CONTEXT FREE GRAMMAR :-

A grammer CFG, G, is a structure :-

variables / non terminals

$$G = \langle \Sigma, V, R, S \rangle$$

Σ → terminals (small letters) ; V → CAPS ; R → rules

r1 r2 r3

$$E = a \mid E + E \mid E * E$$

(production rule)

a → terminal

$$\Sigma = \{a, +, *\} \qquad V = \{E\}$$

$$R \subseteq V \times (\Sigma \cup V)^* \quad (\Sigma \cup V)^* \rightarrow \text{string} \rightarrow \text{serial form}$$

$$R = \{(E, a), (E, E*E), (E, E+E)\}$$

$S \in V$

$E \overset{r_2}{\Rightarrow} E + E \overset{r_1}{\Rightarrow} E + a \overset{r_1}{\Rightarrow} a + a$

variable position reader

reader

Sentence is a sentential form without variables

Prog. Eg:- $S ::= N_p \; V_p$

$N_p ::= D \; N$

$V_p ::= V \; N_p \mid V$

determinat D = the | a

Nouns N = man | woman.

$V ::=$ greets

G:

Σ : a, the, man, woman, greets

V = S, N_p, V_p, N, V, D

Based on the rules of the Grammar

1 step reduction

Multiple step reductions gives you

→ Let $a = (V, \Sigma, R, S)$ be a CFG :-

Defn ($\Rightarrow$) (yields)

$\overset{G}{\Rightarrow} \quad \subseteq \quad (\Sigma \cup V)^* \times (\Sigma \cup V)^*$

definition:

$$\alpha A \beta \overset{G}{\Rightarrow} \alpha \gamma \beta$$

$$(A, \gamma) \in R$$

One sentential form yields to another

Reflexive Transitive Closure:

: Apply the rule 'yields' 0 or more times.

Eg: $E * E + E \overset{r_3}{\Rightarrow} a * E + E$

$E * E + E \overset{r_1}{\Rightarrow} E * E * E + E$

Derivation WRT G is the reflexive, transitive closure of $\Rightarrow$ & is denoted by $\overset{*}{\Rightarrow}$

$L(G) = df : \{ \omega \in \Sigma^* \mid S^* \Rightarrow \omega \}$

Set of all sentences (× sentential form) that can be derived from the start symbol of the grammar

Language is context free by definition if it is generatable by a grammar.

⚠ The sequence of rules applied determine the string!

→ Each derivation can have 2 different ~~derivation~~ witnesses.

ie:-

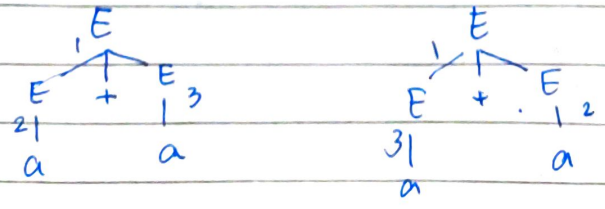
$S \Rightarrow \ldots\ldots \Rightarrow \omega$ one route of yields

$S \Rightarrow \ldots\ldots \Rightarrow \omega$ another yield route

Eg:- $E + E \Rightarrow a + E \Rightarrow a + a$.

(or) $E + E \Rightarrow E + a \Rightarrow a + a$] ω.

Parse Tree Rep:

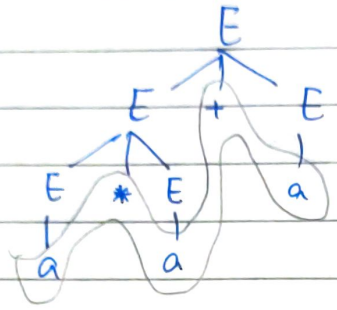


Fringe of the parse tree = leaves

Always choose the leftmost expression as the Redex

Type 1: $E \Rightarrow^{r1} E*+E \Rightarrow^{r2} E*E+E \Rightarrow^{r3} a*E+E \Rightarrow^{r3} a*a+E$

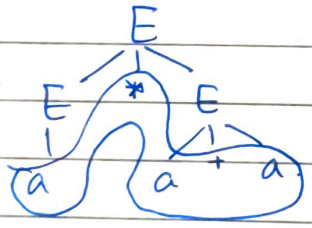
$\Rightarrow a*a+a$



What occurs if the leaf is either a leaf of epsilon

Type 2: $E \Rightarrow^{r2} E*E \Rightarrow^{r3} a*E \Rightarrow r_1 a*E+E$

$\Rightarrow a*a+E \Rightarrow \boxed{a*a+a}$



Problem: Two parse trees cause an ambiguity in the way you parse the strings

~~No grammar possible~~ Q: Can you find a grammar that doesn't have this prob.

~ Q: Is it possible to have such a grammar

▲

- $E' \rightarrow T+E' \mid T$
- $T \rightarrow F*T \mid F$
- $F \rightarrow a.$

G_A

T: Term

F: Factor

Now variable set becomes $V = \{E, T, F\}$
& $S = E$

Claim: This G = The old G
languages generated are same

Prob: Some languages are inherently ambiguous
Amb: Grammar that has a sentence that has more than 1 parse tree.

~~Converting Ambiguous Languages~~

→ 2 context free languages are ~~~~ closed under union.

Consider languages: $\{a^n b^n c^m \mid n, m \geq 0\}$ ∪
$\{a^m b^n c^n \mid n, m \geq 0\}$

$\therefore$ $a^k b^k c^k$ ... which rule? ... ambiguous

$L = \{a^n b^n \mid n \in \mathbb{N}\}$ $S \to \epsilon \mid aSb$

→ Given a CFG G for a language L we want a CFG G' such that $L(G') = L^*$

$G' = \langle V', \Sigma', R', S' \rangle$
$\Sigma' = \Sigma$
$V' = V \cup \{S'\}$
$R = ?$
Start symbols of G' is S'.

$S' \to SS' \mid \epsilon$

2 grams (Must are equal) gen exactly the same set of sentences

- Content-Free lang not closed under intersection

* Getting rid of the ambiguity

G'_A =>

$E' \rightarrow T + E' \mid T$ — ①

$T \rightarrow F \times T \mid F$ — ②

$F \rightarrow a.$ — ③

Now modifying ②

$T \rightarrow a \times T + a$ .... regex

$T = (a\times)^* a$ — (β)

Further :- $E' \rightarrow T + E' \div T$

$E' = (T+)^* T \rightarrow (\alpha)$

→ To show the two grammars $G_A = G'_A$

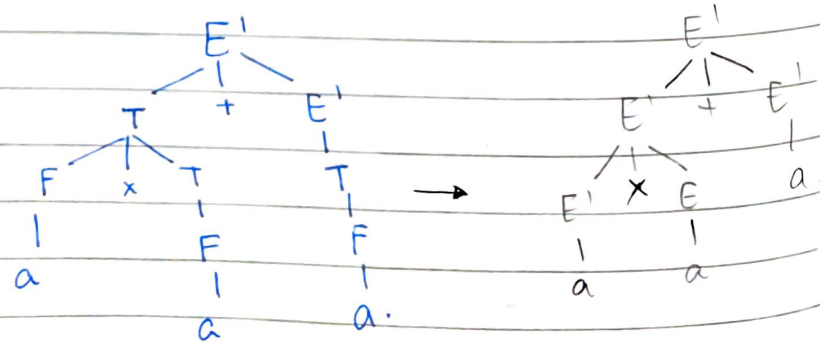
→ To show that the language $-(\alpha)$ is generated by G_A

→ TO show $L(G_A) = L(G_A')$

① $L(G_A) \subseteq L(G_A')$

② $L(G_A') \subseteq L(G_A)$

Drawing the parse tree for G_A'



Prunate vars & remove redundant terms

Parse tree turn every step of the derivation into a tree

classmate Date Page

Proving ① of proof

$w \in T$ base case for (α)

Now run induction on (β)

$T = \alpha \ldots$ base

- **Abstract Syntax Tree:** (rustanalysis.com)

- Used in compilers.
- Parsers generates an AST.

what was the CST for 2+3?

Derivation happens Top Down

Parsing happens Bottom Up.

Thm: $\vdash_{GA} e : w\ E$

evidence that $E \overset{*}{\Rightarrow} w$

Rules:

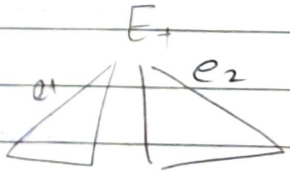
$$\frac{}{Ea() : a\ E}\ Ea$$

.... essentially a constructor.

$$\frac{e_1 : w_1\ E \qquad e_2 : w_2 E}{E_+(e_1, e_2) : w_1 + w_2 E}\ E_+$$

AST $\leftarrow E_+(e_1, e_2) : w_1 + w_2 E$

PT $\leftarrow E(e_1 + e_2)$



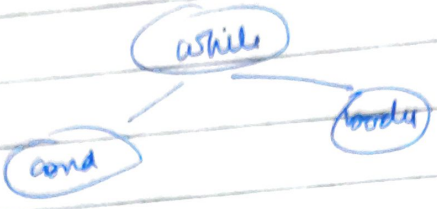
$$\frac{e_1 : w_1\ E \qquad e_2 : w_2 E}{E_*(e_1, e_2) : w_1 \times w_2 E}\ E_*$$

AST is better as none of the ~~non-terminals~~ terminals are excluded. It is more concise & apt.

Eg: while (x>0) {
    x++;
}

{} → not req. in a tree.

AST:



CHOMSKY NORMAL FORM:

A CFG G = (V, Σ, R, S) is in the Chomsky normal form if every production in G is of the form

1. $A \rightarrow BC$
2. $A \rightarrow a$ or
3. $S \rightarrow \epsilon$ where

$BC \in V \setminus \{S\}$

$a \in \Sigma$.

TUTORIAL :

Consider the grammars :-

1. $A \rightarrow A \mid \epsilon$
2. $A \rightarrow A+A \mid a$
3. $S \rightarrow A+A$

   $A \rightarrow 0 \mid 1$
4. $A \rightarrow A+A \mid A*A \mid a$.

A grammar is ambiguous if we can generate the same string via more than 1 parse tree following the leftmost redex rule. (derivation)

→ ① is ambiguous :

$\varepsilon = A \to \varepsilon$

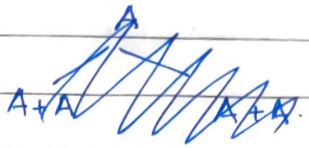
$\varepsilon = A \to A \to \varepsilon$

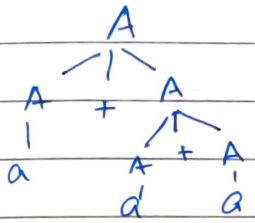
Converting to non ambiguous

$S \to A$ | $A \to \varepsilon$

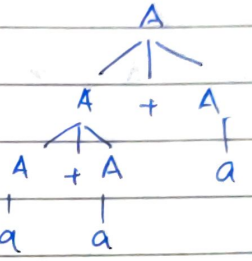
$A \to \epsilon$

→ ② is ambiguous





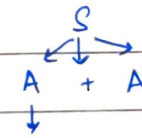
$a + (a+a)$



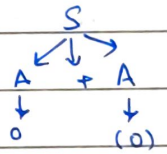
$(a+a) + a$.

→ ③ $S \to A + A$.

$A \to 0 \mid 1$



$(0) + 0$



Incorrect derivation

∵ doesn't follow leftmost rule

Pf: Only 4 possible strings generatable by this grammar ∎

N Correction for ② :-

$A \to A + a \mid a$ (OR) $A \to a + A \mid a$.

→ Consider the grammar accepting strings $a^i b^j c^k$ s.t. $i=j$ or $j=k$
- Inherently ambiguous.

- CNF Chomsky Normal Form

(i) No ε (except start state
(ii) $A \rightarrow BC \mid a$.
↓ 2 vars ↘ or 1 terminals.

All CFGs can be rep as CNF
Benefit: certain algos are written for CNF.

Q: Convert to CNF

Eg: $S \rightarrow ASA \mid aB$ ← trouble
$A \rightarrow B \mid S$
$B \rightarrow b \mid \varepsilon$ ← trouble

Book: You can have a b or ε in B → output

Modif:- $S \rightarrow ASA \mid aB \mid a$
$A \rightarrow B \mid S \mid \varepsilon$ - ✱
$B \rightarrow b$

delete start ε

Move to ✱ $A \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid$ ~~S~~
$A \rightarrow B \mid S$
$B \rightarrow b$

unit rule elimination

Unit transitions like $A \rightarrow B$ not allowed
∴ replace B with b everywhere

Pushdown automaton: class of machines identifying CFL.

$S' \rightarrow S$ unit

$\Rightarrow S' \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid \varepsilon$

$S \rightarrow \varepsilon$

$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$

$B \rightarrow b$

Great Step XD to remove

Let $U \rightarrow SA$

$V \rightarrow a$

$S' \rightarrow AU \mid VB \mid a \mid SA \mid AS$

$S \rightarrow \varepsilon$

$A \rightarrow b \mid AU \mid VB \mid a \mid SA \mid AS$

$B \rightarrow b$

$U \rightarrow SA$

$V \rightarrow a$.

ORDER is important!

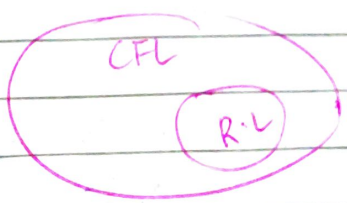
- BIN before DEL gives linear increase
- DEL before BIN gives exponential increase

~ logically del introduces more rules ∴ it causes a ~~more~~ lengthy derivation

START → BIN → DEL ... continue

↓ UNIT → TERM

All regular languages are context free but not vice versa



- Given a string w & a CFG $\langle G \rangle$

ST: $G = \langle V, R, S, \Sigma \rangle$

Find if $S \overset{*}{\Rightarrow} w$.

Let us assume WLG G is in CNF

→ CYK Algorithm: (Uses DP)

Let us assume $w \neq \epsilon$

Let $w = w_1 w_2 \dots w_k$ where $w_i \in \Sigma$

→ Idea: Look for a substring that is generated by a variable

ie: $\exists X \quad X \overset{*}{\Rightarrow} w_i \dots w_j$ (specific)

We want $i=1 \quad j=|w| \quad X \Rightarrow S$

Define set $M_{ij} = \{ X \in V \mid X \overset{*}{\Rightarrow} w_i \dots w_j \}$

Now,

w is accepted if

$S \in M_{1|w|}$

CNF prod:-

$S \to \epsilon$ empty

$S \to a$ Ground

$S \to XY$ Dynamic

Let $g(a) = \{ X \in V \mid X \to a \in P \}$ ... ground

? Also $M_{ii} = g(w_i)$

Suppose $Y \in M_{ik}$ then $Y \overset{*}{\Rightarrow} w_i \dots w_k$

$Z \in M_{(k+1)j}$ $Z \overset{*}{\Rightarrow} w_{k+1} \dots w_j$

$\therefore$, $X \xrightarrow{*} w_i \ldots w_j$

$X \rightarrow YZ$.

Suppose $M = \{A, B\}$ $M' = \{A, C\}$

$MM' = \{AA, AC, BA, BC\}$ $\rightarrow$ dyads

Now we want to know what are the non-terminals that produce the dyad

Here all $i \leq j$.

$\therefore$ $MM' \rightarrow$ upper triangular matrix.

$$\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ & M_{22} & M_{23} & M_{24} \\ & & M_{33} & M_{34} \\ & & & M_{44} \end{bmatrix}$$

Now, $M_{14} = ?$

$w_1 w_2 w_3 w_4$ (Regroup)

$= M_{11}M_{24} + M_{12}M_{34} + M_{13}+M_{44}$.

$\therefore$, $M_{14} = [M_{11} \; M_{12} \; M_{13}] \begin{bmatrix} M_{24} \\ M_{34} \\ M_{44} \end{bmatrix}$

Pick the row to the left of the element & col. below the el

eg: String = w = abaab.

Rules:
$S \rightarrow AB$
$A \rightarrow BC \mid a$
$B \rightarrow AC \mid b$
$C \rightarrow a \mid b$

M_{11} M_{12} M_{13} ...
M_{22} M_{23} ...
M_{33}

M_{ii}'s can be trivially computed
ie: $M_{11} = q(a) = \{A, C\}$
$M_{22} = q(b) = \{B, C\}$

$M_{11}\ M_{22} = \{A, C\}\{B, C\}$
$= \{X \mid X \rightarrow YZ \wedge YZ \in \{AB, AC, CB, CC\}\}$

Now apply δ $\Rightarrow \delta(\{\quad\}) = \{S, B\}$

- **Idea:** Model a Machine that accepts a string if the grammar accepts it

PUSHDOWN AUTOMATON (PDA) Stack Machine

Side note: Considering a stack model, at any instant, the state is finite but the state space is infinite.

Machine: Input buffer - tells you how much of the input is read, to be read & reacts accordingly

We have 2 machines here namely
a) Stack b) Input buffer.

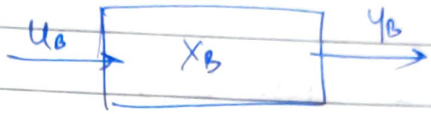
And of course the PDA

All the three pieces are then integrated.

∴ Defining :- $\langle X_B, X_B^\circ, U_B, \xrightarrow{}_B, Y_B, h_B \rangle$

X_B : State space
X_B° : set of initial state
U_B : Input alphabet
$\xrightarrow{}_B \subseteq X_B \times U \times X_B$
Y_B = output alphabet
h_B $X_B \to Y_B$

$X_B = \Sigma^*$



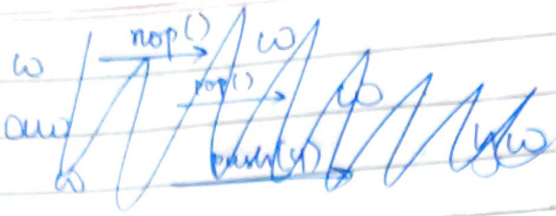
$Y_B = \Sigma \cup \{\cdot\} = \Sigma_\cdot$

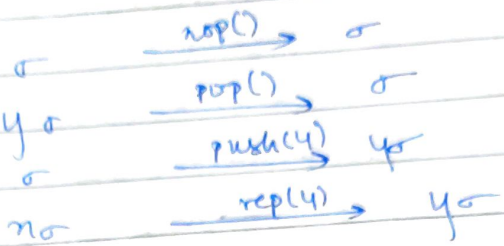
$h(a_i w) = a$
$h(\epsilon)$

$U_B = \{$ next, hold $\}$ → Prompt.
(→ can be for ϵ transitions)

So say $aw \neq \epsilon$
then $aw \xrightarrow{\text{next}} w$. $h(aw) = a$
Also, $aw \xrightarrow{\text{hold}} aw$.

STACK: $S = \langle X_S, X_S^\circ, U_S, \xrightarrow{}_S, Y_S, h_S \rangle$.
stack over Γ
$X_S = \Gamma^*$
$X_S^\circ = \epsilon$
U_S : cmd = { push (y), pop (), nop (), rep (y) } (rep: ↓ replace)
$\Gamma_\perp = Y_S = \Gamma \cup \{\perp\}$
$h_S(y\sigma) = y$ $\sigma \in \Gamma^*$
$h_S(\epsilon) = \perp$





- PUMPING LEMMA FOR CFLs.

If A is CFL ⇒ satisfies pumping lemma.

ie: CFL(A) ⇒

∃ a p ST:- $|S| \geq p$

And:-

~~(i)~~ $S = uvxyz$

(ii) $A \Rightarrow uv^i xy^i z$ ~~for~~ for all $i \geq 0$

(iii) $|vy| > 0$

(iii) $|vxy| \leq p$

Let b be the maximum length of the RHS in the production rules. [max branching factor]

So in the parse trees during the string. If the ht. of the tree is at most h will have $|s| \leq b^h$

ie: at most ht h ⇒ $|s| \leq b^h$

$|s| > b^h$ ⇒ at least ht h

$|s| \geq b^h + 1$ ⇒ at least ht h

Initial parse tree ~~one~~ with smallest no: of nodes

Proof:

(i) Let $|V|$ = no: of variables in the CFG

Choose $p = b^{|V|+1}$

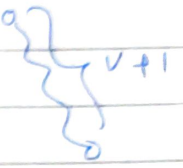
By pumping lemma,

$|S| \geq p$

$|S| \geq b^{|V|+1} \geq b^{|V|} + 1$ for $b > 1$.

$\Rightarrow$ ht. of parse tree $\geq |V| + 1$.

There will be a path of length $|V| + 1$ in the parse tree



$\therefore$,

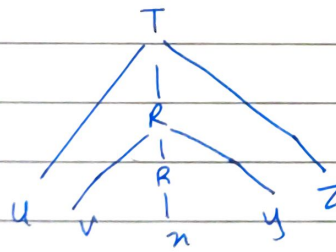
No: of nodes = $|V| + 2$

① $R \to vRy$

② $R \to x$

Eg:- recursively doing $R \to vRy$ will pump up the string else doing ② pumps down

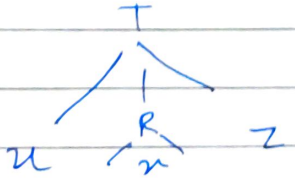
(ii) Let us start by choosing the parse tree with the smallest no: of nodes.



Now let $S = uvxyz$

if $|vy| = 0$

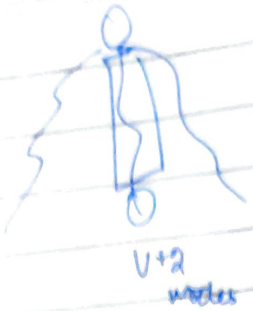
and if we can still pump down the string then we can



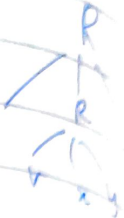
$\#$ Hence contradiction

[By PHP one variable will surely repeat

(iii)



select V+1 nodes.
1 var will repeat.



Max list of (R) is $|V|+1$

$$h \leq |V|+1$$

$$|S| \leq b^{|V|+1}$$

$$\therefore |vxy| \leq b^{|V|+1} = p$$

$$\therefore |vxy| \leq p.$$

Eg (1) Take $A = \{a^n b^n c^n \mid n \geq 0\}$

Prove that A is not CF

$\exists$ u, v, x, y, z. such that

$S = uvxyz$ such that

i) $uv^i xy^i z \in A$

ii) $|vy| > 0$

iii) $|vxy| \leq p$

Let $a^p b^p c^p$ be a string

vxy cannot straddle over a, b, c's $\because |vxy|$

Now, if straddles over abs, pattern breaks

Eg ②: $\{ ww \mid w \in \{0,1\}^* \}$

let $S = 0^p 1^p 0^p 1^p$ ∃ ~~uxv~~ uvxyz

If $vxy \in 0^p 1^p$ ... pump up changes ww condition

~~flow~~

If straddles here:-

$0^p \underbrace{1^p 0^p} 1^p$

straddles

then :- It changes to :- $0^p 1^i 0^i 1^p$.

w w condition fail.