1. Define: Context Free Grammar & Context Free Language.

Context Free Grammar:

A context free grammar is a 4-tuple G=(V,Σ,S,P) where,

V is finite set of non terminals,

Σ is disjoint finite set of terminals,

S is an element of V and it's a start symbol,

P is a finite set formulas of the form A→α where A ∈ V and α∈ (V U Σ)*.

Application of Context Free Grammar(CFG):

1) CFG are extensively used to specify the syntax of programming language.
2) CFG is used to develop a parser.

Context Free Language:

Language generated by CFG is called context free language.

Let G= (V, Σ, S, P) be a CFG. The Language generated by G is

L(G) : {x ∈Σ*/S$\Rightarrow_G$* x}

A language L is a context free Language(CFL) if, there is a CFG G so that L=L(G)

2. Define: Regular Grammar.

A grammar G=(V,Σ,S,P) is regular if every production takes one of the two forms,

B→aC

B→a

Where B and C are Nonterminals and a is terminal.

3. Give Recursive Definitions for following.

I. Recursive Definition of {a,b}*

1. ^∈L.
2. For any S∈L, Sa∈L.
3. For any S∈L, Sb∈L.
4. No other strings are in L.

II. Recursive Definition of Palindrome (pal)

1. ^, a, b ∈ pal
2. For any S ∈ pal , aSa ∈ pal and bSb ∈ pal
3. No other string are in pal

III. The language {$a^n b^n$ / n≥0}

1. ^∈ L
2. For every x ∈ L, axb ∈L
3. No other strings are in L

4. Write CFG for following.

1) Write CFG for ab*

S→aX

X→^| bX

2) Write CFG for a*b*

S→XY
X→aX|^
Y→bY|^

3) **Write CFG for (011+1)*(01)***
S→AB
A→011A | 1A | ^
B→01B | ^

4) **Write CFG which contains at least three times 1.**
S→A1A1A1A
A→0A | 1A | ^

5) **Write CFG that must start and end with same symbol.**
S→0A0 | 1A1
A→0A | 1A | ^

6) **The language of even & odd length palindrome string over {a,b}**
S→aSa|bSb|a|b|^

7) **No. of a and no. of b are same**
S→aSb|bSa|^

8) **Write CFG for regular expression (a+b)*a(a+b)*a(a+b)***
S→XaXaX
X→aX|bX|^

9) **Write CFG for L={$a^ib^jc^k$ | i=j or j=k}**

For i=j	for j=k
S->AB	S->CD
A->aAb | ab	C->aC | a
B->cB | c	D->bDc | bc

10) **Write CFG for L={ $a^ib^jc^k$ | j>i+k}**
S→ABC
A→aAb |^
B→bB | b
C→bCc |^

11) **Write CFG for L={ $0^i1^j0^k$ | j>i+k}**
S→ABC
A→0A1 |^
B→1B | 1
C→1C0 |^

5. Define: Types of Derivation & Ambiguity.

There are mainly two types of derivations.

1. Left most derivation
2. Right most derivation

Let Consider the CFG with the Production S→S+S | S-S | S*S | S/S |(S)| a

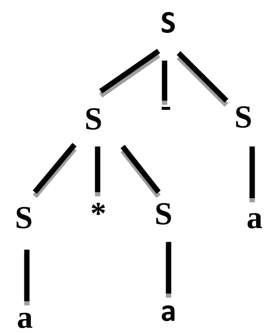
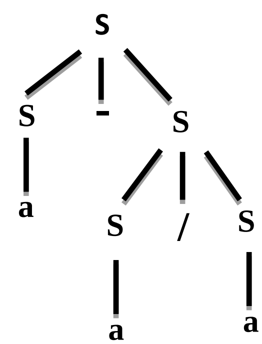
Left Most Derivation	Right Most Derivation
A derivation of a string W in a grammar G is a Left most derivation if at every step the left most nonterminal is replaced	A derivation of a string W in a grammar G is a Right most derivation if at every step the right most nonterminal is replaced
Consider string a*a-a S→S-S S*S-S a*S-S a*a-S a*a-a	Consider string: a-a/a S→S-S S-S/S S-S/a S-a/a a-a/a
Equivalent left most derivation tree 	Equivalent Right most derivation tree 

Table 3.1 Difference between left most & right most derivation

An Ambiguous CFG :

A context free grammar G is ambiguous if there is at least one string in L(G) having two or more distinct derivation tree. (or, equivalently two or more distinct leftmost derivation or rightmost derivation)

1) **Prove that given grammar is ambiguous. S→S+S | S-S | S*S | S/S | (S) |a**

String : a+a+a

S→S+S	S→S+S
a+S	S+S+S
a+S+S	a+S+S
a+a+S	a+a+S
a+a+a	a+a+a

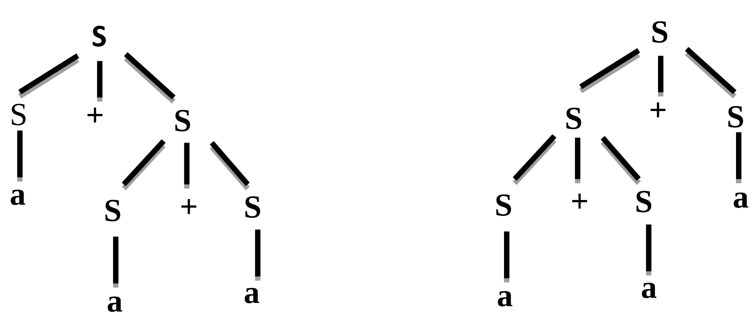


Fig. 3.1 Two left most derivation tree for string a+a+a

- Here, we have two left most derivation for string *a+a+a* hence, above grammar is ambiguous.

2) Prove that S->a | Sa | bSS | SSb | SbS is ambiguous

String: baaab

S→bSS	S→SSb
baS	bSSSb
baSSb	baSSb
baaSb	baaSb
baaab	baaab

- We have two left most derivation for string *baaab* hence, above grammar is ambiguous.

6. Conversion from Finite Automata to Grammar.

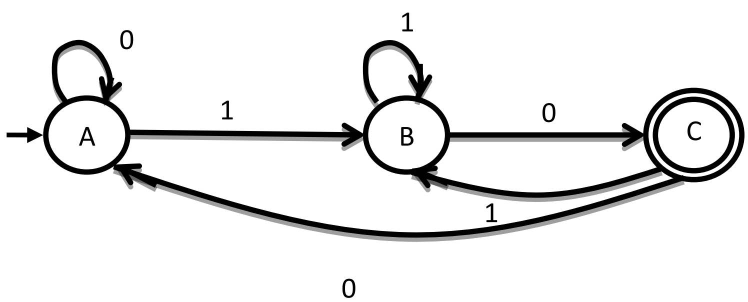


Fig. 3.2 Finite Automata

Equivalent CFG

A→0A
A→1B
B→1B
B→0C
B→0
C→0A
C→1B

7. Backus-Naur Form (BNF)

- BNF is one of the notation techniques for context free grammar.
- It is often used to describe syntax of the language used in computing.
- BNF is shorthand notation for CFG.
- Following grammar uses the notation known as Backus Naur Form. Here, variables written between <..> are non terminals and vertical bar '|' indicating a alternate choice. Apart from the familiar notation =, | and <..>, a new element here is […], which is used to enclosed an optional specification.

Example:

<exp>=<exp> + <term> | <term>
<term>=<term> * <factor> | <factor>

<factor>=<factor> ^ <primary> | <primary>
<primary>=<id> | <const>
<id>=<letter>
<const>=[+/-]<digit>
<letter>=a | b | c |……| z
<digit>=0 | 1 |…………| 9

8. Simplified forms & Normal forms.

Definition: Nullable Variable

A Nullable variable in a CFG G=(V,Σ,S,P) is defined as follows:

1) Any variable A for which P contains A→^ is nullable.
2) if P contains production
   A→B1B2…..Bn where B1B2…Bn are nullable variable, then A is nullable.
3) No other variables in V are nullable.

Eliminate ^ production :

1) **S→aX/Yb**
   X→S/^
   Y→bY/b
   Grammar after elimination of ^ production:
   S→aX/Yb/a
   X→S
   Y→bY/b
2) **S→XaX/bX/Y**
   X→XaX/XbX/^
   Y→ab
   Grammar after elimination of ^ production:
   S→XaX/bX/Y/aX/Xa/a/b
   X→XaX/aX/Xa/a/XbX/Xb/bX/b
   Y→ab

Definition: Unit Production

Unit productions are always in the form of A→B. Where A & B are single non terminals.

Eliminate Unit Production:

1) **S→ABA/BA/AA/AB/A/B**
   A→aA/a
   B→bB/b
   Grammar after elimination of unit production:
   Unit production are S→A and S→B
   S→ABA/BA/AA/AB/aA/a/bB/b
   S→aA/a
   S→bB/b
2) **S→Aa/B**
   A→a/bc/B
   B→A/bb

Grammar after elimination of unit production:
Unit production are S→B,A→B and B→A.
A→a/bc/B
A→a/bc/A/bb
A→a/bc/bb

B→A/bb
B→a/bc/bb

S→Aa/B
S→Aa/a/bc/bb

So CFG after removing unit production is:
S→Aa/a/bc/bb
A→a/bc/bb
B→a/bc/bb

Definition: Chomsky Normal Form

A context free grammar is in Chomsky normal form (CNF) is every production is one of these two forms:

A→BC
A→a

Where A, B, C are nonterminals and a is terminal.

Step to convert CFG into CNF:

1) Eliminate ^-Productions.
2) Eliminate Unit Productions.
3) Restricting the right side of productions to single terminal or string of two or more nonterminals.
   (Replace all mixed string with solid NTs)
4) Final step of CNF. (shorten the string of NT to length 2)

9. Convert following CFG to CNF:
S→aX/Yb
X→S/^
Y→bY/b

Step-1: Eliminate ^-Production:

Nullable production is X→^, new CFG without ^-production is:
S→aX/a/Yb
X→S
Y→bY/b

Step-2: Eliminate Unit Production:

Unit Production is X→S, new CFG without Unit Production is:
S→aX/a/Yb

X→aX/a/Yb
Y→bY/b

Step-3: Replace all mixed string with solid NT:

S→AX/YB/a
X→AX/YB/a
Y→BY/b
A→a
B→b

Step-4 : Shorten the string of NT to length 2

All NT strings on the RHS in the above CFG are already the required length.
So, CFG is in CNF.

10. Convert following CFG to CNF
S→AACD
A→aAb/^
C→aC/a
D→aDa/bDb/^

Step-1: Eliminate ^-Production:

Nullable production is A→^ and D→^, new CFG without ^-production is:
*apply for A →^*
S→AACD/ACD/CD
A→ab/aAb
C→aC/a
D→aDa/bDb/^
*apply for D →^*
S→AACD/ACD/CD/AAC/AC/C
A→ab/aAb
C→aC/a
D→aDa/bDb/aa/bb

Step-2: Eliminate Unit Production:

Unit Production is S→C, new CFG without Unit Production is:
S→AACD/ACD/CD/AAC/AC/aC/a
A→ab/aAb
C→aC/a
D→aDa/bDb/aa/bb

Step-3:Replace all mixed string with solid NT:

S→AACD/ACD/CD/AAC/AC/PC/a
A→PQ/PAQ
C→PC/a
D→PDP/QDQ/PP/QQ
P→a

Q→b

Step-4 : Shorten the string of NT to length 2

S→AT1, T1→AT2 ,T2→CD
S→AU1,U1→CD
S→AV1,V1→AC
S→CD/AC/PC/a
A→PQ
A→PW1, W1→AQ
C→PC/a
D→PP/QQ D→PY1, Y1→DP
D→QZ1, Z1→DQ
P→a
Q→b

11. Convert following CFG to CNF
S→S(S)/^

Step-1: Eliminate ^-Production:

Nullable production is S→^, new CFG without ^-production is:
S→S(S)/(S)/S()/()

Step-2: Eliminate Unit Production:

Here, there is no unit production,
S→S(S)/(S)/S()/()

Step-3:Replace all mixed string with solid NT:

S→SXSY/XSY/SXY/XY
X→(
Y→)

Step-4 : Shorten the string of NT to length 2

S→ST1, T1→XT2, T2→SY
S→XV1 V1→SY
S→SU1 U1→XY
S→XY
X→(
Y→)

12. Unions, Concatenations and Kleen's of Context free language.

Theorem:- If L_1 and L_2 are context - free languages, then the languages L_1 U L_2, L_1L_2 , and L_1^* are also CFLs.

The proof is constructive: Starting with CFGs

$G_1 = (V_1, \Sigma, S_1,P_1)$ and $G_2 = (V_2, \Sigma, S_2,P_2)$,

Generating L_1 and L_2, respectively, we show how to construct a new CFG for each of the three cases.

A grammar $G_u = (V_u, \Sigma, S_u, P_u)$ generating L_1 U L_2. First we rename the element of V_2 if necessary so that $V_1 \cap V_2 = \emptyset$ and we define

$V_u = V_1 \cup V_2 \cup \{S_u\}$

Where S_u is a new symbol not in V_1 or V_2. Then we let

$P_u = P_1 \cup P_2 \cup \{ S_u \rightarrow S_1 \mid S_2 \}$

On the one hand, if x is in either L_1 or L_2, then $S_u \Rightarrow^* x$ in the grammar G_u, because we can start a derivation with either $S_u \rightarrow S_1$ or $S_u \rightarrow S_2$ and continue with the derivation of x in G_1 or G_2. Therefore,

$L_1 \cup L_2 \subseteq L(G_u)$

On the other hand, if x is derivable from S_u in G_u, the first step in any derivation must be

$S_u \Rightarrow S_1$ or $S_u \Rightarrow S_2$

In the first case, all subsequent productions used must be productions in G_1, because no variables in V_2 are involved, and thus $x \in L_1$; in the second case, $x \in L_2$. Therefore,

$L(G_u) \subseteq L_1 \cup L_2$

A grammar $G_c = (V_c, \Sigma, S_c, P_c)$ generating L_1L_2. Again we relabel variables if necessary so that $V_1 \cap V_2 = \emptyset$ and define

$V_c = V_1 \cup V_2 \cup \{S_c\}$

This time we let

$P_c = P_1 \cup P_2 \cup \{ S_c \rightarrow S_1S_2 \}$

If $x \in L_1L_2$ then $x = x_1x_2$, where $x_i \in L_i$ for each i. we may then derive x in G_c as follows:

$S_c \Rightarrow S_1 S_2 \Rightarrow^* x_1 S_2 \Rightarrow^* x_1x_2 = x$

Where the second step is the derivation of x_1 in G_1 and the third step is the derivation of x_2 in G_2. Conversely, if x can be derived from S_c, then since the first step in the derivation must be $S_c \Rightarrow S_1 S_2$, x must be derivable from $S_1 S_2$. Therefore, $x = x_1x_2$, where for each i, x_i can be derived from S_i in G_c. Since $V_1 \cap V_2 = \emptyset$, being derivable from S_i in G_c means being derivable from S_i in G_i, and so $x \in L_1L_2$.

A grammar $G^* = (V, \Sigma, S, P)$ generating L_1^*. Let

$V = V_1 \cup \{S\}$

Where $S \notin V_1$. The language L_1^* contains strings of the form $x = x_1x_2 \ldots x_k$, where each $x_i \in L_1$. Since each x_i can be derived from S_1, then to derive x from S it is enough to be able to derive a string of k S_1'S. We can accomplish this by including the productions

$S \rightarrow S_1S \mid \wedge$

In P. Therefore, let

$P = P_1 \cup \{ S \rightarrow S_1S \mid \wedge \}$

The proof that $L_1^* \subseteq L(G^*)$ is straightforward. If $x \in L(G^*)$, on the other hand, then either $x = \wedge$ or x can be derived from some string of the form S_1^k in G^*. In the second case, since the only production in G^* beginning with S_1 are those in G_1, we may conclude that

$x \in L(G_1)^k \subseteq L(G_1)^*$.