Parul® University

THEORY OF COMPUTATION
CODE:303105306

UNIT 3: GRAMMAR

Parul®
University

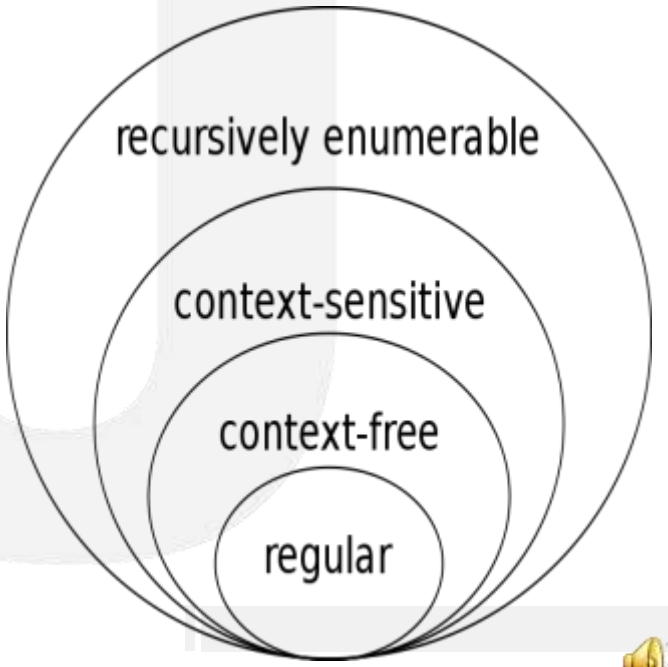
DIGITAL LEARNING CONTENT

CHAPTER-3

Grammar

Context Free Language

- CFLs are produced by context free language structure. The arrangement of all setting free dialects is indistinguishable from the set of dialects acknowledged by push down automata, and the set of regular language is a subset of CFG



Parul® University

Context Free Language

- **Grammar:**

-Sentence structure is a lot of rules which check whether a string have a place with a specific language or not. This defined Sentence structure is called Grammar

-Syntax of language is defined by this notation

- Designing of Parser is done by the help of CFG

Parul® University

Derivation Tree/ Parse Tree

- **Generation of Derivation Tree**

  - A deduction tree or parse tree is an arranged established tree that graphically speaks to the semantic data a string got from a CFG.

- **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.

Image source : Google

Approaches to draw a derivation tree

- **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 of a Tree

- The induction or the yield of a parse tree is the last string acquired by connecting the marks of the leaves of the tree from left to right, overlooking the Nulls. In any case, if all the leaves are Null, deduction is Null

Derivation of a Tree : 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”

Answer:-

S → SS

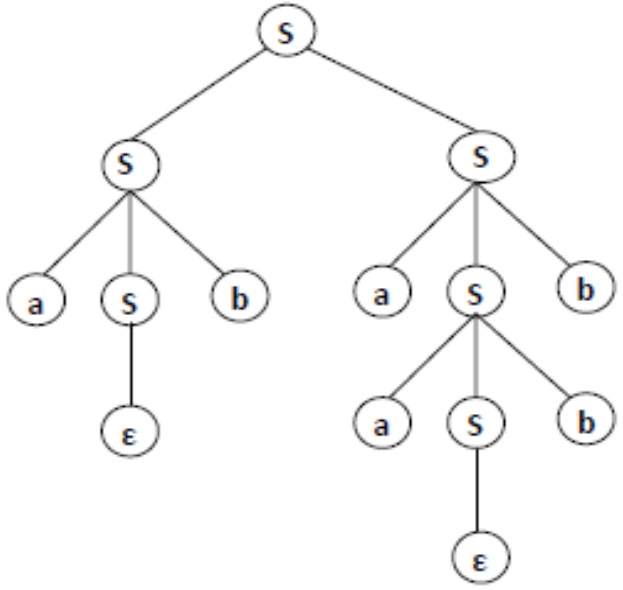
S → aSbS (replace S →aSb)

S →abS (replace S →ε)

S → abaSb (replace S →aSb)

S → abaaSbb (replace S →aSb)

S→ abaabb (replace S →ε)



Sentential Form and Partial Derivation Tree

•A fractional deduction tree is a sub-tree of an induction tree/parse tree with the end goal that either the entirety of its kids are in the sub-tree or none of them are in the sub-tree

•Example

In the event that in any CFG the creations/production are:

S → AB, A → aaA | ε, B →Bb| ε

•The fractional deduction tree can be the accompanying:

-In the event that an incomplete determination tree contains the root S, it is known as a sentential form.

-The above example is likewise in sentential form.

Parul®
University

Types of Derivation Tree

- Leftmost and Rightmost Derivation of a String

- Leftmost derivation: A furthest left induction is acquired by applying creation to the furthest left factor in each progression

- Rightmost derivation: A furthest right deduction is acquired by applying creation to the furthest right factor in each progression.

Left and Right Recursive Grammars

•In a CFG "G", if there is a creation in the structure X → Xa where X is a non-terminal and 'a' will be a series of terminals, it is known as a left recursive creation. The syntax having a left recursive creation is known as a left recursive production

•Also, if in a CFG "G", if there is a creation is in the structure X → aX where X is a non-terminal and 'a' will be a series of terminals, it is known as a right recursive production. The language structure having a privilege recursive creation is known as a right recursive production.

Ambiguity in Context-Free Grammars

•On the off chance that a setting free sentence structure G has more than one induction tree for some string w ∈ L(G), it is called an ambiguous grammar. There exist various right-most or left-most determinations for some string produced from that language structure.

CFG Simplification

•In a CFG, it might happen that all the production rules and symbol are not required for the determination of strings. Furthermore, there might be some invalid production and unit creations. End of these production and symbols is called disentanglement of CFGs.

Steps to simply

•Reduction of CFG

•Removal of Unit Productions

•Removal of Null Productions

Reduction of CFG

Derivation of an equivalent grammar, G', from the CFG, G, such that each variable derives some terminal string.

Derivation Procedure Part 1:

Step 1: Include all symbols, W1, that derive some terminal and initialize i=1.

Step 2: Include all symbols, Wi+1, that derive Wi.

Step 3: Increment i and repeat Step 2, until Wi+1 = Wi.

Step 4: Include all production rules that have Wi in it.

Reduction of CFG

Derivation of an equivalent grammar, G", from the CFG, G' such that each symbol appears in a sentential form.

Derivation Procedure part 2:

Step 1: Include the start symbol in Y1 and initialize i = 1.

Step 2: Include all symbols, Yi+1, that can be derived from Yi and include all production rules that have been applied.

Step 3: Increment i and repeat Step 2, until Yi+1 = Yi.

Removal of Unit Productions

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 Productions

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:

Step1 Find out nullable non-terminal variables which derive ϵ.

Step2 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.

Step3 Combine the original productions with the result of step 2 and remove ϵ-productions.

Chomsky Normal Form

In formal language theory, a CFG ”G ” is said to be in Chomsky normal form if all of its production rules are of the form:

Q → RE,
or Q → f,
or S → ε,

Here Q,R,E are Non terminals and S is starting symbol. f and ε(epsilon) are terminals where ε represent NULL

Greibach Normal Form

In formal language theory, a CFG is in Greibach normal form (GNF) if the right-hand sides of all production rules start with a terminal symbol, optionally followed by some variables.

Either that in the form:

$P \rightarrow aQWERTY$

$P \rightarrow a$

$S \rightarrow \varepsilon$

Here P,Q,W,E,R,T,Y are non terminals and S is start symbol. ε and a are terminals

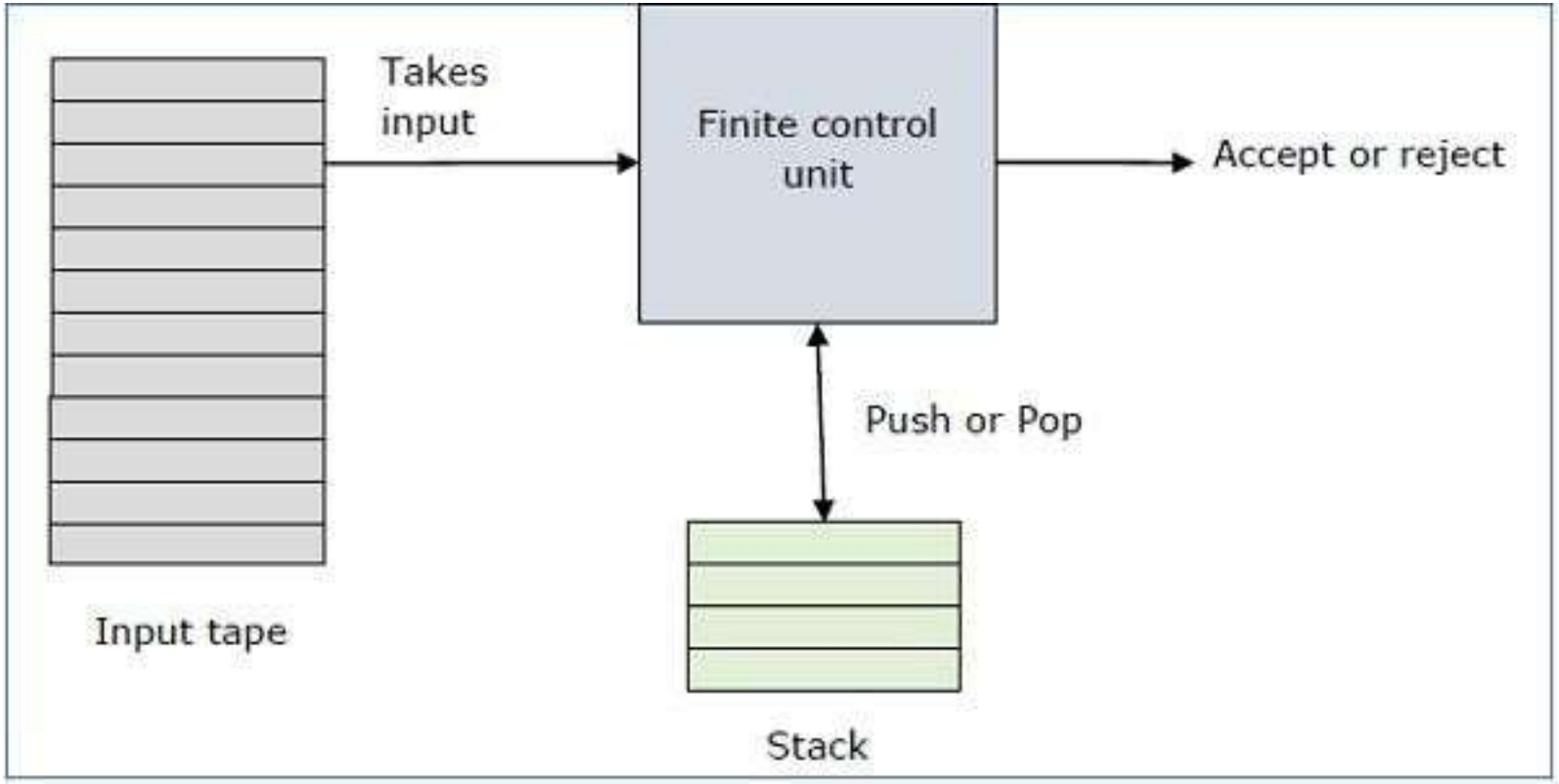
Non Deterministic Push down Automata

- A nondeterministic pushdown automaton (npda) is basically an nfa with a stack added to it.
- A nondeterministic pushdown automaton or npda is a 7-tuple

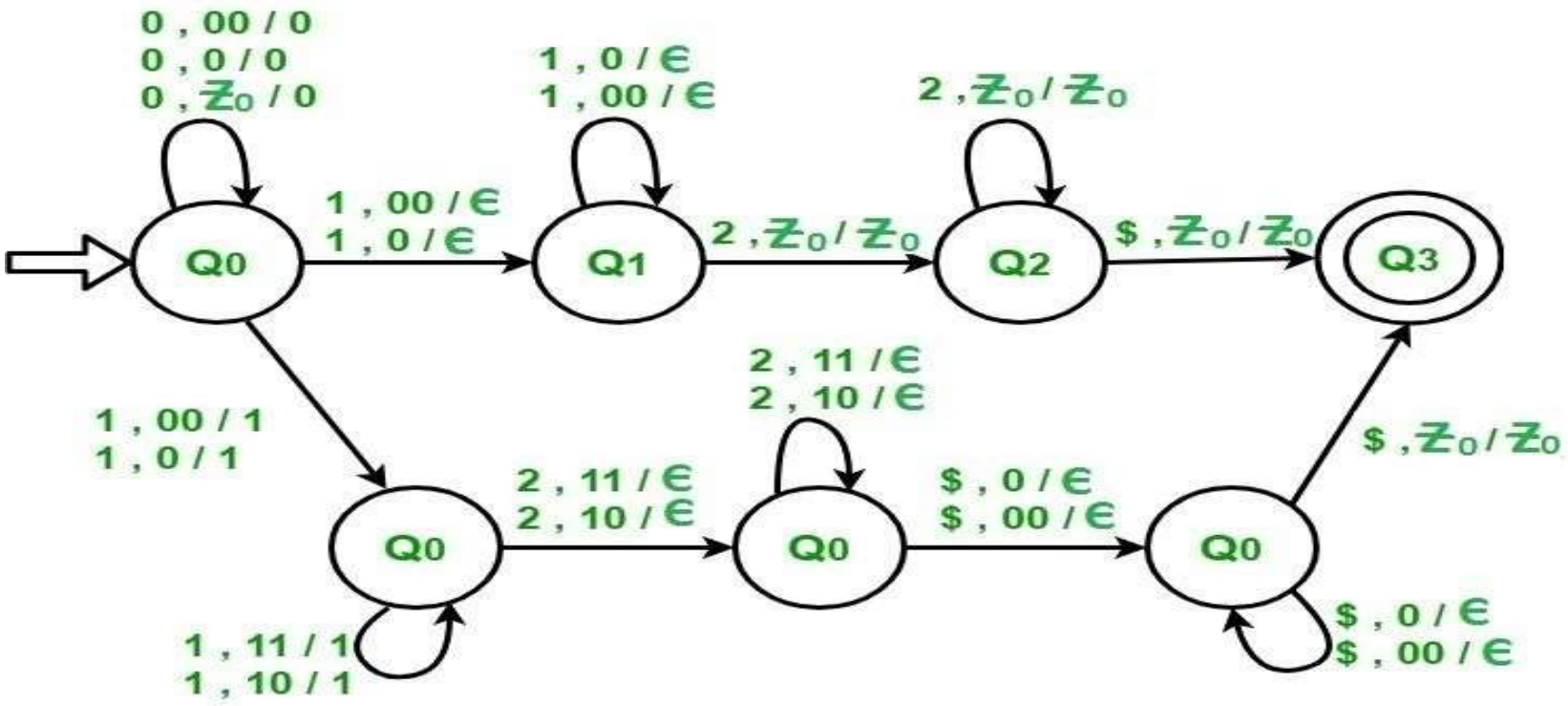
$$M = (Q, \sum, \Gamma, \delta, q0, z, F)$$

- Q is a finite set of states,
  - $\sum$ is a the input alphabet,
  - Γ is the stack alphabet,
  - δ is a transition function,
  - $q0 \in Q$ is the initial state,
  - $z \in \Gamma$ is the stack start symbol, and
  - $F \subseteq Q$ is a set of final states.

Block diagram of PDA



Example of NPDA



[1]https://www.google.co.in/imgres?imgurl=https%3A%2F%2Fmedia.geeksforgeeks.org%2Fwp-content%2Fuploads%2F33333.jpg&imgrefurl=https%3A%2F%2Fwww.geeksforgeeks.org%2Fnpda-for-l-0i1j2k-ij-or-jk-i-j-k-1%2F&tbnid=7m8Az1XEry4VSM&vet=12ahUKEwjS3IKF-tbpAhVQRSsKHa-1DH

PDA equivalence to CFG

•Algorithm to find PDA corresponding to a given CFG:

Input − A CFG, G = (V, T, P, S)
Output − Equivalent PDA, P = (Q, ∑, S, δ, q0, I, F)

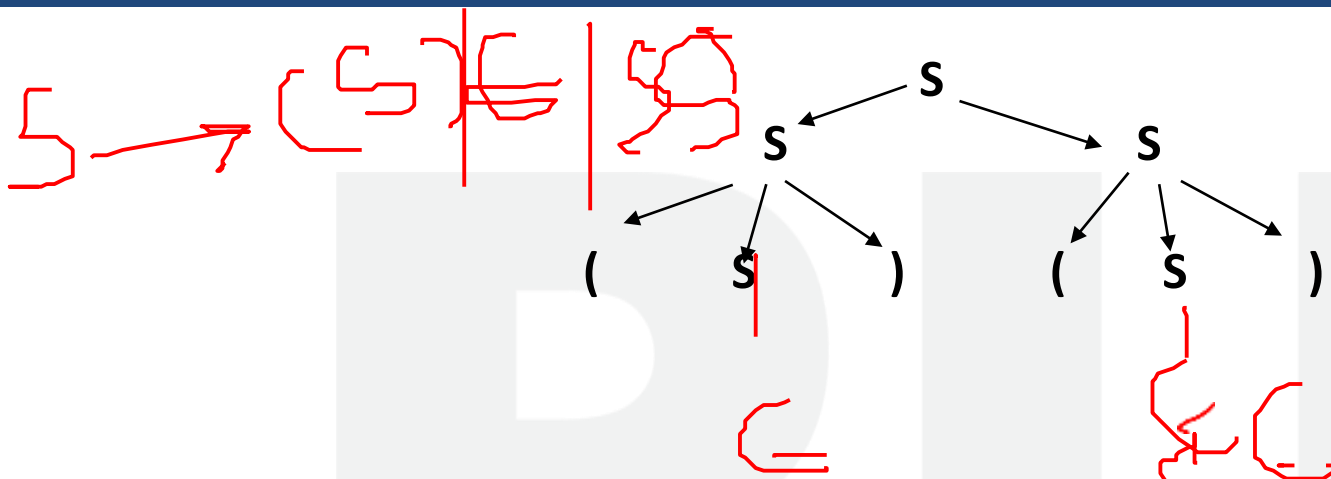
Step 1 − Convert the productions of the CFG into GNF.
Step 2 − The PDA will have only one state {q}.
Step 3 − The start symbol of CFG will be the start symbol in the PDA.
Step 4 − All non-terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA.
Step 5 − For each production in the form A → aX where a is terminal and A, X are combination of terminal and non-terminals, make a transition δ (q, a, A)

What is Parse Tree??

Parse trees are a representation of derivations that is much more compact. Several derivations may correspond to the same parse tree. For example, in the balanced parenthesis grammar, the following parse tree:

Image source : Google

What is Parse Tree??



Corresponds to the derivation S ⇒ SS ⇒ S(S) ⇒ (S)(S) ⇒ (S)() ⇒ ()()

as well as this one:

S ⇒ SS ⇒ (S)S ⇒ (S)(S) ⇒ ()(S) ⇒ ()()

Parul® University

Parse Tree

- In a parse tree, the points are called nodes. Each node has a label on it.
- The topmost node is called the root. The bottom nodes are called leaves.
- In a parse tree for a grammar G, the leaves must be labelled with terminal symbols from G, or with ϱ. The root is often labeled with the start symbol of G, but not always.
- If a node N labeled with A has children N1, N2, . . . , Nk from left to right, labeled with A1, A2, . . . , Ak, respectively, then A → A1A2, . . . Ak must be a production in the grammar G.

Image source : Google

Parse Tree

- The yield of a parse tree is the concatenation of the labels of the leaves, from left to right. The yield of the tree above is ()().
- Leftmost and Rightmost Derivations

- In a leftmost derivation, at each step the leftmost nonterminal is replaced. In a rightmost derivation, at each step the rightmost nonterminal is replaced.
- Such replacements are indicated by $\underset{\rightarrow}{R}$ and $\underset{\rightarrow}{L}$ respectively.
- Their transitive closures are R* and L* respectively.

Parse Tree

- In the balanced parenthesis grammar, this is a leftmost derivation:

S ⇒ SS ⇒ (S)S ⇒ ()S ⇒ ()(S) ⇒ ()().

This is a rightmost derivation:

S ⇒ SS ⇒ S(S) ⇒ S() ⇒ (S)() ⇒ ()()

Ambiguity In CFG

- A context-free grammar G = (V, Σ, R, S) is ambiguous if there is some string w ∈ Σ ∗ such that there are two distinct parse trees T 1 and T 2 having S at the root and having yield w.
- Equivalently, w has two or more leftmost derivations, or two or more rightmost derivations.
- Note that languages are not ambiguous; grammars are. Also, it has to be the same string w with two different (leftmost or rightmost) derivations for a grammar to be ambiguous.

Ambiguity In CFG

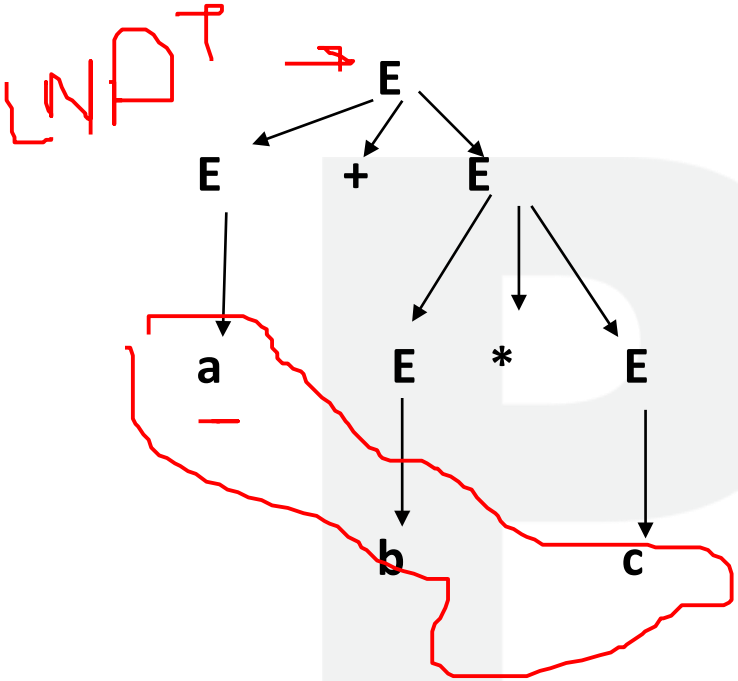
- Here is an example of an ambiguous grammar:
- E → E + E
- E → E * E
- E → (E)
- E → a
- E → b
- E → c

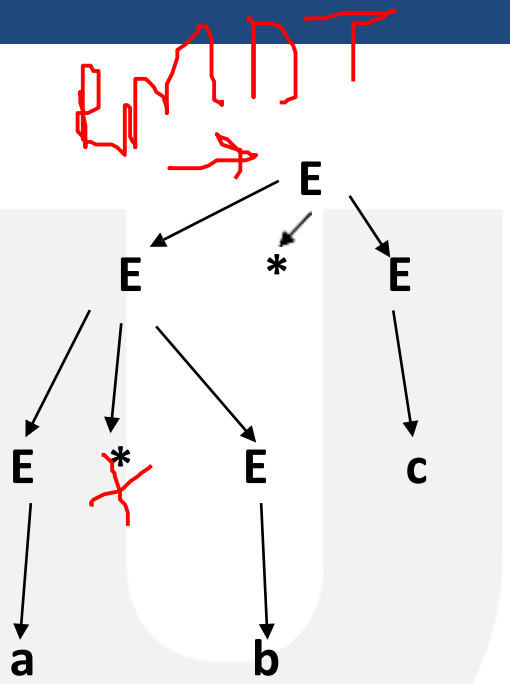
Ambiguity In CFG

- In this grammar, the string a + b ∗ c can be parsed in two different ways corresponding to doing the addition before or after the multiplication. This is very bad for a compiler, because the compiler uses the parse tree to generate code, meaning that this string could have two very different semantics. Here are two parse trees for the string a + b ∗ c in this grammar:

  Here are two parse trees for the string a + b ∗ c in this grammar:

Ambiguity In CFG





Ambiguity In CFG

- There is a notion of inherent ambiguity for context-free languages; a context-free language L is inherently ambiguous if every context-free grammar G for L is ambiguous. As an example, the language{$a^n b^n c^m d^m : n \geq 1, m \geq 1$} U {$a^n b^m c^m d^n : n \geq 1, m \geq 1$} is inherently ambiguous. In any context-free grammar for L, some strings of the form $a^n b^n c^n d^n$ will have two distinct parse trees.

  Unfortunately, the problem of whether a context-free grammar is ambiguous, is undecidable. However, there are some patterns in a context-free grammar that frequently indicate ambiguity:

Ambiguity In CFG

- S → SS
- S → a
- S → A
- A → AA
- A → a
- S → AA
- A → S
- A → a
- S → SbS

Ambiguity in CFG

- S → a
- S → AbA
- A → S
- A → a

Ambiguity In CFG

The following is not ambiguous:

S → aS

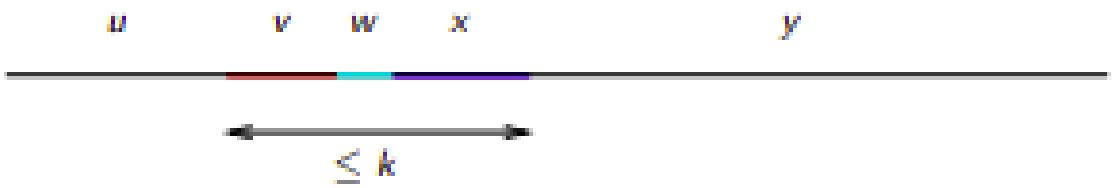
S → bS

S → ε

In general, a production A → AA causes ambiguity if it is reachable from the start symbol and some terminal string is derivable from A.

Pumping Lemma of CFG

- For every CFL L there is a constant k ≥ 0 such that for any word z in L of length at least k, there are strings u, v,w, x, y such that
- z = uvwxy,
- Vx ≠ ϵ
- |vwx| ≤ k,
- and for each i ≥ 0, the string uv^iwx^iy belongs to L.



Parse trees for CFG's

Derivation can be represented as parse Tree

CFG : G_2

S--->aSb |€ w=aaaabbbb

Derivation :

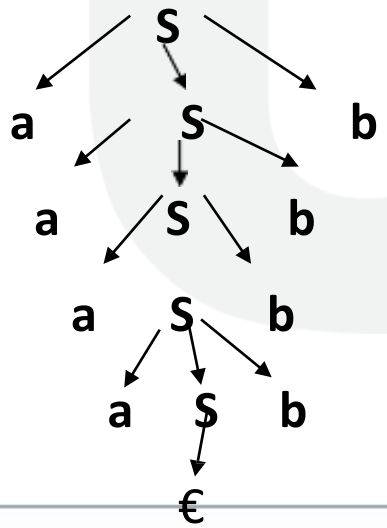
S--->aSb

--->aaSbb

--->aaaSbbb

--->aaaabbbb

Parse Tree



Deterministic Push Down Automata

Two Transitions ((p1,u1,v1),(q1,z1)) and ((p2,u2,v2),(q2,z2)) are compatible if

1. **p1=p2**
2. **U1=u2 are compatible (which means that u1 is a prefix of u2 or that u2is prefix of u1).**
3. **V1 and v2 are compatible.**

Parul® University

Deterministic Push Down Automata

- A pushdown automaton is deterministic if for every pair of compatible transitions, these transitions are identical.
- Let L be a language defined over the alphabet Σ, the language L is deterministic context-free if and only if it is accepted by a deterministic pushdown automaton.

Deterministic Push Down Automata

Not all context free language s are deterministic context –free

L1={wcw^R | w $\in\{a,b\}^*$ } is deterministic contect-free.

L2={ww^R | w $\in\{a,b\}^*$ } is context free but notdeterministic context-free.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in