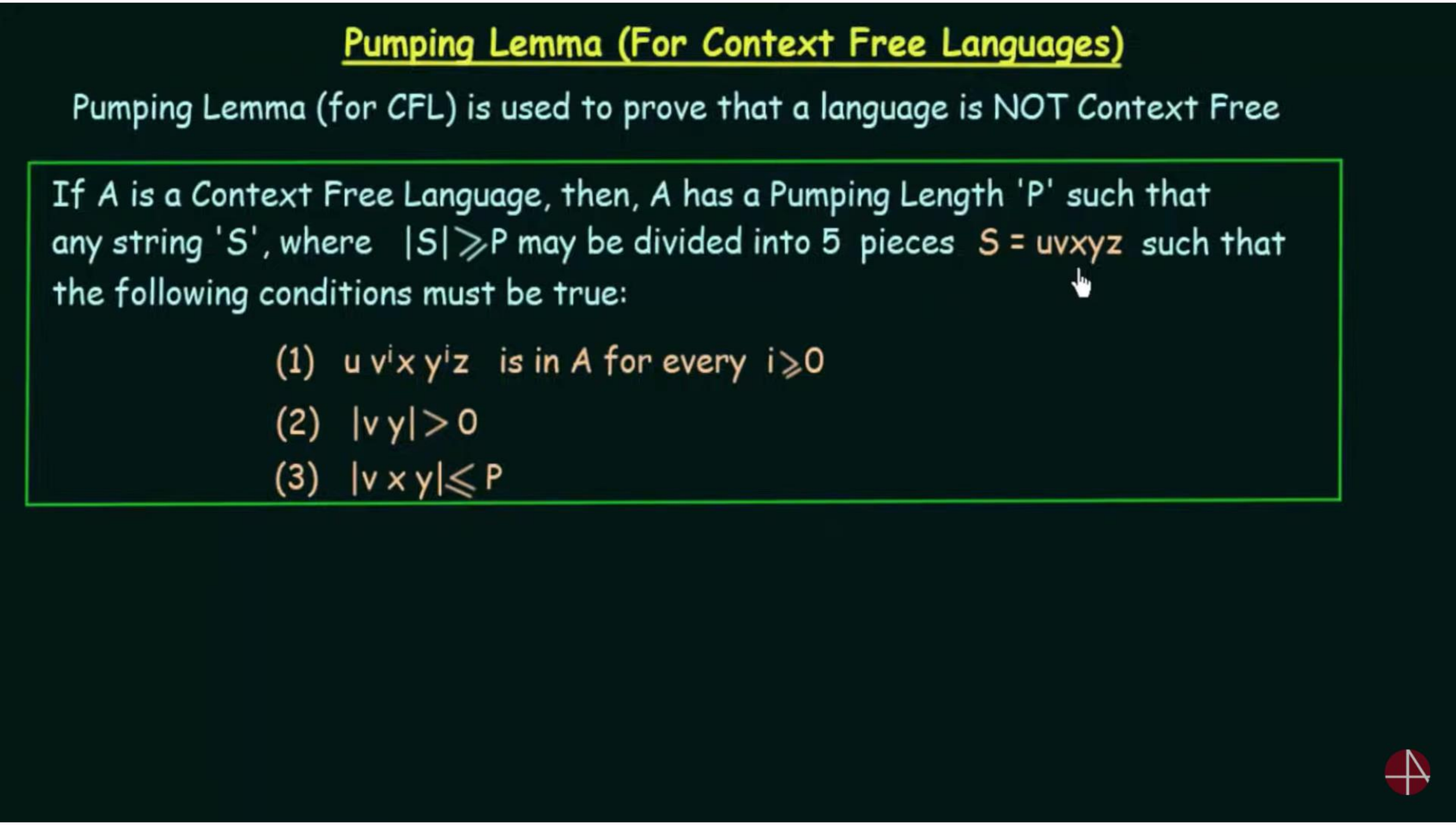


Pump Thrice



Etc., Etc.





Pumping Lemma (For Context Free Languages)

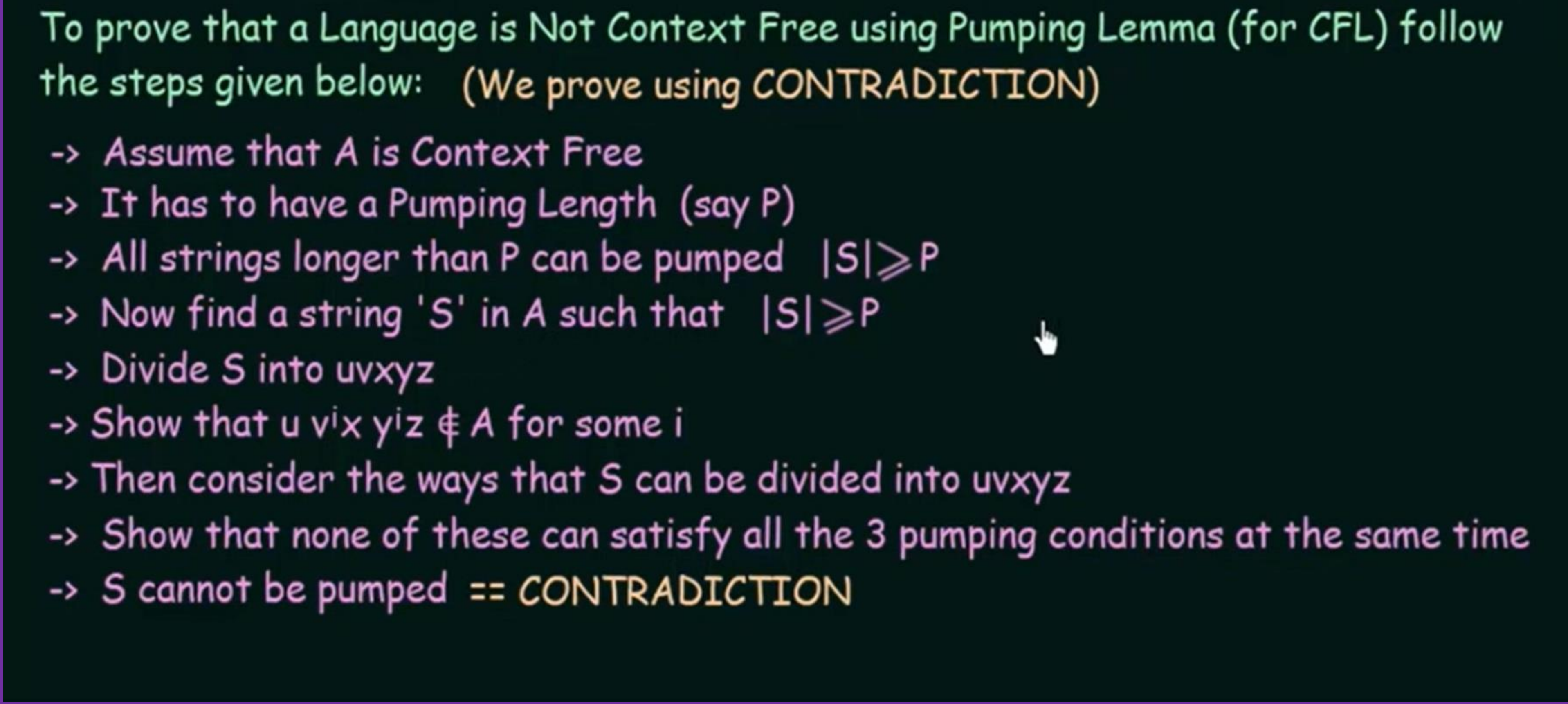
Pumping Lemma (for CFL) is used to prove that a language is NOT Context Free

If A is a Context Free Language, then, A has a Pumping Length 'P' such that any string 'S', where $|S| \geqslant P$ may be divided into 5 pieces $S = uvxyz$ such that the following conditions must be true:

(1) $u v^i x y^i z$ is in A for every $i \geqslant 0$

(2) $|v y| > 0$

(3) $|v x y| \leqslant P$



To prove that a Language is Not Context Free using Pumping Lemma (for CFL) follow the steps given below: (We prove using CONTRADICTION)

-> Assume that A is Context Free

-> It has to have a Pumping Length (say P)

-> All strings longer than P can be pumped $|S| \geqslant P$

-> Now find a string 'S' in A such that $|S| \geqslant P$

-> Divide S into uvxyz

-> Show that $u v^i x y^i z \notin A$ 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

Pumping Lemma (for Context Free Languages) - Example (Part-1)

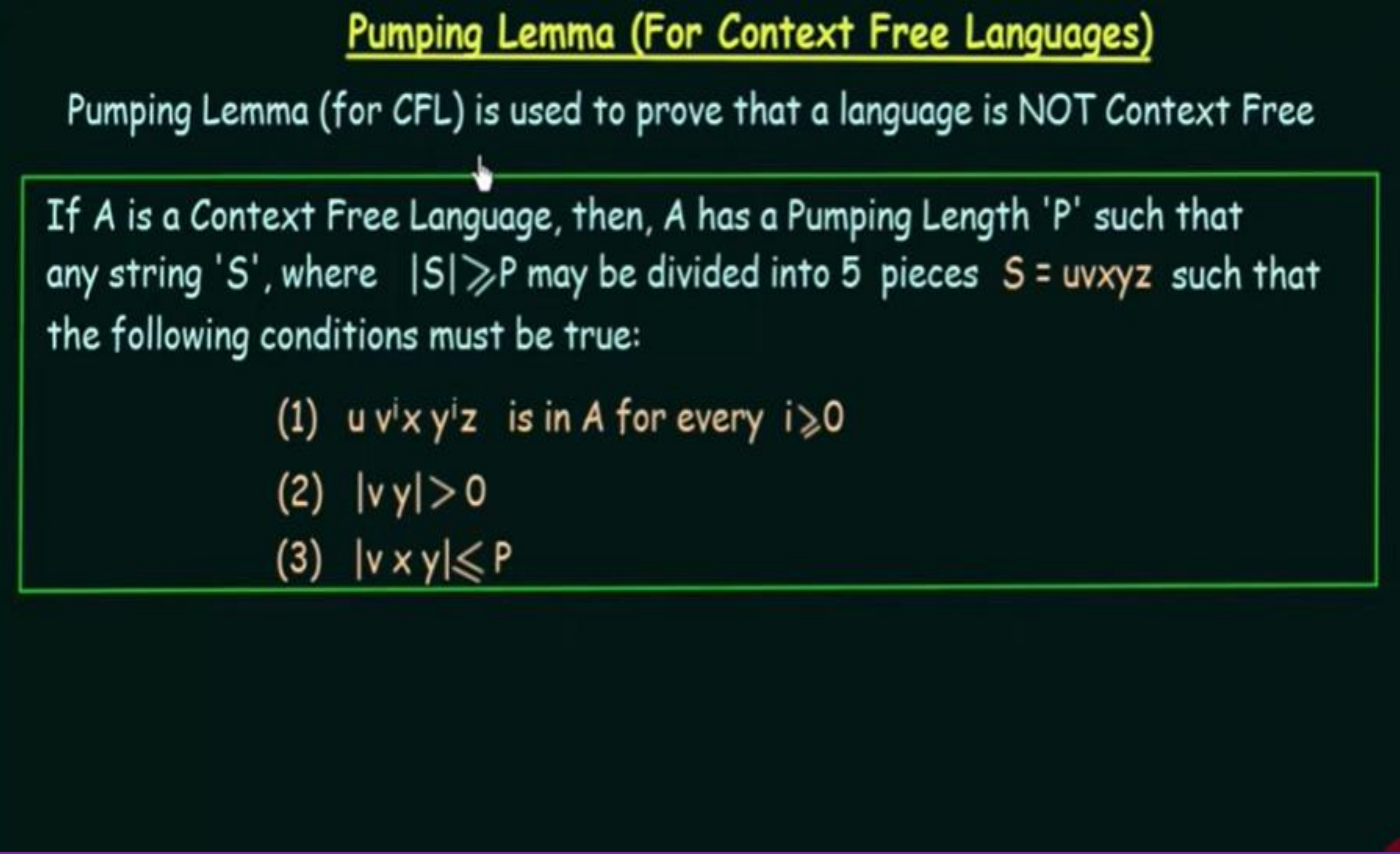
Show that $L = \{ a^N b^N c^N \mid N \geq 0 \}$ is Not Context Free

-> Assume that L is Context Free

-> L must have a pumping length (say P)

-> Now we take a string S such that $S = a^P b^P c^P$

-> We divide S into parts u v x y z



Pumping Lemma (For Context Free Languages)

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 $|S| \geqslant P$ may be divided into 5 pieces $S = uvxyz$ such that the following conditions must be true:

(1) $u v^i x y^i z$ is in A for every $i \geqslant 0$

(2) $|v y| > 0$

(3) $|v x y| \leqslant P$

Pumping Lemma (for Context Free Languages) - Example (Part-1)

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

-> Assume that L is Context Free

-> L must have a pumping length (say P)

-> Now we take a string S such that $S = a^P b^P c^P$

-> We divide S into parts u v x y z

Eg. $P = 4$ So, $S = a^4 b^4 c^4$

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

$$\underbrace{a}_{u}\,\underbrace{a\,a}_{v}\,\underbrace{a\,b\,b\,b\,b\,c}_{x}\,\underbrace{c}_{y}\,\underbrace{c\,c}_{z}$$

$uv^i x y^i z$ $(i = 2)$

$uv^2 x y^2 z$

a a a a a a b b b b c c c c c

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

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

$\underbrace{a}_{u}\underbrace{aa}_{v}\underbrace{abbbbc}_{x}\underbrace{c}_{y}\underbrace{cc}_{z}$

$uv^ixy^iz \quad (i=2)$

uv^2xy^2z

aaaaa abbbbccccc

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

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

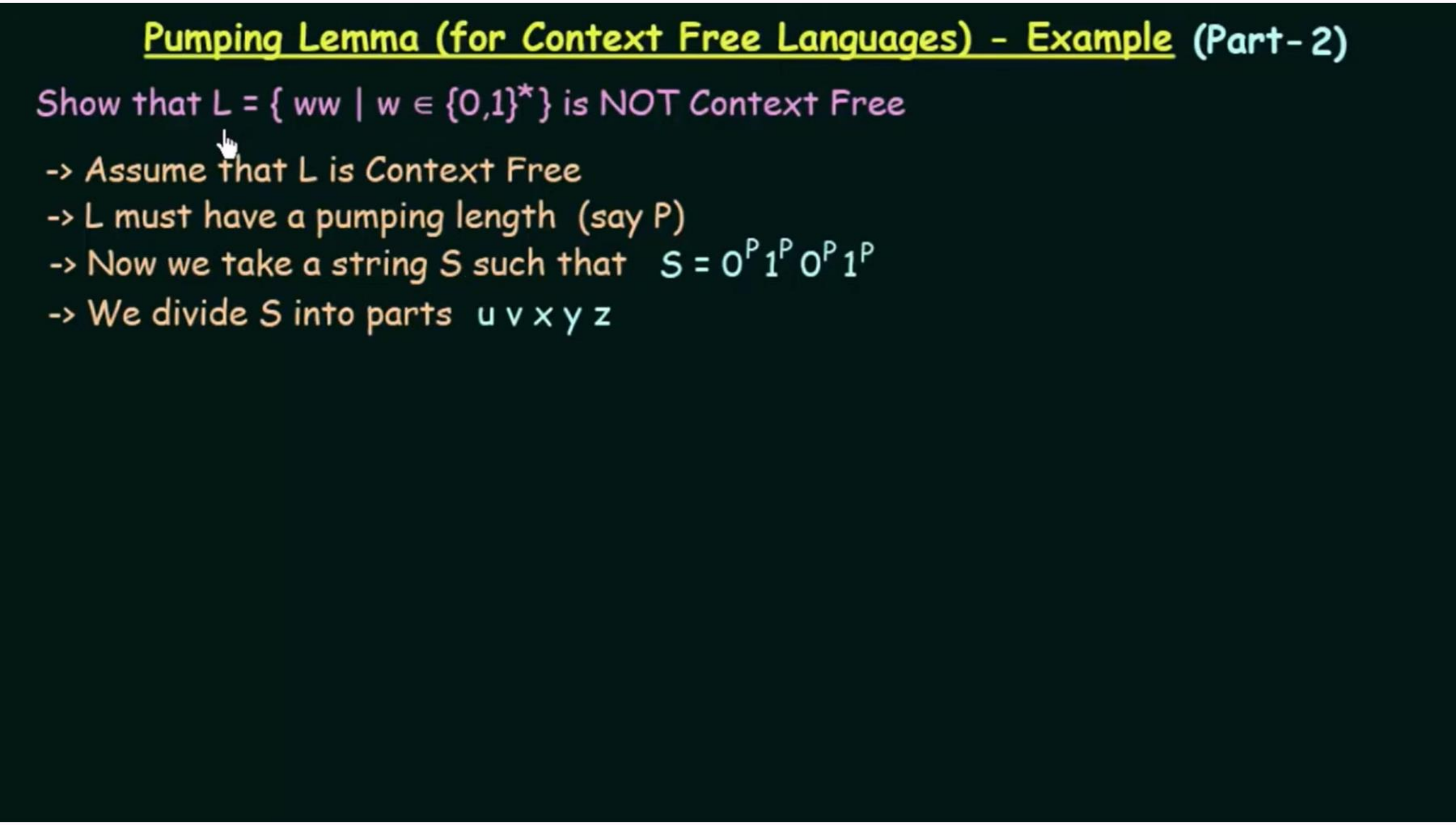
$\underbrace{aa}_{u}\underbrace{aabb}_{v}\underbrace{b}_{x}\underbrace{b}_{y}\underbrace{cccc}_{z}$

$uv^ixy^iz \quad (i=2)$

$a^Nb^Nc^N$

uv^2xy^2z

aa aabbaabbbbbcccc $\notin L$



Pumping Lemma (for Context Free Languages) - Example (Part-2)

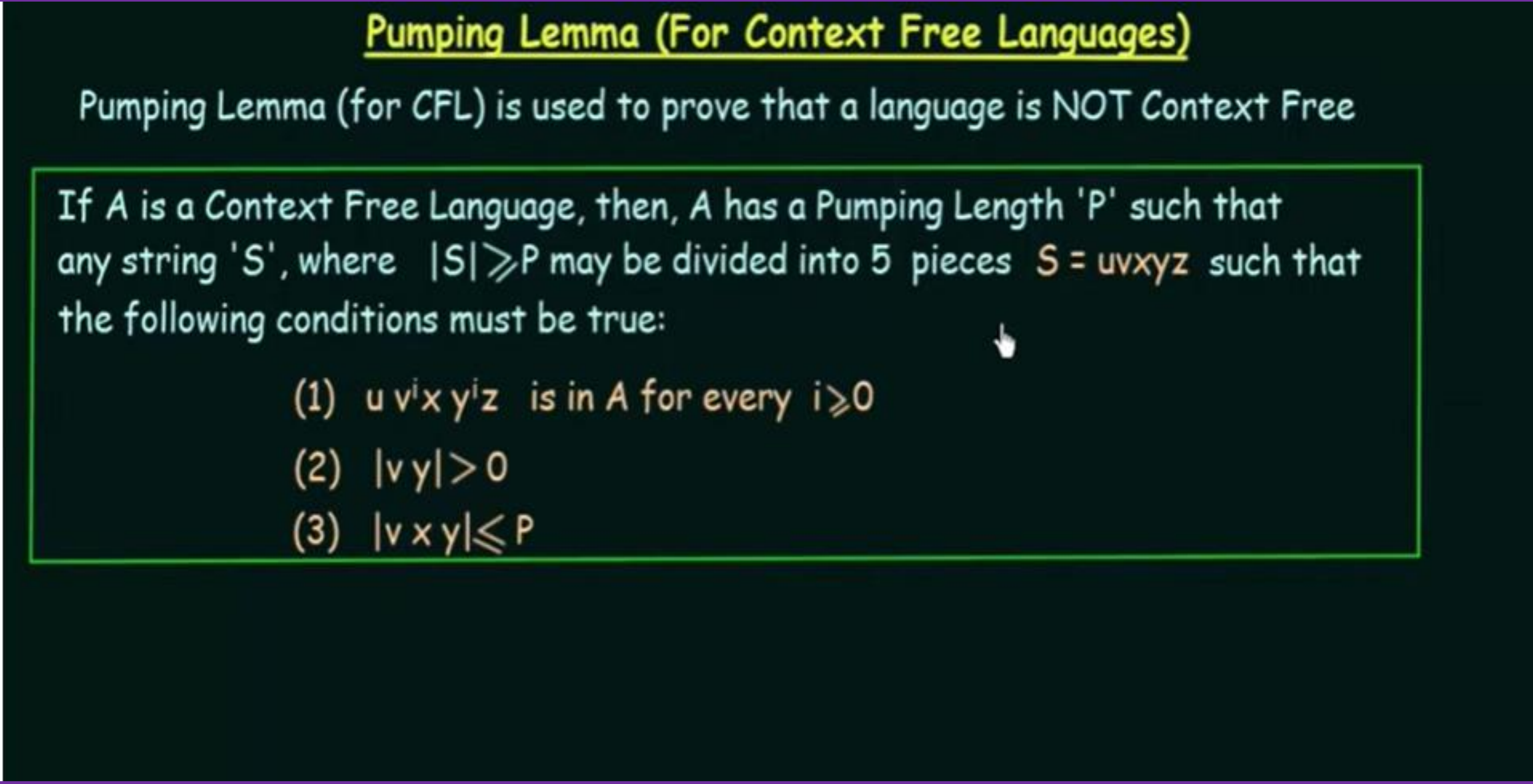
Show that $L = \{ ww \mid w \in \{0,1\}^* \}$ is NOT Context Free

-> Assume that L is Context Free

-> L must have a pumping length (say P)

-> Now we take a string S such that $S = 0^P 1^P 0^P 1^P$

-> We divide S into parts u v x y z



Pumping Lemma (For Context Free Languages)

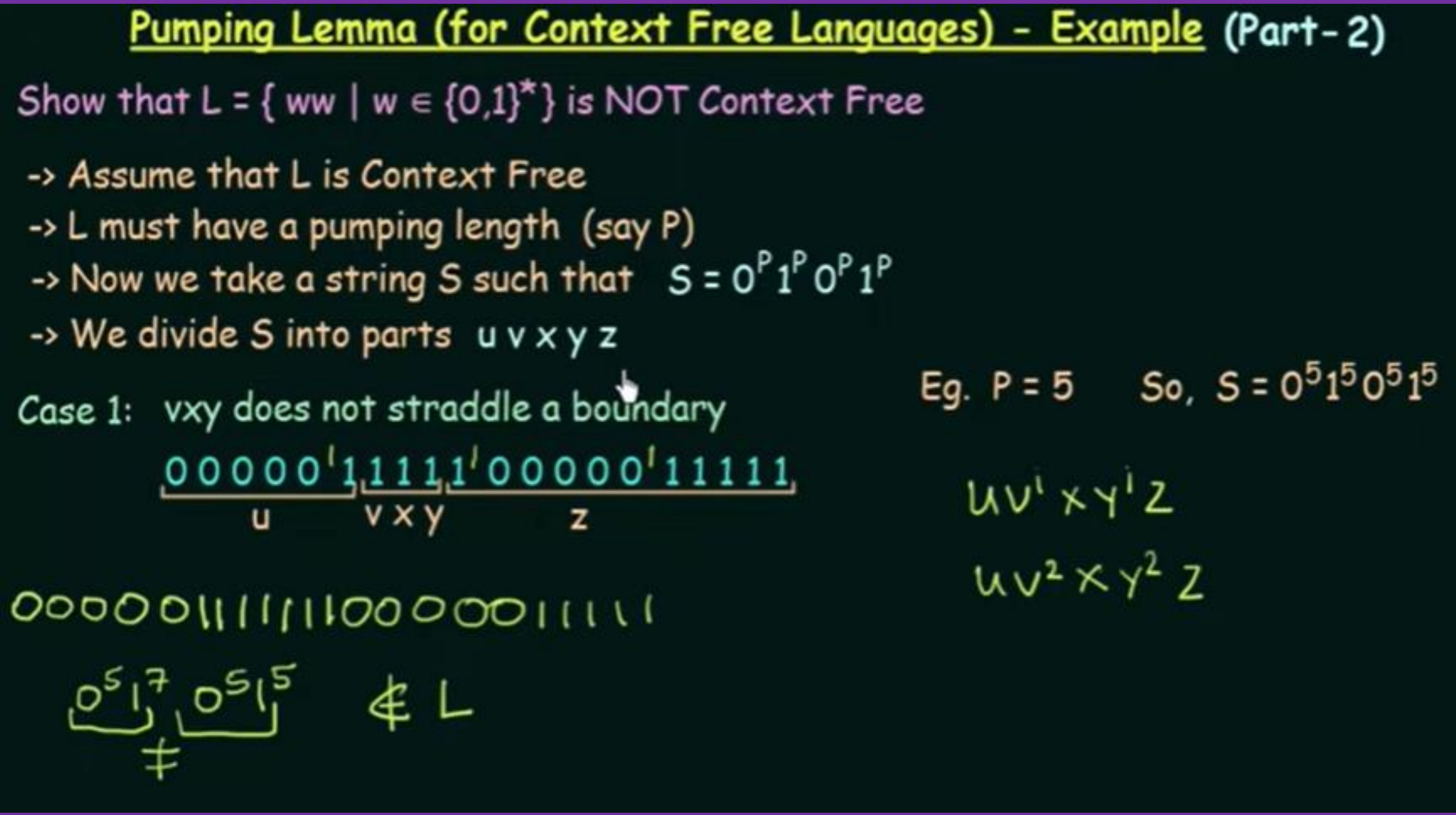
Pumping Lemma (for CFL) is used to prove that a language is NOT Context Free

If A is a Context Free Language, then, A has a Pumping Length 'P' such that any string 'S', where $|S| \geqslant P$ may be divided into 5 pieces $S = uvxyz$ such that the following conditions must be true:

(1) $u v^i x y^i z$ is in A for every $i \geqslant 0$

(2) $|v y| > 0$

(3) $|v x y| \leqslant P$



Pumping Lemma (for Context Free Languages) - Example (Part-2)

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 (say P)

-> Now we take a string S such that $S = 0^P 1^P 0^P 1^P$

-> We divide S into parts u v x y z

Eg. $P = 5$ So, $S = 0^5 1^5 0^5 1^5$

Case 1: vxy does not straddle a boundary

$$\underbrace{00000\,1}_{u}\underbrace{111}_{v\,x\,y}\underbrace{1\,00000\,11111}_{z}$$

uv^1xy^1z

uv^2xy^2z

000001111111000001111 1

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

Case 2a: vxy straddles the first boundary

$$\underbrace{0\,0\,0}_{u}\,\underbrace{0\,0\,1\,1\,1}_{v\quad x\quad y}\,\underbrace{1\,1\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1}_{z}$$

uv^1xy^1z

uv^2xy^2z

000 0000 1111111 00000 11111

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

Case 2b: vxy straddles the third boundary

$$\underbrace{00000'11111'000}_{u}\underbrace{00'}_{v}\underbrace{1}_{x}\underbrace{11}_{y}\underbrace{11}_{z}$$

$u v^2 x y^2 z$

00000 11111 000 00001 111111

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

Case 3: vxy straddles the midpoint

$$\underbrace{00000'11}_{u}\,\underbrace{111'00}_{v\ \ x\ \ y}\,\underbrace{000'11111}_{z}$$

uv^2xy^2z

$0000011111110000000\,11111$

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

L is not context Free