FACULTY DEVELOPMENT PROGRAM ON THEORY OF COMPUTATION 18-05-2020 TO 22-05-2020

ORGANIZED BY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

VELAMMAL ENGINEERING COLLEGE

IN ASSOCIATION WITH

COMPUTER SOCIETY OF INDIA

CONTEXT FREE GRAMMAR

S.RAJALAKSHMI

DEPARTMENT OF COMPUTER SCIENCE AND ENGG

VELAMMAL ENGINEERING COLLEGE

OUTLINE OF THIS SESSION

- CFG – Introduction
- Ambiguity in a Grammar
- Closure Properties
- CFG simplification
- Chomsky Normal form
- Greibach Normal form
- Pumping Lemma

CONTEXT FREE GRAMMAR - DEFN

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

APPLICATIONS OF CFG

Context free languages have great practical significance.

CFLs are used by the compiler in the parsing phase as they define the syntax of a programming language and are used in many editors.

EXAMPLES FOR CFG

- The grammar ({A}, {a, b, c}, P, A), P : A → aA, A → abc.
- The grammar ({S, a, b}, {a, b}, P, S), P: S → aSa, S → bSb, S → ε
- The grammar ({S, F}, {0, 1}, P, S), P: S → 00S | 11F, F → 00F | ε

Sample problem

GENERATION OF 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.

TWO APPROACHES FOR PARSE TREE / DERIVATION TREE CONSTRUCTION

Top-down Approach –

- Starts with the starting symbol **S**
- Goes down to tree leaves using productions

Bottom-up Approach –

- Starts from tree leaves
- Proceeds upward to the root which is the starting symbol **S**

DERIVATION / YIELD OF A TREE

The derivation or the yield of a parse tree is the final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the Nulls. However, if all the leaves are Null, derivation is Null.

Example

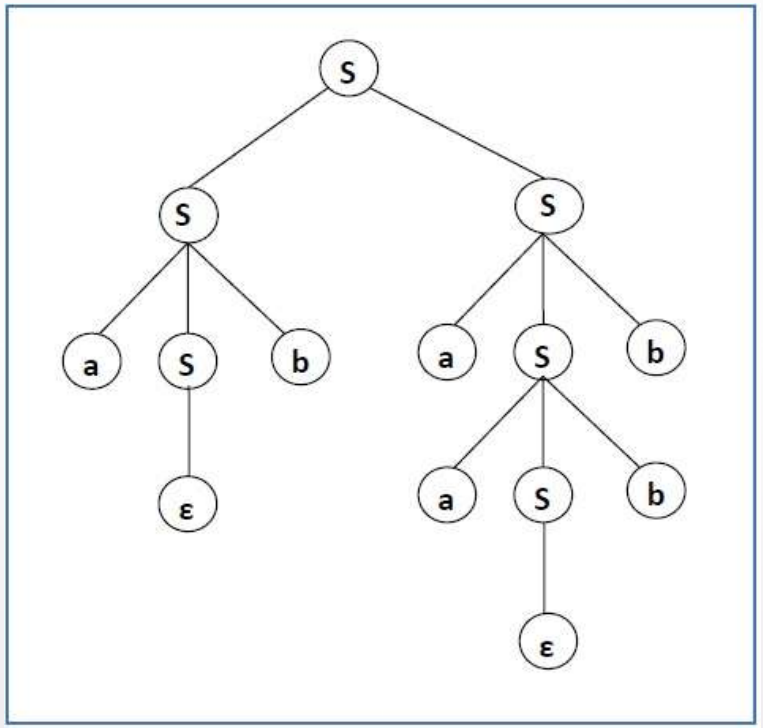
Let a CFG {N,T,P,S} be

N = {S}, T = {a, b}, Starting symbol = S, P = S → SS | aSb | ε

One derivation from the above CFG is "abaabb"

S → SS → aSbS → abS → abaSb → abaaSbb → abaabb

TREE REPRESENTATION



SENTENTIAL FORM AND PARTIAL DERIVATION TREE

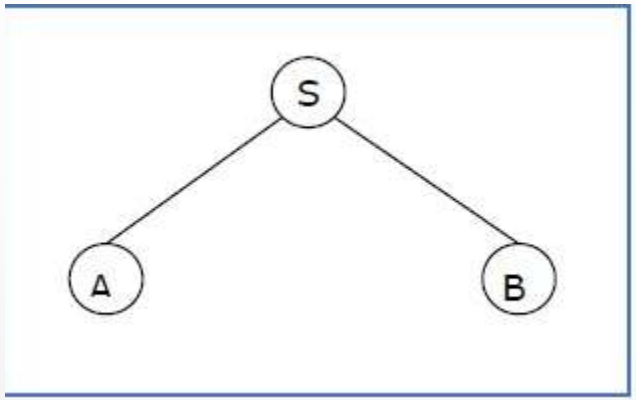
A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

Example

If in any CFG the productions are

$S \rightarrow AB, A \rightarrow aaA \mid \varepsilon, B \rightarrow Bb \mid \varepsilon$

TREE REPRESENTATION OF SENTINEL



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.

LEFTMOST DERIVATION - EXAMPLE

Let any set of production rules in a CFG be

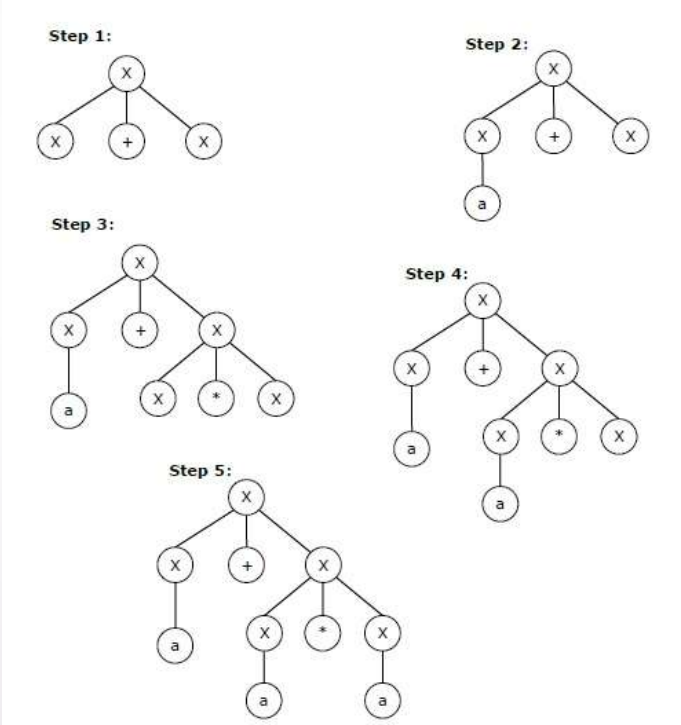
$X \rightarrow X+X \mid X*X \mid |X| \mid a$

over an alphabet {a}.

The leftmost derivation for the string **"a+a*a"** may be –

$X \rightarrow X+X \rightarrow a+X \rightarrow a + X*X \rightarrow a+a*X \rightarrow a+a*a$

LEFTMOST DERIVATION – EXAMPLE CONTD..,

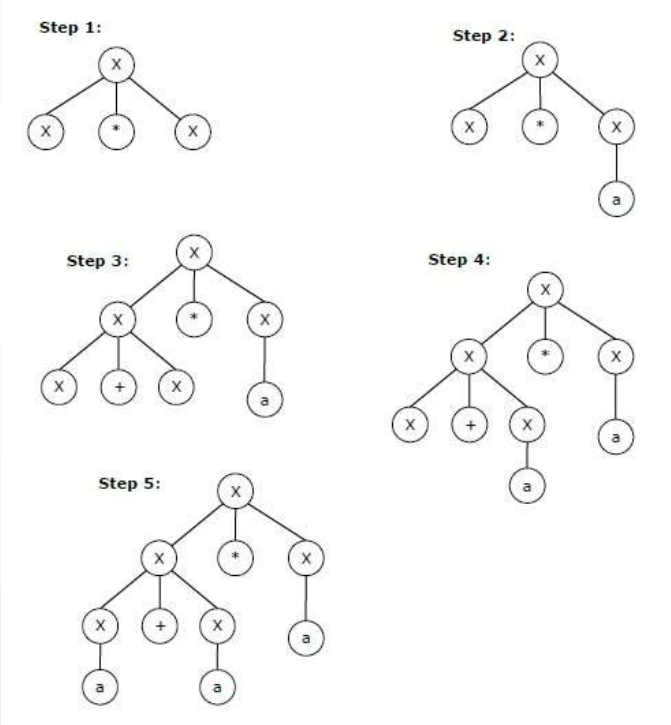


RIGHTMOST DERIVATION - EXAMPLE

The rightmost derivation for the above string **"a+a*a"** may be –

X → X*X → X*a → X+X*a → X+a*a → a+a*a

RIGHTMOST DERIVATION – EXAMPLE CONTD..,



Sample problem for LMD, RMD

AMBIGUITY IN A GRAMMAR

If a context free grammar **G** has more than one derivation tree for some string **w ∈ L(G)**, it is called an **ambiguous grammar**. There exist multiple right-most or left-most derivations for some string generated from that grammar.

PROBLEM

Check whether the grammar G with production rules −

X → X+X | X*X |X| a

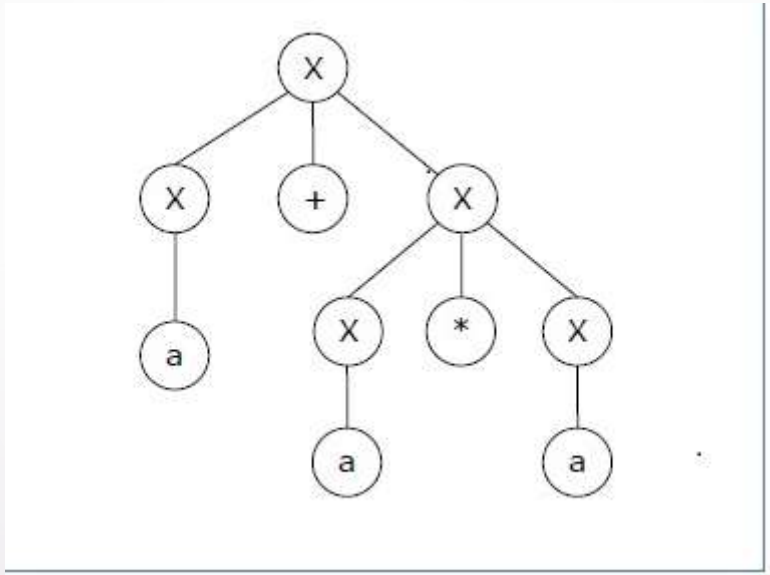
is ambiguous or not.

Solution

Let's find out the derivation tree for the string "a+a*a". It has two leftmost derivations.

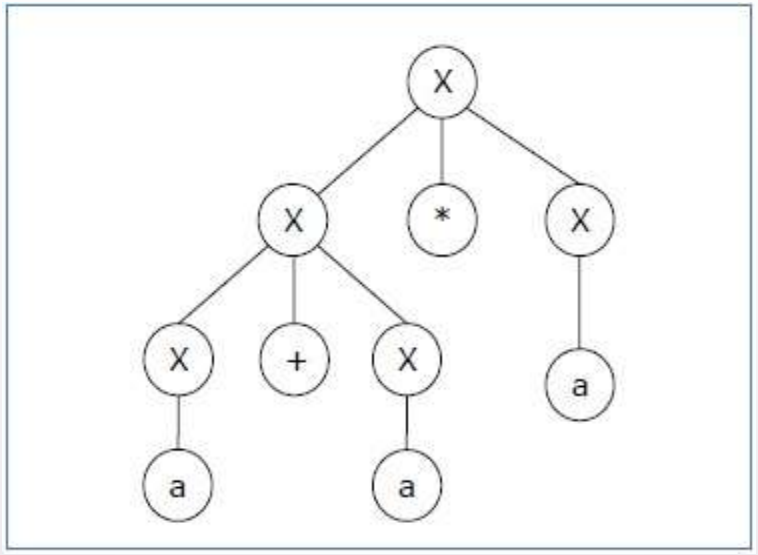
Derivation 1 − X → X+X → a +X → a+ X*X → a+a*X → a+a*a

TREE REPRESENTATION



Derivation 2 – X → X*X → X+X*X → a+ X*X → a+a*X → a+a*a

Parse tree 2 –



Sample problem for Ambiguous Grammar

CLOSURE PROPERTIES

Context-free languages are **closed** under –

- Union
- Concatenation
- Kleene Star operation

CFG SIMPLIFICATION

Simplification essentially comprises of the following steps −

- Removal of Null Productions
- Removal of Unit Productions
- Removal of Useless Symbols

REMOVAL OF UNIT PRODUCTION

Any production rule in the form A → B where A, B ∈ Non-terminal is called **unit production..**

Removal Procedure −

- **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.

REMOVAL OF NULL PRODUCTION

In a CFG, a non-terminal symbol **'A'** is a nullable variable if there is a production **A** → **ε** or there is a derivation that starts at **A** and finally ends up with ε: A → ..... → ε

Removal Procedure

- **Step 1** – Find out nullable non-terminal variables which derive ε.
- **Step 2** – For each production **A** → **a**, construct all productions **A** → **x** where **x** is obtained from **'a'** by removing one or multiple non-terminals from Step 1.
- **Step 3** – Combine the original productions with the result of step 2 and remove **ε - productions**.

REMOVAL OF USELESS SYMBOLS

- Identifying Non generating symbol
- Identifying Non reachable symbol

Sample problem for simplification of CFG

CHOMSKY NORMAL FORM

A CFG is in Chomsky Normal Form if the Productions are in the following forms –

$A \rightarrow a$

$A \rightarrow BC$

$S \rightarrow \varepsilon$

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

ALGORITHM FOR CFG -> CNF

- **Step 1** – If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S'**→ **S**.
- **Step 2** – Remove Null productions. (Using the Null production removal algorithm discussed earlier)
- **Step 3** – Remove unit productions. (Using the Unit production removal algorithm discussed earlier)
- **Step 4** – Replace each production $\mathbf{A \rightarrow B_1 \ldots B_n}$ where **n > 2** with $\mathbf{A \rightarrow B_1 C}$ where $\mathbf{C \rightarrow B_2 \ldots B_n}$. Repeat this step for all productions having two or more symbols in the right side.
- **Step 5** – If the right side of any production is in the form **A** → **aB** where a is a terminal and **A, B** are non-terminal, then the production is replaced by **A** → **XB** and **X** → **a**. Repeat this step for every production which is in the form **A** → **aB**.

Sample problem for Chomsky Normal form

GREIBACH NORMAL FORM

A CFG is in Greibach Normal Form if the Productions are in the following forms −

$A \rightarrow b$

$A \rightarrow bD_1 \ldots D_n$

$S \rightarrow \varepsilon$

where $A, D_1, \ldots, D_n$ are non-terminals and b is a terminal.

ALGORITHM FOR CFG -> GNF

- **Step 1** – If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S'** → **S**.
- **Step 2** – Remove Null productions. (Using the Null production removal algorithm discussed earlier)
- **Step 3** – Remove unit productions. (Using the Unit production removal algorithm discussed earlier)
- **Step 4** – Remove all direct and indirect left-recursion.
- **Step 5** – Do proper substitutions of productions to convert it into the proper form of GNF.

Sample problem for Griebach Normal form

WHY NORMAL FORMS?

- Normal forms can give us more structure to work with, resulting in easier parsing algorithms.
- Chomsky normal form enables a polynomial time algorithm to decide whether a string can be generated by a grammar.
- Greibach Normal Form (GNF) has several important applications in formal language theory. The importance of GNF is that a grammar of this kind always tells us what the first terminal symbol to be derived using any given rule will be.

PUMPING LEMMA FOR CFG

If **L** is a context-free language, there is a pumping length **k** such that any string **w** ∈ **L** of length ≥ **k** can be written as **w = uvwxy**, where **vx≠ ε**, **|vwx| ≤ k**, and for all **i ≥ 0, uv^iwx^iy ∈ L.**

Applications of Pumping Lemma

Pumping lemma is used to check whether a grammar is context free or not.

Problems in Pumping Lemma

Thank you

Any Queries?