THEORY OF COMPUTATION

CSC-251

UNIT 2

DWIT

Sailesh.bajracharya@gmail.com

9841594548

Outline

- Context –Free Grammar(CFG),Parse Trees, Derivation and Ambiguity, Normal Forms (CNF and GNF) of context -Free Grammar, Regular Grammars, Closure Properties of context-Free Languages, Proving a Language to be Non –Context – Free.
- Push Down Automata (PDA), Languages of PDA, Deterministic and Non- deterministic PDA, Equivalences of PDA’s and CFG’s.

Grammars

- Grammars express languages
- Example: the English language grammar

$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle$$

$$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \ \langle noun \rangle$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

Grammar

$$\langle article \rangle \rightarrow a$$

$$\langle article \rangle \rightarrow the$$

$$\langle noun \rangle \rightarrow cat$$

$$\langle noun \rangle \rightarrow dog$$

$$\langle verb \rangle \rightarrow runs$$

$$\langle verb \rangle \rightarrow sleeps$$

Derivation of string “the dog sleeps”:

$$
\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun\_phrase \rangle \; \langle predicate \rangle \\
&\Rightarrow \langle noun\_phrase \rangle \; \langle verb \rangle \\
&\Rightarrow \langle article \rangle \; \langle noun \rangle \; \langle verb \rangle \\
&\Rightarrow the \; \langle noun \rangle \; \langle verb \rangle \\
&\Rightarrow the \;\; dog \; \langle verb \rangle \\
&\Rightarrow the \;\; dog \;\; sleeps
\end{aligned}
$$

Context Free Grammar(CFG)

- more powerful method for describing languages
- Consists of a finite set of grammar rules
- Grammar rules :
  - non-terminals (variables)
  - terminals
- A rule is of the form $A \rightarrow \alpha$, where A is a single nonterminal, and the right-hand side α is a string of terminal and/or nonterminal symbols
- Unlike automata, grammars are used to generate strings, rather than recognize strings.

What `context free' means

All the use of the term context-free really means is that the non-terminal on the left-hand side of the rule is sitting over there all by itself.

$A \rightarrow B\ C$

In other words, I can rewrite A as BC, regardless of the context in which I find the A.

Context Free Grammar: Quadruple

$$G=(V, T, P, S)$$

Set of variables (non-terminals/syntactic categories)

Set of Terminals

Set of Productions(rules)

Recursive definition of a language

Start Symbol

Each Production rules consists of:

- Head
  - variable defined by the production
- Production symbol $\rightarrow$
- Body
  - string of zero or more terminals & variables

$$A \longrightarrow Aab$$

CFG for Palindromes

$$G_{pal} = (\{P\}, \{0,1\}, A, P)$$

where A represents the production rules:

$$\begin{aligned}
P &\to \epsilon \\
P &\to 0 \\
P &\to 1 \\
P &\to 0P0 \\
P &\to 1P1
\end{aligned}$$

We can also write: $P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$

Examples: CFG for expressions in a typical programming language

- Operators: +(addition) and $\ast$(multiplication)
- Identifiers: must begin with a or b, which may be followed by any string in {a,b,0,1}*
- We need two variables in this grammar:
  - E: represents **expressions** .It is the start symbol.
  - I: represents the **identifiers**.
    - Its language is regular and is the language of the regular expression:

$$(a+b)(a+b+0+1)^*$$

Example: The Grammar

Grammar $G_1 = (\{E, I\}, T, P, E)$ where: $T = \{+, *, (,), a, b, 0, 1\}$ and P is the set of productions:

$$
\begin{array}{r|ccc}
1 & E & \rightarrow & I \\
2 & E & \rightarrow & E + E \\
3 & E & \rightarrow & E * E \\
4 & E & \rightarrow & (E) \\
5 & I & \rightarrow & a \\
6 & I & \rightarrow & b \\
7 & I & \rightarrow & Ia \\
8 & I & \rightarrow & Ib \\
9 & I & \rightarrow & I0 \\
10 & I & \rightarrow & I1
\end{array}
$$

Compact Notation for Productions

- We often refers to the production whose head is A as "productions for A" or "A-productions"
- Moreover, the productions

$$A \rightarrow \alpha_1, A \rightarrow \alpha_2 \ldots A \rightarrow \alpha_n$$

can be replaced by the notation

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

Derivations Using a Grammar

- We apply the productions of a CFG to infer that certain strings are in the language of a certain variable
- Two inference approaches:
  - Recursive inference(using productions from body to head)
  - Derivations(using productions from head to body)

Recursive Inference - Example

We consider some inferences we can make using G_1

Recall G_1:

$$
\begin{array}{lcl}
E & \rightarrow & I \mid E + E \mid E * E \mid (E) \\
I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{array}
$$

	String Inferred	For language of	Production used	String(s) used
(i)	a	I	5	—
(ii)	b	I	6	—
(iii)	$b0$	I	9	(ii)
(iv)	$b00$	I	9	(iii)
(v)	a	E	1	(i)
(vi)	$b00$	E	1	(iv)
(vii)	$a + b00$	E	2	$(v), (vi)$
$(viii)$	$(a + b00)$	E	4	(vii)
(ix)	$a * (a + b00)$	E	3	$(v), (viii)$

Sample CFG

1. E→I // Expression is an identifier
2. E→E+E //Add two expressions
3. E→E*E //Multiply two expressions
4. E→(E) //Add parenthesis
5. I→ L // Identifier is a Letter
6. I→ ID //Identifier + Digit
7. I→ IL //Identifier + Letter
8. D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 // Digits
9. L → a | b | c | … A | B | … Z // Letters

Note Identifiers are regular; could describe as (letter)(letter + digit)*

Recursive Inference - Example

- Process of coming up with strings that satisfy individual productions and then concatenating them together according to more general rules is called *recursive inference*.
- Bottom-up approach
- For example, parsing the identifier "r5"
  - Rule 8 tells us that D $\rightarrow$ 5
  - Rule 9 tells us that L $\rightarrow$ r
  - Rule 5 tells us that I$\rightarrow$L so I$\rightarrow$r
  - Apply recursive inference using rule 6 for I$\rightarrow$ID and get
    - I $\rightarrow$ rD.
    - Use D$\rightarrow$5 to get I$\rightarrow$r5.
  - Finally, we know from rule 1 that E$\rightarrow$I, so r5 is also an expression.

Recursive Inference - Exercise

- Show the recursive inference for arriving at *(x+y1)*y* is an expression
  1. E→I
  2. E→E+E
  3. E→E*E
  4. E→(E)
  5. I→ L
  6. I→ ID
  7. I→ IL
  8. D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  9. L → a | b | c | … A | B | … Z

Derivations– top to down approach

- Applying productions from head to body requires the definition of a new relational symbol: $\Rightarrow$
- Let:
  - G=(V,T,P,S) be a CFG
  - $A \in V$
  - $\alpha,\beta \subset (V \cup T)^*$ and
  - $A \rightarrow \gamma \in P$

Then we write

$$\alpha A \beta \Rightarrow_G \alpha\gamma\beta$$

or, if G is understood

$$\alpha A \beta \Rightarrow \alpha\gamma\beta$$

And say that $\alpha A \beta$ derives $\alpha\gamma\beta$.

Derivation: Definition

1 We say that string u *yields* string v, denoted $u \Rightarrow v$, if u turns to v after one application of a derivation rule.
example: $0\ A\ 1 \Rightarrow 0\ 0\ A\ 1\ 1$

2 If u turns to v after many rule applications then we say that $u \Rightarrow^* v$.
example: $0\ A\ 1 \Rightarrow^* 0\ 0\ 0\ 0\ 0\ 0\ A\ 1\ 1\ 1\ 1\ 1\ 1$

3 The sequence $u \Rightarrow v_1 \Rightarrow v_2 \Rightarrow \ldots \Rightarrow v_k \Rightarrow v$ is called a derivation of v from u.

Examples of derivation

Derivation of $a * (a + b000)$ by G_1

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow a * (E) \Rightarrow$$
$$a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow a * (a + I) \Rightarrow$$
$$a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

<u>Note 1</u>: At each step we might have several rules to choose from, e.g.

$I * E \Rightarrow a * E \Rightarrow a * (E)$, versus

$I * E \Rightarrow I * (E) \Rightarrow a * (E)$.

<u>Note 2</u>: Not all choices lead to successful derivations of a particular string, for instance

$E \Rightarrow E + E$ (at the first step)

won't lead to a derivation of $a * (a + b000)$.

<u>Important:</u> Recursive inference and derivation are equivalent. A string of terminals w is infered to be in the language of some variable A iff $A \stackrel{*}{\Rightarrow} w$

Leftmost and Rightmost derivation

- In other 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 (or rightmost) variable by one of its production rules
- Leftmost derivation $\Rightarrow_{lm}$: Always replace the left-most variable by one of its rule-bodies
- Rightmost derivation $\Rightarrow_{rm}$: Always replace the rightmost variable by one of its rule-bodies.

EXAMPLES

$1-$ Leftmost derivation: previous example

$2-$ Rightmost derivation:

$$E \Rightarrow_{rm} E * E \Rightarrow_{rm} E * (E) \Rightarrow_{rm}$$
$$E * (E + E) \Rightarrow_{rm} E * (E + I) \Rightarrow_{rm} E * (E + I0) \Rightarrow_{rm}$$
$$E * (E + I00) \Rightarrow_{rm} E * (E + b00) \Rightarrow_{rm} E * (I + b00) \Rightarrow_{rm}$$
$$E * (a + b00) \Rightarrow_{rm} I * (a + b00) \Rightarrow_{rm} a * (a + b00)$$

We can conclude that $E \Rightarrow_{rm}^{*} a * (a + b00)$

Left or Right?

- Does it matter which method you use?
- Answer: No
- Any derivation has an equivalent leftmost and rightmost derivation. That is, $A \Rightarrow^* \alpha$. iff $A \Rightarrow^*_{lm} \alpha$ and $A \Rightarrow^*_{rm} \alpha$.

Leftmost and rightmost Derivation example

Generate: aaabba

$$\begin{array}{lcl} S & \rightarrow & XBaB \\ X & \rightarrow & aY \mid bY \\ Y & \rightarrow & aY \mid bY \mid \epsilon \\ B & \rightarrow & bB \mid \epsilon \end{array}$$

Leftmost and rightmost Derivation example

Generate: aaabba

$$\begin{array}{lcl} S & \rightarrow & XBaB \\ X & \rightarrow & aY \mid bY \\ Y & \rightarrow & aY \mid bY \mid \epsilon \\ B & \rightarrow & bB \mid \epsilon \end{array}$$

Leftmost Derivation: $S \rightarrow XBaB \rightarrow aYBaB \rightarrow aaYBaB \rightarrow aaaYBaB \rightarrow aaaBaB \rightarrow aaabBaB \rightarrow aaabbBaB \rightarrow aaabbaB \rightarrow aaabba$

Rightmost Derivation: $S \rightarrow XBaB \rightarrow XBa \rightarrow XbBa \rightarrow XbbBa \rightarrow Xbba \rightarrow aYbba \rightarrow aaYbba \rightarrow aaaYbba \rightarrow aaabba$

The Language of the Grammar

If $G(V, T, P, S)$ is a CFG, then the language of G is

$$L(G) = \{w \ \ in \ \ T^* \mid S \overset{*}{\Rightarrow}_G \ w\}$$

i.e., the set of strings over T derivable from the start symbol.

If G is a CFG, we call $L(G)$ a context-free language.

Example: $L(G_{pal})$ is a context-free language.

Give a CFG for the CFL 0^n1^n , where n>=1

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \varepsilon$

- Start: A; $\Sigma = \{0, 1\}$
- Here is an example of using it to produce (or derive) a string:

$A \rightarrow 0A1$

$\rightarrow 00A11$

$\rightarrow 000A111$

….. …. …..

$\rightarrow 0^nA1^n$

$\rightarrow 0^nB1^n$

$\rightarrow 0^n1^n$

CFG example

1. Consider the grammar that generates “ababcbcbb”

   $S \rightarrow abScB \mid \lambda$

   $B \rightarrow bB \mid b$

2. {w | w starts and ends with the same symbol}

CFG example

1. {w | w starts and ends with the same symbol}

   $S \rightarrow 0A0 \mid 1A1$

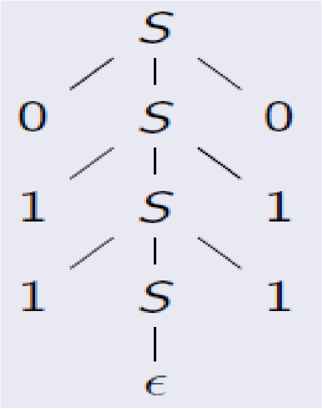
   $A \rightarrow 0A \mid 1A \mid \lambda$

Parse Trees(Derivation Trees)

- Top-Down Tree Representation for derivation
- Good way to visualize the derivation process
- How the symbols of a terminal string are grouped into substrings
- A picture of the structure that a grammar places on the strings of its language

Parse Trees

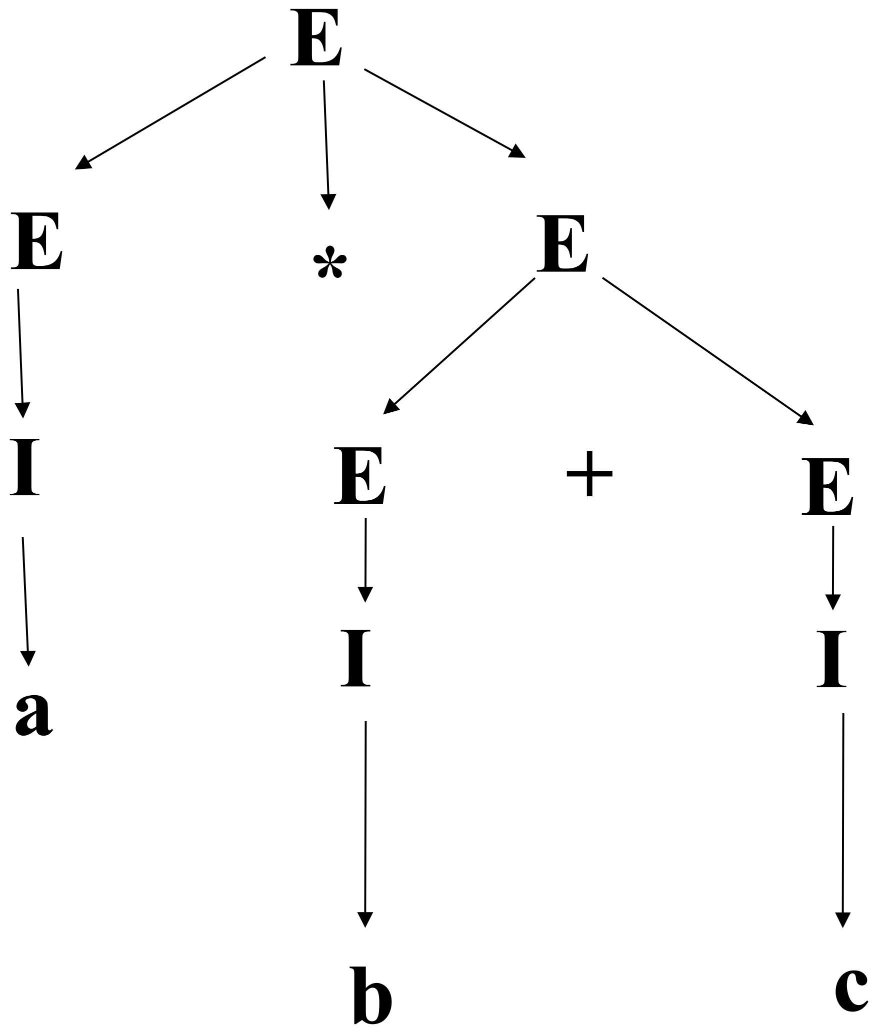
- For CFG G = (V;; R; S), a parse tree (or derivation tree) of G is a tree satisfying the following conditions:
  - Each interior node is labeled by a variable in V
  - Each leaf is labeled by either a variable, a terminal or ε ; a leaf labeled by ε must be the only child of its parent.
  - If an interior node labeled by A with children labeled by X1,X2,….Xk (from the left), then A→X1X2…Xk must be a rule.



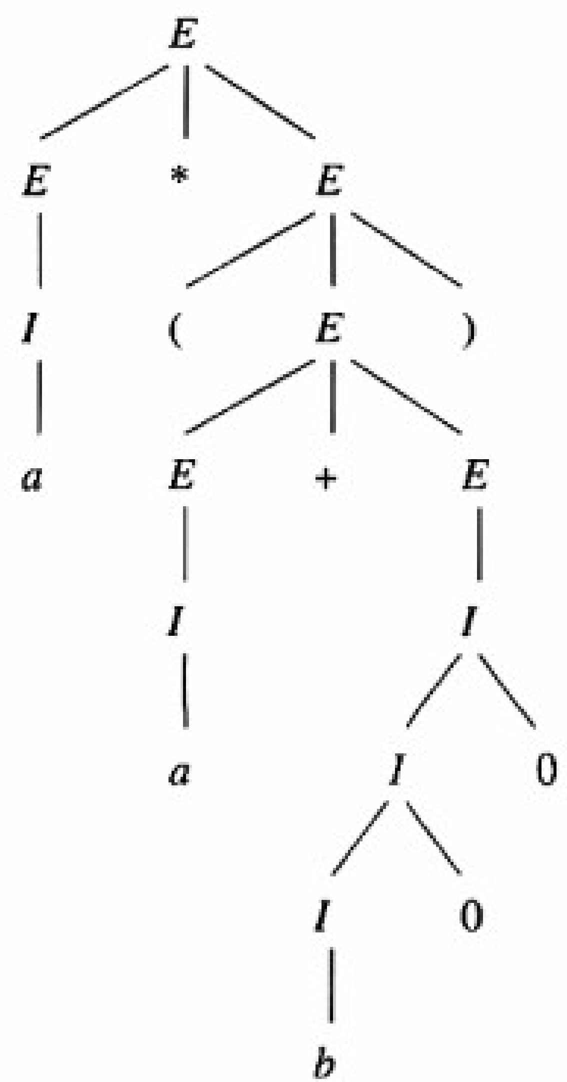
011110

Yield of a parse tree is the concatenation of leaf labels (left-right)

Parse Trees a*(b+c)

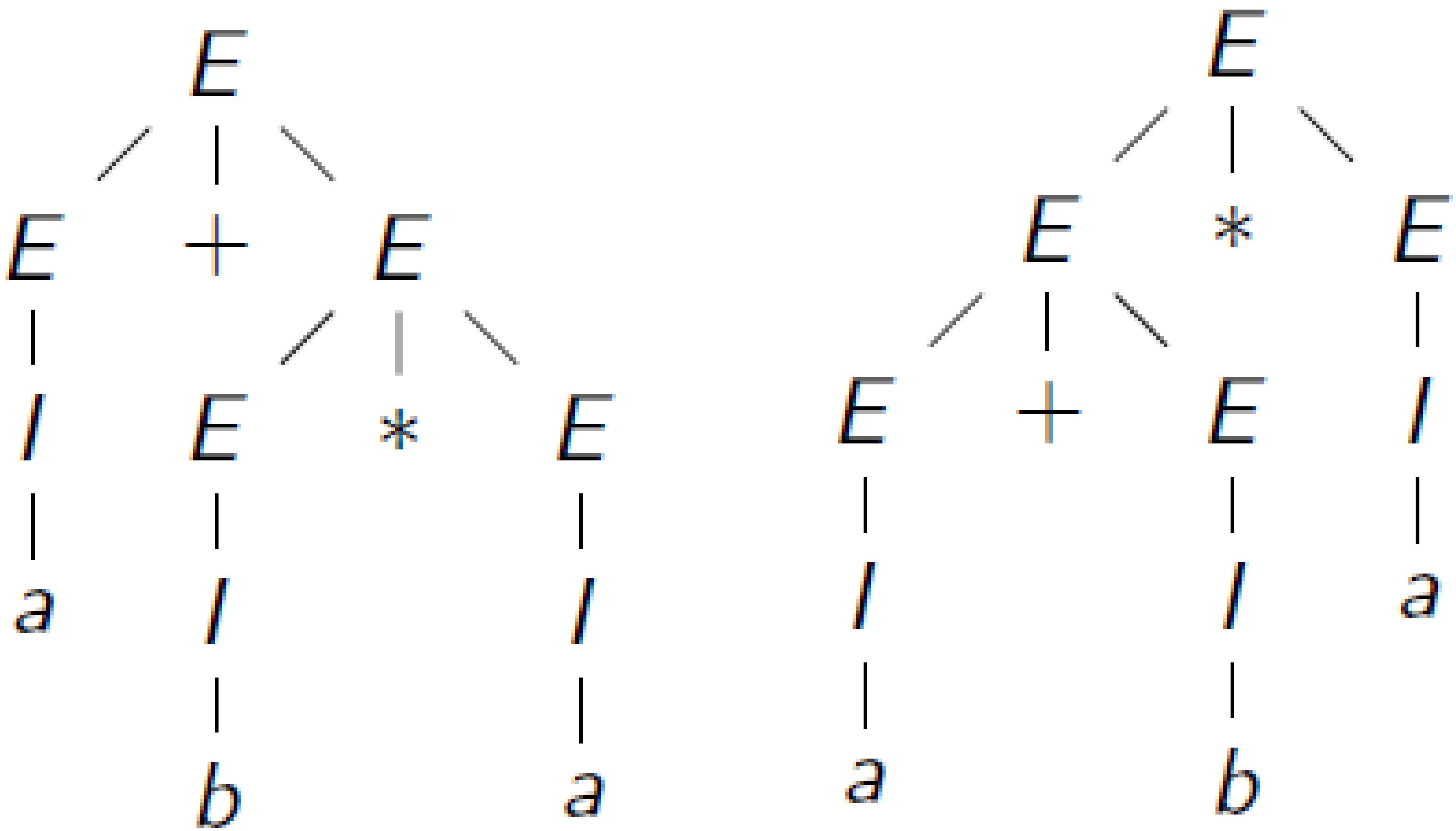


Parse Tree for: a*(a+b00)



Multiple Parse Trees

The parse trees for expression $a + b * a$ in the grammar G_{exp} is



Ambiguity: Grammar

A grammar $G = (V, \Sigma, R, S)$ is said to be ambiguous if there is $w \in \Sigma^*$ for which there are two different parse trees.

Removing Ambiguity

- Ambiguity maybe removed either by
  - Using the semantics to change the rules.
    - For example, if we knew who had the bat (the girl or the boy) from the context, we would know which is the right interpretation.
  - Adding precedence to operators.
    - For example, * binds more tightly than +, or “else" binds with the innermost “if".

Inherently Ambiguous Languages

A context-free language L is said to be inherently ambiguous if every grammar G for L is ambiguous.

Inherently Ambiguous Languages: Example

Consider

$$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

One can show that any CFG G for L will have two parse trees on $a^n b^n c^n$, for all but finitely many values of n

- One that checks that number of a's $=$ number of b's
- Another that checks that number of b's $=$ number of c's