THEORY OF COMPUTATION LECTURE NOTES

UNIT– II **(8 Lectures)**

Regular Expressions and Languages: Regular Expressions: The Operators of regular Expressions, Building Regular Expressions, Precedence of Regular-Expression Operators, Precedence of Regular-Expression Operators

Finite Automata and Regular Expressions: From DFA's to Regular Expressions, Converting DFA's to Regular Expressions, Converting DFA's to Regular Expressions by Eliminating States, Converting Regular Expressions to Automata.

Algebraic Laws for Regular Expressions:

Properties of Regular Languages: The Pumping Lemma for Regular Languages, Applications of the Pumping Lemma Closure Properties of Regular Languages, Decision Properties of Regular Languages, Equivalence and Minimization of Automata,

Context-Free Grammars and Languages: Definition of Context-Free Grammars, Derivations Using a Grammars Leftmost and Rightmost Derivations, The Languages of a Grammar,

Parse Trees: Constructing Parse Trees, The Yield of a Parse Tree, Inference Derivations, and Parse Trees, From Inferences to Trees, From Trees to Derivations, From Derivation to Recursive Inferences,

Applications of Context-Free Grammars: Parsers, Ambiguity in Grammars and Languages: Ambiguous Grammars, Removing Ambiguity From Grammars, Leftmost Derivations as a Way to Express Ambiguity, Inherent Anbiguity

MODULE-II

Regular Expressions: Formal Definition

We construct REs from primitive constituents (basic elements) by repeatedly applying certain recursive rules as given below. (In the definition)

Definition : Let S be an alphabet. The regular expressions are defined recursively as follows.

Basis :

i) ϕ is a RE

ii) $\in$ is a RE

iii) $\forall\ a \in S$, a is RE.

These are called primitive regular expression i.e. Primitive Constituents

Recursive Step :

If r_1 and r_2 are REs over, then so are

i) $r_1 + r_2$

ii) $r_1 r_2$

iii) r_1^*

iv) (r_1)

Closure : *r* is RE over only if it can be obtained from the basis elements (Primitive REs) by a finite no of applications of the recursive step (given in 2).

Example : Let Σ = { 0,1,2 }. Then (0+21)*(1+ F) is a RE, because we can construct this expression by applying the above rules as given in the following step.

Steps	RE Constructed	Rule Used
1	1	Rule 1(iii)
2	ϕ	Rule 1(i)
3	$1+\phi$	Rule 2(i) & Results of Step 1, 2

4	(1+ϕ)	Rule 2(iv) & Step 3
5	2	1(iii)
6	1	1(iii)
7	21	2(ii), 5, 6
8	0	1(iii)
9	0+21	2(i), 7, 8
10	(0+21)	2(iv), 9
11	(0+21)*	2(iii), 10
12	(0+21)*	2(ii), 4, 11

Language described by REs : Each describes a language (or a language is associated with every RE). We will see later that REs are used to attribute regular languages.

Notation : If r is a RE over some alphabet then $L(r)$ is the language associate with r . We can define the language $L(r)$ associated with (or described by) a REs as follows.

1. ϕ is the RE describing the empty language i.e. $L(\phi) = \phi$.

2. $\in$ is a RE describing the language {$\in$} i.e. $L(\in) = \{\in\}$.

3. $\forall a \in S$, a is a RE denoting the language {a} i.e . L(a) = {a} .

4. If r_1 and r_2 are REs denoting language $L(r_1)$ and $L(r_2)$ respectively, then

i) $r_1 + r_2$ is a regular expression denoting the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$

ii) $r_1 r_2$ is a regular expression denoting the language $L(r_1 r_2) = L(r_1)\, L(r_2)$

iii) r_1^* is a regular expression denoting the language $L(r_1^*) = (L(r1))^*$

iv) (r_1) is a regular expression denoting the language $L((r_1)) = L(r_1)$

Example : Consider the RE (0*(0+1)). Thus the language denoted by the RE is

L(0*(0+1)) = L(0*) L(0+1)..... by 4(ii)

= L(0)*L(0) $\cup$ L(1)

= {$\in$, 0,00,000,.....} {0} $\cup$ {1}

= {$\in$, 0,00,000,.....} {0,1}

= {0, 00, 000, 0000,.....,1, 01, 001, 0001,.....}

Precedence Rule

Consider the RE $ab + c$. The language described by the RE can be thought of either $L(a)L(b+c)$ or $L(ab) \cup L(c)$ as provided by the rules (of languages described by REs) given already. But these two represents two different languages lending to ambiguity. To remove this ambiguity we can either

1) Use fully parenthesized expression- (cumbersome) or

2) Use a set of precedence rules to evaluate the options of REs in some order. Like other algebras mod in mathematics.

For REs, the order of precedence for the operators is as follows:

i) The star operator precedes concatenation and concatenation precedes union (+) operator.

ii) It is also important to note that concatenation & union (+) operators are associative and union operation is commutative.

Using these precedence rule, we find that the RE ab+c represents the language $L(ab) \cup L(c)$ i.e. it should be grouped as $((ab)+c)$.

We can, of course change the order of precedence by using parentheses. For example, the language represented by the RE $a(b+c)$ is $L(a)L(b+c)$.

Example : The RE ab^*+b is grouped as $((a(b^*))+b)$ which describes the language $L(a)(L(b))^* \cup L(b)$

Example : The RE $(ab)^*+b$ represents the language $(L(a)L(b))^* \cup L(b)$.

Example : It is easy to see that the RE (0+1)*(0+11) represents the language of all strings over {0,1} which are either ended with 0 or 11.

Example : The regular expression *r* =(00)*(11)*1 denotes the set of all strings with an even number of 0's followed by an odd number of 1's i.e. $L(r) = \{0^{2n}1^{2m+1} \mid n \geq 0,\ m \geq 0\}$

Note : The notation r^+ is used to represent the RE *rr**. Similarly, r^2 represents the RE *rr*, r^2 denotes $r^2 r$, and so on.

An arbitrary string over Σ = {0,1} is denoted as (0+1)*.

Exercise : Give a RE *r* over {0,1} s.t. $L(r)$={$\omega \in \Sigma^* \mid \omega$ has at least one pair of consecutive 1's}

Solution : Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes before is completely arbitrary. Considering these observations we can write the REs as (0+1)*11(0+1)*.

Example : Considering the above example it becomes clean that the RE (0+1)*11(0+1)*+(0+1)*00(0+1)* represents the set of string over {0,1} that contains the substring 11 or 00.

Example : Consider the RE 0*10*10*. It is not difficult to see that this RE describes the set of strings over {0,1} that contains exactly two 1's. The presence of two 1's in the RE and any no of 0's before, between and after the 1's ensure it.

Example : Consider the language of strings over {0,1} containing two or more 1's.

Solution : There must be at least two 1's in the RE somewhere and what comes before, between, and after is completely arbitrary. Hence we can write the RE as (0+1)*1(0+1)*1(0+1)*. But following two REs also represent the same language, each ensuring presence of least two 1's somewhere in the string

i) 0*10*1(0+1)*

ii) (0+1)*10*10*

Example : Consider a RE r over {0,1} such that

$L(r)$ = { $w \in \{0,1\}^* \mid w$ has no pair of consecutive 1's}

Solution : Though it looks similar to ex ……., it is harder to construct to construct. We observer that, whenever a 1 occurs, it must be immediately followed by a 0. This substring may be preceded & followed by any no of 0's. So the final RE must be a repetition of strings of the form: 00…0100….00 i.e. 0*100*. So it looks like the RE is (0*100*)*. But in this case the strings ending in 1 or consisting of all 0's are not accounted for. Taking these observations into consideration, the final RE is r = (0*100*)(1+ $\in$)+0*(1+ $\in$).

Alternative Solution :
The language can be viewed as repetitions of the strings 0 and 01. Hence get the RE as r = (0+10)*(1+ $\in$). This is a shorter expression but represents the same language.

Regular Expression:

FA to regular expressions:

FA to RE (REs for Regular Languages) :

Lemma : If a language is regular, then there is a RE to describe it. i.e. if $L = L(M)$ for some DFA M, then there is a RE r such that $L = L(r)$.

Proof : We need to construct a RE r such that $L(r) = \{w \mid w \in L(M)\}$. Since M is a DFA, it has a finite no of states. Let the set of states of M is Q = {1, 2, 3,..., n} for some integer n. [Note : if the n states of M were denoted by some other symbols, we can always rename those to indicate as 1, 2, 3,..., n]. The required RE is constructed inductively.

Notations : $r_{ij}^{(k)}$ is a RE denoting the language which is the set of all strings w such that w is the label of a path from state i to state j $(1 \le i, j \le n)$ in M, and that path has no intermediate state whose number is greater then k. (i & j (begining and end pts) are not considered to be "intermediate" so i and /or j can be

greater than k)

We now construct $r_{ij}^{(k)}$ inductively, for all $i, j \in Q$ starting at $k = 0$ and finally reaching $k = n$.

Basis : $k = 0$, $r_{ij}^{(0)}$ i.e. the paths must not have any intermediate state (since all states are numbered 1 or above). There are only two possible paths meeting the above condition **:**

1. A direct transition from state i to state j.
   - $r_{ij}^{(0)}$ = a if then is a transition from state i to state j on symbol the single symbol a.
   - $r_{ij}^{(0)} = a_1 + a_2 + \cdots + a_m$ if there are multiple transitions from state i to state j on symbols $a_1, a_2, \cdots, a_m$.
   - $r_{ij}^{(0)}$ = f if there is no transition at all from state i to state j.
2. All paths consisting of only one node i.e. when $i = j$. This gives the path of length 0 (i.e. the RE $\in$ denoting the string $\in$) and all self loops. By simply adding Î to various cases above we get the corresponding REs i.e.
   - $r_{ii}^{(0)} = \in + a$ if there is a self loop on symbol a in state i .
   - $r_{ii}^{(0)} = \in + a_1 + a_2 + \cdots + a_m$ if there are self loops in state i as multiple symbols $a_1, a_2, \cdots, a_m$.
   - $r_{ii}^{(0)} = \in$ if there is no self loop on state i.

Induction :

Assume that there exists a path from state i to state j such that there is no intermediate state whose number is greater than k. The corresponding Re for the label of the path is $r_{ij}^{(k)}$.
There are only two possible cases **:**

1. The path dose not go through the state k at all i.e. number of all the intermediate states are less than k. So, the label of the path from state i to state j is tha language described by the RE $r_{ij}^{(k-1)}$.
2. The path goes through the state k at least once. The path may go from i to j and k may appear more than once. We can break the into pieces as shown in the figure 7.

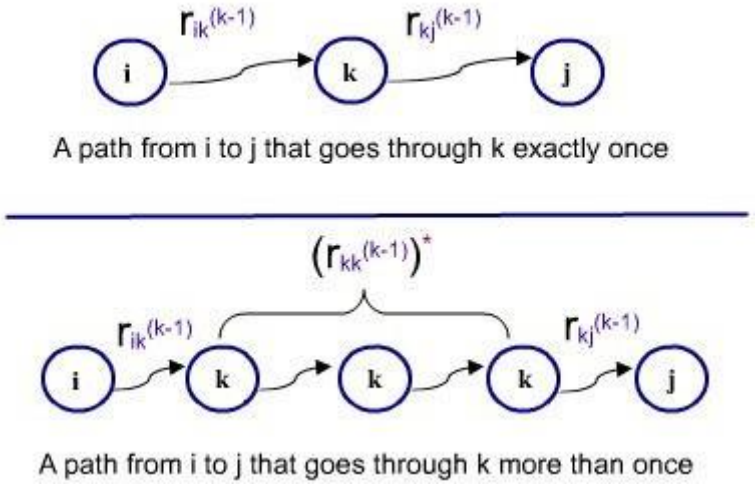

Figure 7

1. The first part from the state i to the state k which is the first recurence. In this path, all intermediate states are less than k and it starts at iand ends at k. So the RE $r_{ik}^{(k-1)}$ denotes the language of the label of path.
2. The last part from the last occurence of the state k in the path to state j. In this path also, no intermediate state is numbered greater than k. Hence the RE $r_{kj}^{(k-1)}$ denoting the language of the label of the path.
3. In the middle, for the first occurence of k to the last occurence of k , represents a loop which may be taken zero times, once or any no of times. And all states between two consecutive k's are numbered less than k.

Hence the label of the path of the part is denoted by the RE $\left(r_{ij}^{(k-1)}\right)^*$ .The label of the path from state i to state j is the concatenation of these 3 parts which is

$$r_{ik}^{(k-1)}\left(r_{kk}^{(k-1)}\right)^* r_{kj}^{(k-1)}$$

Since either case 1 or case 2 may happen the labels of all paths from state i to j is denoted by the following RE

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} + r_{ik}^{(k-1)}\left(r_{kk}^{(k-1)}\right)^* r_{kj}^{(k-1)}$$

We can construct $r_{ij}^{(k)}$ for all $i, j \in \{1,2,..., n\}$ in increasing order of k starting with the basis $k = 0$ upto $k = n$ since $r_{ij}^{(k)}$ depends only on expressions with a small superscript (and hence will be available). WLOG, assume that state 1 is the start state and $j_1, j_2, \cdots, j_m$ are the m final states where $ji \in \{1, 2, ... , n\}$, $1 \leq i \leq m$ and $m \leq n$. According to the convention used, the language of the automatacan be denoted by the RE

$r_{1j_1}^{(n)} + r_{1j_2}^{(n)} + \cdots + r_{1j_m}^{(n)}$

Since $r_{1j_i}^{(n)}$ is the set of all strings that starts at start state 1 and finishes at final state j_i following the transition of the FA with any value of the intermediate state (1, 2, ... , *n*) and hence accepted by the automata.

Regular Grammar:

A *grammar* $G = (N, \Sigma, P, S)$ is right-linear if each production has one of the following three forms:

- $A \rightarrow cB$,
- $A \rightarrow c$,
- $A \rightarrow \in$

Where $A, B \in N'$ (with $A = B$ allowed) and $c \in \Sigma$. A *grammar* G is left-linear if each production has once of the following three forms.

$A \rightarrow Bc$, $A \rightarrow c$, $A \rightarrow \in$

A right or left-linear grammar is called a regular grammar.

Regular grammar and Finite Automata are equivalent as stated in the following theorem.

Theorem : A language L is regular iff it has a regular grammar. We use the following two lemmas to prove the above theorem.

Lemma 1 : If L is a regular language, then L is generated by some right-linear grammar.

Proof : Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts L.

Let $Q = \{q_0, q_1, \cdots, q_n\}$ and $\Sigma = \{a_1, a_2, \cdots, a_m\}$.

We construct the right-linear grammar $G = (N, \Sigma, P, S)$ by letting

$N = Q$, $S = q_0$ and $P = \{A \rightarrow cB \mid \delta(A, c) = B\} \cup \{A \rightarrow c \mid \delta(A, c) \in B\}$

[Note: If $B \in F$, then $B \rightarrow \in P$]

Let $w = a_1 a_2 \ldots a_k \in L(M)$. For M to accept *w*, there must be a sequence of states $q_0, q_1, \ldots, q_k$ such that

$\delta(q_0, a_1) = q_1$

$\delta(q_1, a_2) = q_2$

$\cdots$

$\delta(q_{k-1}, a_k) = q_k$

and $q_k \in F$

By construction, the grammar G will have one production for each of the above transitions. Therefore, we have the corresponding derivation.

$$S = q_0 \underset{G}{\Rightarrow} a_1 q_1 \underset{G}{\Rightarrow} a_1 a_2 q_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} a_1 a_2 \cdots a_k q_k \underset{G}{\Rightarrow} a_1 a_2 \cdots a_k = w$$

Hence $w \in L(g)$.

Conversely, if $w = a_1 a_2 \ldots a_k \in L(G)$, then the derivation of w in G must have the form as given above. But, then the construction of G from M implies that

$\delta(q_0, a_1 a_2 \cdots a_k) = q_k$, where $q_k \in F$, completing the proof.

Lemma 2 : Let $G = (N, \Sigma, P, S)$ be a right-linear grammar. Then $L(G)$ is a regular language.

Proof: To prove it, we construct a FA M from G to accept the same language.

$M = (Q, \Sigma, \delta, q_0, F)$ is constructed as follows:

$Q = N \cup \{q_f\}$ (q_f is a special sumbol not in N)

$q_0 = S$, $F = \{q_f\}$

For any $q \in N$ and $a \in \Sigma$ and δ is defined as

$\delta(q, a) = \{p \mid q \to ap \in P\}$ if $q \to a \notin P$

and $\delta(q, a) = \{p \mid q \to ap \in P\} \cup \{q_f\}$, if $q \to a \in P$.

We now show that this construction works.

Let $w = a_1 a_2 \ldots a_k \in L(G)$. Then there is a derivation of w in G of the form

$$S \underset{G}{\Rightarrow} a_1 q_1 \underset{G}{\Rightarrow} a_1 a_2 q_2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} a_1 a_2 \cdots a_{k-1} a_k \,(= w)$$

By contradiction of M, there must be a sequence of transitions

$$\delta(q_0, a_1) = q_1$$
$$\delta(q_1, a_2) = q_2$$
$$\cdots$$
$$\delta(q_{k-1}, a_k) = q_f$$

implying that $w = a_1 a_2 \ldots a_k \in L(M)$ i.e. *w* is accepted by M.

Conversely, if $w = a_1 a_2 \cdots a_k$ is accepted by M, then because q_f is the only accepting state of M, the transitions causing *w* to be accepted by M will be of the form given above. These transitions corresponds to a derivationof *w* in the grammar G. Hence $w \in L(G)$, completing the proof of the lemma.

Given any left-linear grammar G with production of the form $A \to cB \mid c \mid \in$, we can construct from it a right-linear grammar $\hat{G}$ by replacing every production of G of the form $A \to cB$ with $A \to Bc$

It is easy to prove that $L(G) = \left(L(\hat{G})\right)^R$. Since $\hat{G}$ is right-linear, $L(\hat{G})$ is regular. But then so are $\left(L(\hat{G})\right)^R$ i.e. $L(G)$ because regular languages are closed under reversal.

Putting the two lemmas and the discussions in the above paragraph together we get the proof of the theorem-

A language L is regular iff it has a regular grammar

Example : Consider the grammar

$$G: S \to 0A \mid 0$$
$$A \to 1S$$

It is easy to see that G generates the language denoted by the regular expression (01)*0.
The construction of lemma 2 for this grammar produces the follwoing FA.
This FA accepts exactly (01)*1.

Decisions Algorithms for CFL

In this section, we examine some questions about CFLs we can answer. A CFL may be represented using a CFG or PDA. But an algorithm that uses one representation can be made to work for the others, since we can construct one from the other.

Testing Emptiness :

Theorem : There are algorithms to test emptiness of a CFL.

Proof : Given any CFL L, there is a CFG G to generate it. We can determine, using the construction described in the context of elimination of useless symbols, whether the start symbol is useless. If so, then $L(G) = \phi$; otherwise not.

Testing Membership :

Given a CFL L and a string x, the membership, problem is to determine whether $x \in L$?

Given a PDA P for L, simulating the PDA on input string x doesnot quite work, because the PDA can grow its stack indefinitely on $\in$ input, and the process may never terminate, even if the PDA is deterministic.

So, we assume that a CFG $G = (N, \Sigma, P, S)$ is given such that $L = L(G)$.

Let us first present a simple but inefficient algorithm.

Convert G to $G' = (N', \Sigma', P', S')$ in CNF generating $L(G) - \{\in\}$. If the input string $x = \in$, then we need to determine whether $S \overset{*}{\underset{G}{\Rightarrow}} \in$ and it can easily be done using the technique given in the context of elimination of $\in$-production. If , $x \neq \in$ then $x \in L(G')$ iff $x \in L(G)$. Consider a derivation under a grammar in CNF. At every step, a production in CNF in used, and hence it adds exactly one terminal symbol to the sentential form. Hence, if the length of the input string x is n, then it takes exactly n steps to derive x (provided x is in $L(G')$).

Let the maximum number of productions for any nonterminal in G' is K. So at every step in derivation, there are atmost k choices. We may try out all these choices, systematically., to derive the string x in G'. Since there are atmost $K^{|x|}$ i.e. K^n choices. This algorithms is of exponential time complexity. We now present an efficient (polynomial time) membership algorithm.

Pumping Lemma:

Limitations of Finite Automata and Non regular Languages :

The class of languages recognized by FA s is strictly the regular set. There are certain languages which are non regular i.e. cannot be recognized by any FA

Consider the language $L = \{a^n b^n \mid n \geq 0\}$

In order to accept is language, we find that, an automaton seems to need to remember when passing the center point between a's and b's how many a's it has seen so far. Because it would have to compare that with the number of b's to either accept (when the two numbers are same) or reject (when they are not same) the input string.

But the number of *a*'s is not limited and may be much larger than the number of states since the string may be arbitrarily long. So, the amount of information the automaton need to remember is unbounded.

A finite automaton cannot remember this with only finite memory (i.e. finite number of states). The fact that FA s have finite memory imposes some limitations on the structure of the languages recognized. Inductively, we can say that a language is regular only if in processing any string in this language, the information that has to be remembered at any point is strictly limited. The argument given above to show that $a^n b^n$ is non regular is informal. We now present a formal method for showing that certain languages such as $a^n b^n$ are non regular

Properties of CFL's

Closure properties of CFL:

We consider some important closure properties of CFLs.

Theorem : If L_1 and L_2 are CFLs then so is $L_1 \cup L_2$

Proof : Let $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and $G_2 = (N_2, \Sigma_2, P_2, S_2)$ be CFGs generating. Without loss of generality, we can assume that $N_1 \cap N_2 = \phi$. Let S_3 is a nonterminal not in N_1 or N_1. We construct the grammar $G_3 = (N_3, \Sigma_3, P_3, S_3)$ from G_1 and G_2, where

$N_3 = N_1 \cup N_2 \cup \{S_3\}$,

$\Sigma_3 = \Sigma_1 \cup \Sigma_2$

$P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1 \mid S_2\}$

We now show that $L(G_3) = L(G_1) \cup L(G_2) = L_1 \cup L_2$

Thus proving the theorem.

Let $w \in L_1$. Then $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$. All productions applied in their derivation are also in G_2. Hence $S_3 \overset{1}{\underset{G_3}{\Rightarrow}} S_1 \overset{*}{\underset{G_3}{\Rightarrow}} w$ i.e. $w \in L(G_3)$

Similarly, if $w \in L_2$, then $w \in L(G_3)$

Thus $L_1 \cup L_2 \subseteq L(G_3)$.

Conversely, let $w \in L(G_3)$. Then $S_3 \overset{*}{\underset{G_3}{\Rightarrow}} w$ and the first step in this derivation must be either $S_3 \overset{1}{\underset{G_3}{\Rightarrow}} S_1$ or $S_3 \overset{1}{\underset{G_3}{\Rightarrow}} S_2$. Considering the former case, we have $S_3 \overset{1}{\underset{G_3}{\Rightarrow}} S_1 \overset{*}{\underset{G_3}{\Rightarrow}} w$

Since N_1 and N_1 are disjoint, the derivation $S_1 \overset{*}{\underset{G_3}{\Rightarrow}} w$ must use the productions of P_1 only (which are also in P_3) Since $S_1 \in N_1$ is the start symbol of G_1. Hence, $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} w$ giving $w \in L(G_1)$.

Using similar reasoning, in the latter case, we get $w \in L(G_2)$. Thus $L(G_3) \subseteq L_1 \bigcup L_2$.

So, $L(G_3) = L_1 \bigcup L_2$, as claimed

Theorem : If L_1 and L_2 are CFLs, then so is $L_1 L_2$.

Proof : Let $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and $G_2 = (N_2, \Sigma_2, P_2, S_2)$ be the CFGs generating L_1 and L_2 respectively. Again, we assume that N_1 and N_1 are disjoint, and S_3 is a nonterminal not in N_1 or N_1. we construct the CFG $G_3 = (N_3, \Sigma_3, P_3, S_3)$ from G_1 and G_2, where

$$N_3 = N_1 \bigcup N_2 \bigcup \{S_3\}$$

$$\Sigma_3 = \Sigma_1 \bigcup \Sigma_2$$

$$P_3 = P_1 \bigcup P_2 \bigcup \{S_3 \rightarrow S_1 S_2\}$$

We claim that $L(G_3) = L(G_1) L(G_2) = L_1 L_2$

To prove it, we first assume that $x \in L_1$ and $y \in L_2$. Then $S_1 \overset{*}{\underset{G_1}{\Rightarrow}} x$ and $S_2 \overset{*}{\underset{G_1}{\Rightarrow}} y$. We can derive the string *xy* in G_2 as shown below.

$$S_3 \overset{1}{\underset{G_3}{\Rightarrow}} S_1 S_2 \overset{*}{\underset{G_3}{\Rightarrow}} x S_2 \overset{*}{\underset{G_3}{\Rightarrow}} xy$$

since $P_1 \subseteq P$ and $P_2 \subseteq P$. Hence $L_1 L_2 \subseteq L(G_3)$.

For the converse, let $w \in L(G_3)$. Then the derivation of *w* in G_2 will be of the form

$S_3 \underset{G_3}{\overset{1}{\Rightarrow}} S_1 S_2 \underset{G_3}{\overset{*}{\Rightarrow}} w$ i.e. the first step in the derivation must see the rule $S_3 \to S_1 S_2$. Again, since N_1 and N_1 are disjoint and $S_1 \in N_1$ and $S_2 \in N_2$, some string *x* will be generated from S_1 using productions in P_1 (which are also in P_3) and such that $xy = w$.

Thus $S_3 \Rightarrow S_1 S_2 \Rightarrow xS_2 \Rightarrow xy = w$

Hence $S_1 \underset{G_1}{\overset{*}{\Rightarrow}} x$ and $S_2 \underset{G_2}{\overset{*}{\Rightarrow}} y$.

This means that *w* can be divided into two parts *x, y* such that $x \in L_1$ and $y \in L_2$. Thus $w \in L_1 L_2$ .This completes the proof

Theorem : If L is a CFL, then so is L^*.

Proof : Let $G = (N, \Sigma, P, S)$ be the CFG generating L. Let us construct the CFG $G' = (N, \Sigma, P', S)$ from G where $P' = P \cup \{S \to SS \mid \in\}$.

We now prove that $L(G') = (L(G))^* = L^*$, which prove the theorem.

G' can generate $\in$ in one step by using the production $S \to \in$ since $P \subseteq P'$, G' can generate any string in L.

Let $w \in L^n$ for any $n > 1$ we can write $w = w_1 w_2 \cdots w_n$ where $w_i \in L$ for $1 \le i \le n$. *w* can be generated by G' using following steps.

$$S \underset{G}{\overset{n-1}{\Rightarrow}} SS \cdots S \underset{G}{\overset{*}{\Rightarrow}} w_1 Ss \cdots s \underset{G}{\overset{*}{\Rightarrow}} w_1 w_2 ss \cdots s \underset{G}{\overset{*}{\Rightarrow}} w_1 w_2 \cdots w_n = w$$

First $(n\text{-}1)$-steps uses the production $S \to SS$ producing the sentential form of n numbers of S 's. The nonterminal S in the i-th position then generates w_i using production in P (which are also in P')

It is also easy to see that G can generate the empty string, any string in L and any string $w \in L^n$ for $n > 1$ and none other.

Hence $L(G') = (L(G))^* = L^*$

Theorem : CFLs are not closed under intersection

Proof : We prove it by giving a counter example. Consider the language $L_1 = \{a^i b^i c^j \mid i, j \ge 0\}$ .The following CFG generates L_1 and hence a CFL

$S \rightarrow XC$

$X \rightarrow aXb \mid \in$

$C \rightarrow cC \mid \in$

The nonterminal X generates strings of the form $a^n b^n, n \geq 0$ and C generates strings of the form c^m, $m \geq 0$. These are the only types of strings generated by X and C. Hence, S generates L_1.

Using similar reasoning, it can be shown that the following grammar $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$ and hence it is also a CFL.

$S \rightarrow AX$

$A \rightarrow aA \mid \in$

$X \rightarrow bXc \mid \in$

But, $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ and is already shown to be not context-free.

Hence proof.

Theorem : A CFL's are not closed under complementations

Proof : Assume, for contradiction, that CFL's are closed under complementation. SInce, CFL's are also closed under union, the language $\overline{\overline{L_1} \cup \overline{L_2}}$, where L_1 and L_2 are CFL's must be CFL. But by DeMorgan's law

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$$

This contradicts the already proved fact that CFL's are not closed under intersection.

But it can be shown that the CFL's are closed under intersection with a regular set.

Theorem : If L is a CFL and R is a regular language, then $L \cap R$ is a CFL.

Proof : Let $P = (Q_p, \Sigma, \Gamma, \delta_p, q_p, z_0, F_p)$ be a PDA for L and let $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ be a DFA for R.

We construct a PDA M from P and D as follows

$$M = (Q_p \times Q_D, \Sigma, \Gamma, \delta_M, (q_p, q_D), z_0, F_p \times F_D)$$

where δ_M is defined as

$\delta_M((p,q), a, X)$ contains $((r,s), \alpha)$ iff

$\delta_D(q,a) = s$ and $\delta_P(p,a,X)$ contains (r,α)

The idea is that M simulates the moves of P and D parallely on input *w*, and accepts *w* iff both P and D accepts. That means, we want to show that

$$L(M) = L(P) \cap L(D) = L \cap R$$

We apply induction on *n*, the number of moves, to show that

$\left((q_P, q_D), w, z_0\right) \overset{n}{\underset{M}{\mapsto}} \left((p,q), \in, \gamma\right)$ iff

$\left(q_P, w, z_0\right) \overset{n}{\underset{P}{\mapsto}} (p, \in, \gamma)$ and $\hat{\delta}(q_D, w) = q$

Basic Case is *n*=0. Hence $p = q_P$, $q = q_D$, $\gamma = z_0$ and $w = \in$. For this case it is trivially true

Inductive hypothesis : Assume that the statement is true for *n* -1.

Inductive Step : Let *w* = *xa* and

Let $\left((q_P, q_D), x_a, z_0\right) \overset{n-1}{\underset{M}{\mapsto}} \left((p',q'), a, \alpha\right) \overset{1}{\underset{M}{\mapsto}} \left((p,q), \in, \gamma\right)$

By inductive hypothesis, $\left(q_P, x, z_0\right) \overset{n-1}{\underset{P}{\mapsto}} (p', \in, \alpha)$ and $\hat{\delta}_D(q_D, x) = q'$

From the definition of δ_M and considering the *n*-th move of the PDA M above, we have

$\delta_P(p', a, \alpha) = (p, \in, \gamma)$ and $\delta_D(q', a) = q$

Hence $\left(q_P, xa, z_0\right) \overset{n-1}{\underset{P}{\mapsto}} (p', a, \alpha) \overset{1}{\underset{P}{\mapsto}} (p, \in, \gamma)$ and $\hat{\delta}_D(q_D, w) = q$

If $p \in F_P$ and $q \in F_D$, then $p, q \in F_P \times F_D$ and we got that if M accepts *w*, then both P and D accepts it.
We can show that converse, in a similar way. Hence $L \cap R$ is a CFL (since it is accepted by a PDA M)
This property is useful in showing that certain languages are not context-free.

Example : Consider the language

$$L = \left\{ w \in \{a,b,c\}^* \mid w \text{ contains equal number of } a\text{'s}, b\text{'s and } c\text{'s} \right\}$$

Intersecting L with the regular set $R = a^*b^*c^*$, we get

$$L \cap R = L \cap a^* b^* c^*$$
$$= \left\{ a^n b^n c^n \mid n \geq 0 \right\}$$

Which is already known to be not context-free. Hence L is not context-free

Theorem : CFL's are closed under reversal. That is if L is a CFL, then so is L^R

Proof : Let the CFG $G = (N, \Sigma, P, S)$ generates L. We construct a CFG $G' = (N, \Sigma, P', S)$ where $P = \left\{ A \to \alpha \mid A \to \alpha^R \in P \right\}$. We now show that $L(G') = L^R$, thus proving the theorem.

We need to prove that

$A \underset{G}{\overset{n}{\Rightarrow}} \alpha$ iff $A \underset{G'}{\overset{n}{\Rightarrow}} \alpha^R$.

The proof is by induction on *n*, the number of steps taken by the derivation. We assume, for simplicity (and of course without loss of generality), that G and hence G' are in CNF.

The basis is *n*=1 in which case it is trivial. Because α must be either $a \in \Sigma$ or BC with $B, C \in N$.

Hence $A \underset{G}{\overset{1}{\Rightarrow}} a$ iff $A \underset{G'}{\overset{1}{\Rightarrow}} a$

Assume that it is true for (n-1)-steps. Let $A \underset{G}{\overset{n}{\Rightarrow}} \alpha$. Then the first step must apply a rule of the form $A \to BC$ and it gives

$A \underset{G}{\overset{1}{\Rightarrow}} BC \underset{G}{\overset{n-1}{\Rightarrow}} \beta\gamma = \alpha$ where $B \underset{G}{\overset{*}{\Rightarrow}} \beta^R$ and $C \underset{G}{\overset{*}{\Rightarrow}} \gamma^R$

By constructing of G', $A \to CB \in P'$

Hence

$A \underset{G'}{\overset{1}{\Rightarrow}} CB \underset{G'}{\overset{n-1}{\Rightarrow}} \gamma^R \beta^R = \alpha^R$

The converse case is exactly similar

Substitution :

$\forall a \in \Sigma$, let L_a be a language (over any alphabet). This defines a function S, called substitution, on Σ which is denoted as $s(a) = L_a$ - for all $a \in \Sigma$

This definition of substitution can be extended further to apply strings and langauge as well.

If $w = a_1 a_2 \cdots a_n$, where $a_i \in \Sigma$, is a string in Σ^*, then

$s(w) = s(a_1 a_2 \cdots a_n) = s(a_1) s(a_2) \cdots s(a_n)$.

Similarly, for any language L,

$s(L) = \left\{ s(w) \mid w \in L \right\}$

The following theorem shows that CFLs are closed under substitution.

Thereom : Let $L \subseteq \Sigma^*$ is a CFL, and s is a substitution on Σ such that $s(a) = L_a$ is a CFL for all $a \in \Sigma$, thus $s(L)$ is a CFL

Proof : Let $L = L(G)$ for a CFG $G = (N, \Sigma, P, S)$ and for every $a \in \Sigma$, $L_a = L(G_a)$ for some $G_a = (N_a, \Sigma_a, P_a, S_a)$. Without loss of generality, assume that the sets of nonterminals N and N_a's are disjoint.

Now, we construct a grammar G', generating $s(L)$, from G and G_a's as follows :

- $G' = (N', \Sigma', P', S)$
- $N' = N \cup \bigcup_{a_i \in \Sigma} N_{a_i}$
- $\Sigma' = \bigcup_{a_i \in \Sigma} \Sigma_{a_i}$
- P' consists of
  1. $\bigcup_{a_i \in \Sigma} P_{a_i}$ and
  2. The production of P but with each terminal a in the right hand side of a production replaced by S_a everywhere.

We now want to prove that this construction works i.e. $w \in L(G')$ iff $w \in s(L)$.

If Part : Let $w \in s(L)$ then according to the definition there is some string $x = a_1 a_2 \cdots a_n \in L$ and $x_i \in S(a_i)$ for $i = 1, 2, \cdots, n$ such that $w = x_1 x_2 \cdots x_n \left(= s(a_1) s(a_2) \cdots s(a_n)\right)$

We will show that $S \overset{*}{\underset{G'}{\Rightarrow}} w$.

From the construction of G', we find that, there is a derivation $S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_1} S_{a_2} \cdots S_{a_n}$ corresponding to the string $x = a_1 a_2 \cdots a_n$ (since G' contains all productions of G but every ai replaced with S_{a_i} in the RHS of any production).

Every S_{a_i} is the start symbol of G_{a_i} and all productions of G_{a_i} are also included in G'.
Hence

$$S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_1} S_{a_2} \cdots S_{a_n}$$
$$\overset{*}{\underset{G'}{\Rightarrow}} x_1 S_{a_2} \cdots S_{a_n}$$
$$\overset{*}{\Rightarrow} x_1 x_2 \cdots x_n = w$$

Therefore, $w \in L(G')$

(Only-if Part) Let $w \in L(G')$. Then there must be a derivative as follows :

$S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_1} S_{a_2} \cdots S_{a_n}$ (using the production of G include in G' as modified by (step 2) of the construction of P'.)

Each S_{a_i} ($i = 1, 2, \cdots, n$) can only generate a string $x_i \in L_{a_i}$, since each N_{a_i}'s and N are disjoin. Therefore, we get

$$S \overset{*}{\underset{G'}{\Rightarrow}} S_{a_1} S_{a_2} \cdots S_{a_n}$$
$$\overset{*}{\underset{G'}{\Rightarrow}} x_1 S_{a_2} \cdots S_{a_n} \text{ since } S_{a_1} \overset{*}{\underset{G_{a_1}}{\Rightarrow}} x_1$$

$\overset{*}{\underset{G}{\Rightarrow}} x_1 x_2 S_{a_3} \cdots S_{a_n}$ since $S_{a_2} \overset{*}{\underset{G_{a_2}}{\Rightarrow}} x_2$

$\overset{*}{\underset{G}{\Rightarrow}} x_1 x_2 \cdots x_n$

$= w$

The string $w = x_1 x_2 \cdots x_n$ is formed by substituting strings x_i for each a_i's and hence $w \in s(L)$.

Theorem : CFL's are closed under homomorphism

Proof : Let $L \subseteq \Sigma^*$ be a CFL, and *h* is a homomorphism on Σ i.e $h : \Sigma \to \Delta^*$ for some alphabets Δ. consider the following substitution S:Replace each symbol $a \in \Sigma$ by the language consisting of the only string h(a), i.e. $s(a) = \{h(a)\}$ for all $a \in \Sigma$. Then, it is clear that, $h(L) = s(L)$. Hence, CFL's being closed under substitution must also be closed under homomorphism.

Grammar

A grammar is a mechanism used for describing languages. This is one of the most simple but yet powerful mechanism. There are other notions to do the same, of course.

In everyday language, like English, we have a set of symbols (alphabet), a set of words constructed from these symbols, and a set of rules using which we can group the words to construct meaningful sentences. The grammar for English tells us what are the words in it and the rules to construct sentences. It also tells us whether a particular sentence is well-formed (as per the grammar) or not. But even if one follows the rules of the english grammar it may lead to some sentences which are not meaningful at all, because of impreciseness and ambiguities involved in the language. In english grammar we use many other higher level constructs like noun-phrase, verb-phrase, article, noun, predicate, verb etc. A typical rule can be defined as

$$< \text{sentence} > \rightarrow < \text{noun-phrase} > < \text{predicate} >$$

meaning that "a sentence can be constructed using a 'noun-phrase' followed by a predicate".

Some more rules are as follows:

$$< \text{noun-phrase} > \rightarrow < \text{article} >< \text{noun} >$$

$$< \text{predicate} > \rightarrow < \text{verb} >$$

with similar kind of interpretation given above.

If we take {a, an, the} to be <article>; cow, bird, boy, Ram, pen to be examples of <noun>; and eats, runs, swims, walks, are associated with <verb>, then we can construct the sentence- a cow runs, the boy eats, an pen walks- using the above rules. Even though all sentences are well-formed, the last one is not meaningful. We observe that we start with the higher level construct <sentence> and then reduce it to <noun-phrase>, <article>, <noun>, <verb> successively, eventually leading to a group of words associated with these constructs.

These concepts are generalized in formal language leading to formal grammars. The word 'formal' here refers to the fact that the specified rules for the language are explicitly stated in terms of what strings or symbols can occur. There can be no ambiguity in it.

Formal definitions of a Grammar

A grammar G is defined as a quadruple.

$$G = (N, \Sigma, P, S)$$

N is a non-empty finite set of non-terminals or variables,

Σ is a non-empty finite set of terminal symbols such that $N \cap \Sigma = \phi$

$S \in N$, is a special non-terminal (or variable) called the start symbol, and $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ is a finite set of production rules.

The binary relation defined by the set of production rules is denoted by $\rightarrow$, i.e. $\alpha \rightarrow \beta$ iff $(\alpha, \beta) \in P$.

In other words, P is a finite set of production rules of the form $\alpha \rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^+$ and $\beta \in (N \cup \Sigma)^*$

Production rules:

The production rules specify how the grammar transforms one string to another. Given a string $\delta\alpha\gamma$, we say that the production rule $\alpha \rightarrow \beta$ is applicable to this string, since it is possible to use the rule $\alpha \rightarrow \beta$ to rewrite the α (in $\delta\alpha\gamma$) to β obtaining a new string $\delta\beta\gamma$. We say that $\delta\alpha\gamma$ derives $\delta\beta\gamma$ and is denoted as

$$\delta\alpha\gamma \Rightarrow \delta\beta\gamma$$

Successive strings are dervied by applying the productions rules of the grammar in any arbitrary order. A particular rule can be used if it is applicable, and it can be applied as many times as described.

We write $\alpha \overset{*}{\Rightarrow} \beta$ if the string β can be derived from the string α in zero or more steps; $\alpha \overset{+}{\Rightarrow} \beta$ if β can be derived from α in one or more steps.

By applying the production rules in arbitrary order, any given grammar can generate many strings of terminal symbols starting with the special start symbol, S, of the grammar. The set of all such terminal strings is called the language generated (or defined) by the grammar.

Formaly, for a given grammar $G = (N, \Sigma, P, S)$ the language generated by G is

$$L(G) = \left\{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \right\}$$

That is $w \in L(G)$ iff $S \overset{*}{\Rightarrow} w$.

If $w \in L(G)$, we must have for some $n \geq 0$, $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \cdots \Rightarrow \alpha_n = w$, denoted as a derivation sequence of *w*, The strings $S = \alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_n = w$ are denoted as sentential forms of the derivation.

Example : Consider the grammar $G = (N, \Sigma, P, S)$, where N = {*S*}, Σ ={*a*, *b*} and P is the set of the following production rules

$$\{ S \rightarrow ab, S \rightarrow aSb \}$$

Some terminal strings generated by this grammar together with their derivation is given below.

S $\Rightarrow$ ab

S $\Rightarrow$ aSb $\Rightarrow$ aabb

S $\Rightarrow$ aSb $\Rightarrow$ aaSbb $\Rightarrow$ aaabbb

It is easy to prove that the language generated by this grammar is

$$L(G) = \{ a^i b^i \mid i \geq 1 \}$$

By using the first production, it generates the string ab (for i =1).

To generate any other string, it needs to start with the production $S \rightarrow aSb$ and then the non-terminal S in the RHS can be replaced either by ab (*in which we get the string aabb*) or the same production $S \rightarrow aSb$ can be used one or more times. Every time it adds an '*a*' to the left and a '*b*' to the right of S, thus giving the sentential form $a^i S b^i, \ i \geq 1$. When the non-terminal is replaced by ab (which is then only possibility for generating a terminal string) we get a terminal string of the form $a^i b^i, \ i \geq 1$.

There is no general rule for finding a grammar for a given language. For many languages we can devise grammars and there are many languages for which we cannot find any grammar.

Example: Find a grammar for the language $L = \{ a^n b^{n+1} \mid n \geq 1 \}$.

It is possible to find a grammar for L by modifying the previous grammar since we need to generate an extra b at the end of the string. $a^n b^n, n \geq 1$ We can do this by adding a production $S \rightarrow Bb$ where the non-terminal B generates $a^i b^i, \ i \geq 1$ as given in the previous example.

Using the above concept we devise the follwoing grammar for L.

$G = (N, \Sigma, P, S)$ where, N = { *S*, *B* }, P = { $S \rightarrow Bb, B \rightarrow ab, B \rightarrow aBb$ }

Parse Trees:

There is a tree representation for derivations that has proved extremely useful. This tree shows us clearly how the symbols of a terminal string are grouped into substrings, each of which belongs to the language of one of the variables of the grammar. But perhaps more importantly, the tree, known as a "parse tree"

Construction of a Parse tree:

Let us fix on a grammar $G = (V, T, P, S)$. The *parse trees* for G are trees with the following conditions:

1. Each interior node is labeled by a variable in V.
2. Each leaf is labeled by either a variable, a terminal, or ϵ. However, if the leaf is labeled ϵ, then it must be the only child of its parent.
3. If an interior node is labeled A, and its children are labeled

$$X_1, X_2, \ldots, X_k$$

respectively, from the left, then $A \to X_1 X_2 \cdots X_k$ is a production in P. Note that the only time one of the X's can be ϵ is if that is the label of the only child, and $A \to \epsilon$ is a production of G.

Example 5.10: Figure 5.5 shows a parse tree for the palindrome grammar of Fig. 5.1. The production used at the root is $P \to 0P0$, and at the middle child of the root it is $P \to 1P1$. Note that at the bottom is a use of the production $P \to \epsilon$. That use, where the node labeled by the head has one child, labeled ϵ, is the only time that a node labeled ϵ can appear in a parse tree. □

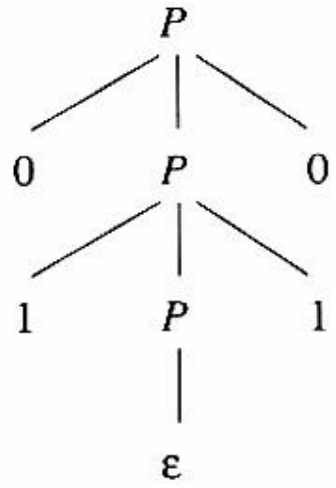

Figure 5.5: A parse tree showing the derivation $P \stackrel{*}{\Rightarrow} 0110$

Yield of a Parse tree:

If we look at the leaves of any parse tree and concatenate them from the left, we get a string, called the *yield* of the tree, which is always a string that is derived from the root variable. The fact that the yield is derived from the root will be proved shortly. Of special importance are those parse trees such that:

1. The yield is a terminal string. That is, all leaves are labeled either with a terminal or with ϵ.

2. The root is labeled by the start symbol.

Ambiguity in languages and grammars:

When a grammar fails to provide unique structures, it is sometimes possible to redesign the grammar to make the structure unique for each string in the language. Unfortunately, sometimes we cannot do so. That is, there are some CFL's that are "inherently ambiguous"; every grammar for the language puts more than one structure on some strings in the language.

grammar lets us generate expressions with any sequence of $*$ and $+$ operators, and the productions $E \to E + E \mid E * E$ allow us to generate these expressions in any order we choose.

Example 5.25: For instance, consider the sentential form $E + E * E$. It has two derivations from E:

1. $E \Rightarrow E + E \Rightarrow E + E * E$
2. $E \Rightarrow E * E \Rightarrow E + E * E$

Notice that in derivation (1), the second E is replaced by $E * E$, while in derivation (2), the first E is replaced by $E + E$. Figure 5.17 shows the two parse trees, which we should note are distinct trees.

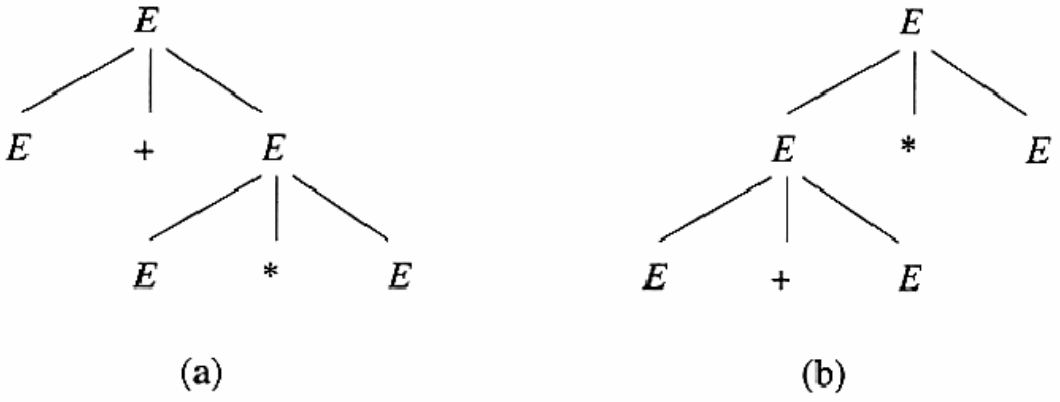

Figure 5.17: Two parse trees with the same yield

we say a CFG $G = (V, T, P, S)$ is *ambiguous* if there is at least one string w in T^* for which we can find two different parse trees, each with root labeled S and yield w. If each string has at most one parse tree in the grammar, then the grammar is *unambiguous*.