II Unit

REGULAR EXPRESSIONS

Regular Language:

A language is said to be a regular language if and only if some finite state machine recognizes it.

The Language which are not recognized by FSM is not a regular Language.

The languages accepted by finite automata are easily described by simple expression called regular expressions.

The regular expressions are useful for representing certain sets of strings in an algebraic fashion. These describe the languages accepted by finite state automata.

Regular Set:
Any set that represents the value of the Regular Expression is called a **Regular Set.**

Properties of Regular Sets:

Property 1: . The union of two regular set is regular.

Proof −

Let us take two regular expressions

$RE_1 = a(aa)^*$ and $RE_2 = (aa)^*$

So, L_1 = {a, aaa, aaaaa,.....} (Strings of odd length excluding Null)

and L_2 ={ ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 \cup L_2$ = { ε, a, aa, aaa, aaaa, aaaaa, aaaaaa,.....}

(Strings of all possible lengths including Null)

RE ($L_1 \cup L_2$) = a* (which is a regular expression itself)

Hence, proved.

Property 2. The intersection of two regular set is regular.

Proof −

Let us take two regular expressions

$RE_1 = a(a^*)$ and $RE_2 = (aa)^*$

So, L_1 = { a,aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 \cap L_2$ = { aa, aaaa, aaaaaa,.....} (Strings of even length excluding Null)

RE ($L_1 \cap L_2$) = aa(aa)* which is a regular expression itself.

Hence, proved.

Property 3. The complement of a regular set is regular.

Proof −

Let us take a regular expression −

RE = (aa)*

So, L = {ε, aa, aaaa, aaaaaa, ...} (Strings of even length including Null)

Complement of **L** is all the strings that is not in **L**.

So, L' = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

RE (L') = a(aa)* which is a regular expression itself.

Hence, proved.

Property 4. The difference of two regular set is regular.

Proof −

Let us take two regular expressions −

$RE_1 = a\ (a^*)$ and $RE_2 = (aa)^*$

So, L_1 = {a, aa, aaa, aaaa, ...} (Strings of all possible lengths excluding Null)

L_2 = { ε, aa, aaaa, aaaaaa,.....} (Strings of even length including Null)

$L_1 - L_2$ = {a, aaa, aaaaa, aaaaaaa, ...}

(Strings of all odd lengths excluding Null)

RE ($L_1 - L_2$) = a (aa)* which is a regular expression.

Hence, proved.

Property 5. *The reversal of a regular set is regular.*

Proof −

We have to prove $\mathbf{L^R}$ is also regular if **L** is a regular set.

Let, L = {01, 10, 11, 10}

RE (L) = 01 + 10 + 11 + 10

L^R = {10, 01, 11, 01}

RE (L^R) = 01 + 10 + 11 + 10 which is regular

Hence, proved.

Property 6. The closure of a regular set is regular.

Proof −

If L = {a, aaa, aaaaa, ...} (Strings of odd length excluding Null)

i.e., RE (L) = a (aa)*

L* = {a, aa, aaa, aaaa , aaaaa,……} (Strings of all lengths excluding Null)

RE (L*) = a (a)*

Hence, proved.

Property 7. The concatenation of two regular sets is regular.

Proof −

Let RE_1 = (0+1)*0 and RE_2 = 01(0+1)*

Here, L_1 = {0, 00, 10, 000, 010, ...} (Set of strings ending in 0)

and L_2 = {01, 010,011,.....} (Set of strings beginning with 01)

Then, L_1 L_2 = {001,0010,0011,0001,00010,00011,1001,10010,.....}

Set of strings containing 001 as a substring which can be represented by an

RE − (0 + 1)*001(0 + 1)*

Hence, proved.

A formal recursive definition of regular expressions over ∑ is as Follows:

1. Any terminal symbol (i.e. an element of ∑), and symbols are regular expressions which we, denoted by **a**.(bold letter) where a in ∑.

2. The union of two regular expressions **R1 and R2** written as **R 1 + R2,** is also a regular expression.

3. The concatenation of two regular expressions **R1and R2**, written as **R1 R2,** is also a regular expression.

4. The iteration (or closure) of a regular expression **R** written as **R*,** is also a regular expression.

5. If **R** is a regular expression, then **(R)** is also a regular expression.

The regular expressions over ∑ are obtained recursively by the application of the rules *1to 5* once or several times.

Identities of Regular Expressions

I_1 $\quad \emptyset + R = R$

I_2 $\quad \emptyset R = R\emptyset = \emptyset$

I_3 $\quad \Lambda R = R\Lambda = R$

I_4 $\quad \Lambda^* = \Lambda$ and $\emptyset^* = \Lambda$

I_5 $\quad R + R = R$

I_6 $\quad R^*R^* = R^*$

I_7 $\quad RR^* = R^*R$

I_8 $\quad (R^*)^* = R^*$

I_9 $\quad \Lambda + RR^* = R^* = \Lambda + R^*R$

I_{10} $\quad (PQ)^*P = P(QP)^*$

I_{11} $\quad (P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

I_{12} $\quad (P + Q)R = PR + QR$ and $R(P + Q) = RP + RQ$

Arden's Theorem:

Let P and Q be two regular expressions over ∑. If P does not contain, then the following equation in R

$$R = Q + RP \quad (4.1)$$

has a unique solution (i.e. one and only one solution) given by R = QP*.

Proof: $Q + (QP^*)P = Q(\varepsilon + P^*P) = QP^*$ by I_9

Hence (4.1) is satisfied when R =QP*. This means **R =QP*** is a solution of (4.1).

To prove uniqueness, consider (4.1). Here, replacing R by Q + **RP** on the R.H.S., we get the equation

$$\begin{aligned} &Q + RP = Q + (Q + RP)P \\ &= Q + QP + RPP \\ &= Q + QP + RP^2 \\ &= Q + QP + QP2 + + QP^i + RP^{i+1} \\ &= Q(\varepsilon + P + p^2 + + p^i) + RP^{i+1} \end{aligned}$$

From (4.1),

$$R = Q(\varepsilon + P + p^2 + ... + p^i) + RP^{i+1} \quad \text{for } i \geq 0 \quad (4.2)$$

We now show that any solution of (4.1) is equivalent to **QP*.** Suppose **R** satisfies (4.1), then it satisfies (4.2). Let *w* be a string of length i in the set **R** .Then *w* belongs to the set $Q(\varepsilon + P + p^2 + ... + p^i) + RP^{i+1}$

As **P** does not contain, RP^{i+1} has no string of length less than i + 1 and so *w* is not in the set RP^{i+1}

This means that *w* belongs to the set $Q(\varepsilon + P + p^2 + ...P^i)$

Consider a string *w* in the set **QP*.** Then *w* is in the set QP^k for some $k \geq 0$, and hence in $Q(\varepsilon + P + p^2 + \ldots + p^k)$

So *w* is on the R.H.S. of (4.2). Therefore, *w* is in R (L.H.S. of (4.2)). Thus **Rand QP*** represent the same set. This proves the uniqueness of the solution of (4.1).

Equivalence of finite automata and regular expressions:

Statement: Let r be a regular expression. Then there exists an NFA with -transitions that accepts L(r).

Proof: We show by induction on the number of operators in the regular expression 'r' that there is an NFA M with -transitions, having one final state and no transitions out of this final state, such that L(M)= L(r).

Basis: (Zero operators): The expression r must be ,Ø, or a for some a in∑.

The NFA's in Fig. 2.13(a), (b), and (c) clearly satisfy the conditions.

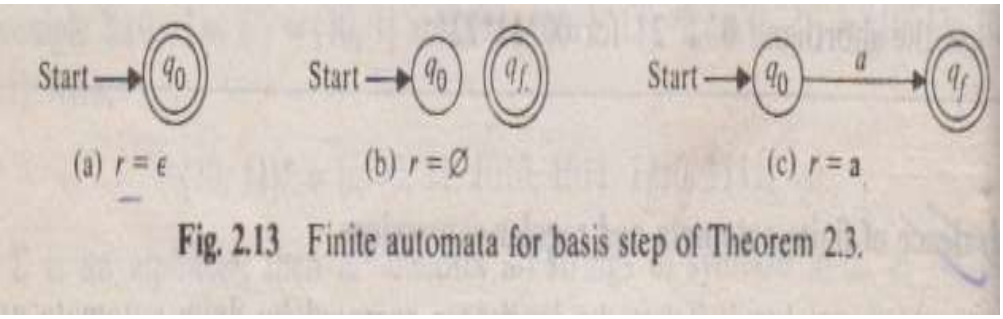


Fig. 2.13 Finite automata for basis step of Theorem 2.3.

Induction: (One or more operators) : Assume that the theorem is true for regular expressions with fewer than i operators, i≥ 1. Let r have i operators. There three cases depending on the form of r.

Case 1: r = r1 + r2.

Both r1 and r2 must have fewer than i operators. Thus there are NFA's

$M_1 = (Q1. \sum_1, \ _1, q_1, \{f_1\})$ and

$M2 = (Q_2 , \ \sum_2, \ _2, q_2, \{f_2\})$ with L(M1)= $L(r_1)$ and L(M2) = $L(r_2)$.

We may assume Q_1 and Q_2 are disjoint. Let q_0 be a new initial state and f_0 a new final state.

Construct

$M= (Q_1 \ U \ Q_2 \ U \ \{q_0, f_0\}, \sum_1, \ \sum_2,, q_0, \{f_0\})$,

Where is defined by

i). $(q_0,) = \{q_1, q_2\}$,

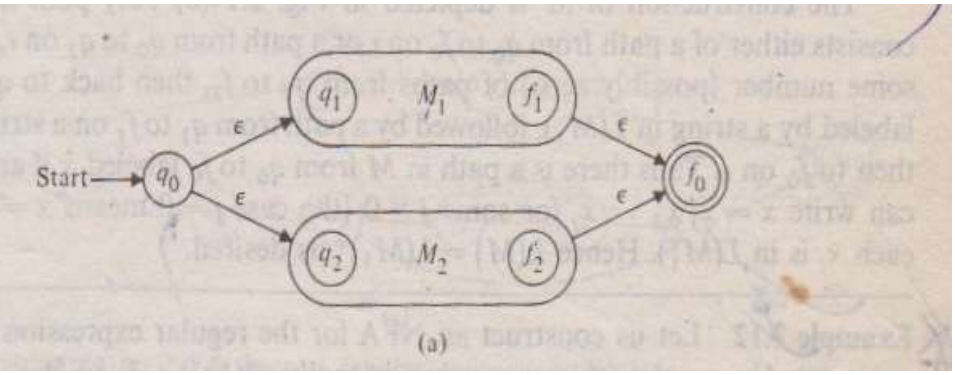
ii) $(q,a) = _1(q,a)$ for q in Q_1-$\{f_1\}$ and a in $\sum_1$ U {

iii) $(q,a) = _2(q,a)$ for q in Q_2-$\{f_2\}$ and a in $\sum_2$ U {

iv) $(f_1,) = (f_2,) = \{f_0\}$

by the inductive hypothesis that there are no transitions out of f_1 or f_2 in M_1 or M_2. Thus all the moves of M1 and M2 are present in M.

The construction of M is depicted in Fig 2.14(a).



Any path in the transition diagram of M from q_0 to f_0 must begin by going to either q_1 or q_2 on . If the path goes to q_1, it may follow any path in M_1 to f_1 and then go to f_0 on .
Similarly, the path that begin by going to q_2 may follow any path in M_2 tof_2 and then go to f_0 on .

These are the only paths from q_0 to f_0. It follows immediately that there is a path labeled ' *x* ' in M from q_0 tof_0 if and only if there is a path labeled *x* in M_1 from $q_1$to f_1 or a path in M_2 from q_2 to f_2.

Hence $L(M) = L(M_1) \cup L(M_2)$.

Case2: r = r1r2

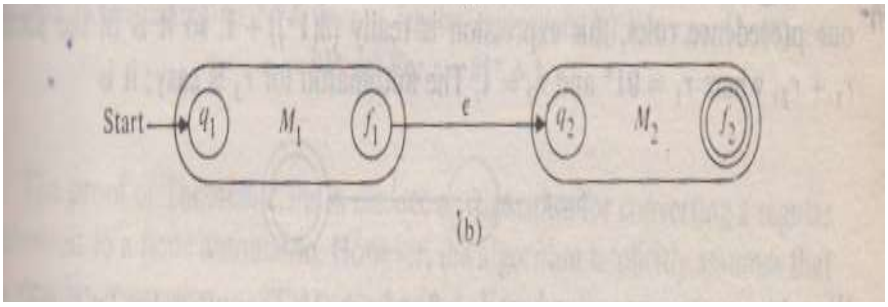
Let M1 and M2 be defined as in case1

Construct: $M = (Q_1 \cup Q_2, \sum_1 \cup \sum_2, {}_1, \{q_1\}, \{f_2\})$ where

Where is given by

i) $(q,a) = {}_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\sum_1 \cup \{$.
ii) $(f_1,) = \{q2\}$
iii) $(q,a) = {}_2(q, a)$ for q in Q_2 and a in $\sum_2 \cup \{$.

The construction of M is given in Fig. 2.14(b).



Every path in M from q_1 to f_2 is a path labeled by some string *x* from q_1 tof_1, followed by the edge from f_1 to q_2 by labeled, followed by a path labeled by some string 'y' from q_2 tof_2.

Thus L(M) ={ *xy* |*x* is in L(M_1) and y is in L(M_2)} and
Hence L(M) = L(M_1)L(M_2).

Case 3: r= r_1*

Let $M_1 = (Q_1. \sum_1, {}_1, q_1, \{f_1\})$ and $L(M_1)=r_1$

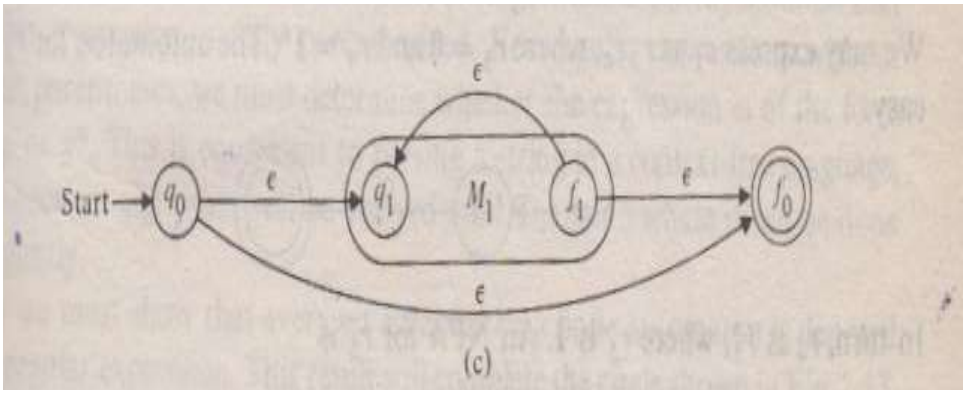
Construct : $M = (Q_1 U \{q_0, f_0\}, \sum_1, , q_0, \{f_0\})$

Where

is given by

i) $(q_o,) = (f_1,)=\{q_1,f_0\}$
ii) $(q,a)=_1(q,a)$ for q in Q_1 - $\{f_1\}$ and a in $\sum_1 U \{ \}$

The construction of M is depicted in Fig. 2.14(c).



Any path from q_0 to f_0 consists either of a path from q_0 tof_0 on or a path from q_0 to q_1 on , followed by some number (possibly zero) of paths from q_1 to f_1, then back to q_1 on , each labeled by a string in $L(M_1)$, followed by a path from q_1 to f_1 on a string in $L(M_1)$, then to f_0 on .Thus there is a path in M from q_0 to f_0 labeled x if and only if we can write $x = x_1 x_2 ... x_j$ for some $j \geq 0$(the case j=0 means x=) such that each x_i is in $L(M_1)$.

Hence $L(M) = L(M_1)^*$

Regular Grammar(Type 3 Grammar)

A regular grammar or type-3 defines the language called regular language that is accepted by finite Automata

A Regular Grammar is denoted G = (V. T, P. S), where

1. V is a finite non-emptyset of variables or non terminal symbols.
2. T is a finite set of terminals. We assume that V and T are disjoint.
3. P is a finite set of rules or productions; each production is of the form A→ where A is a variable and is a string of symbols from (V ∑)*.
4. Finally, S is a special variable called the start symbol.

Variables are represented by A, B, C……
Terminals are represented by a, b,c…..

Left and Right Linear Grammars

Right Linear Grammar Definition:

A grammar G =(V, T, S, P) is said to be right-linear if all productions are of the form

A → xB,

A → x,

where A, B ∈ V, and x ∈ T*.

Example:

S → aS | bA | b
A → abA

Left Linear Grammar Definition:

A grammar is said to be left-linear if all productions are of the form $A \rightarrow Bx$,

or $A \rightarrow x$.

A regular grammar is one that is either right-linear or left-linear

Example:

$S \rightarrow Sa \mid Ab \mid b$

$A \rightarrow a \mid b$

Constructing of Finite Automata From Grammar

Finite Automata:

A FA is a collection of 5-Tuple **(Q, Σ, q0, F, δ)** where :

Q is Finite Set of States, $Q \neq \phi$.

Σ is Input Alphabet.

q0 is an Initial State, $q0 \in Q$.

F is Final State.

δ is a Transition Function, $Q \times \Sigma \rightarrow Q$.

In a Grammar: (V, T, P, S)

V is set of **variables** or **Non Terminals,**

T is a set of **terminals,**

P is the **production rules** and,

S is the **starting symbol.**

S = q0 V = Q T = Σ P = δ

In a Finite Automata: (Q, Σ, q0, F, δ)

q0 is the initial **state,**

Q refers to the **set of states,**

Σ refers to the **input symbols** and

δ refers to the **transition functions**

Rules:

1 : Include edge **δ(qi,a) = qj,** if **Ai →aAj** exists.

OR

Include an edge state **qi → qj** with an edge label **'a',** if production **Ai →aAj** exists in the grammar.

2 : Include edge **δ(qi,a) = qf,** if **Ai →a** exists.

OR

Include an edge state **qi → qf** with an edge label **'a',** if production **Ai →a** exists in the grammar.

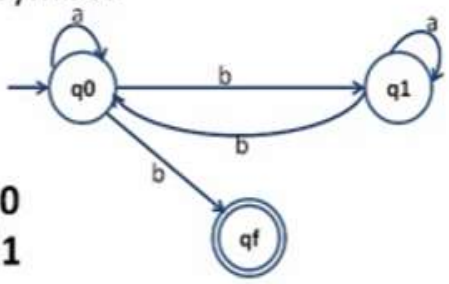
Here '**a**' can be **ε,** in that case **qf is final state without any transition.**

Example:

Construct a FA recognizing L(G), where G is the grammar **A0** with starting symbol.

A0 → a**A0** |b**A1** | b

A1 → a**A1** | b**A0** | a



A0 → a**A0** ⇔ **δ(q0,a) =q0**

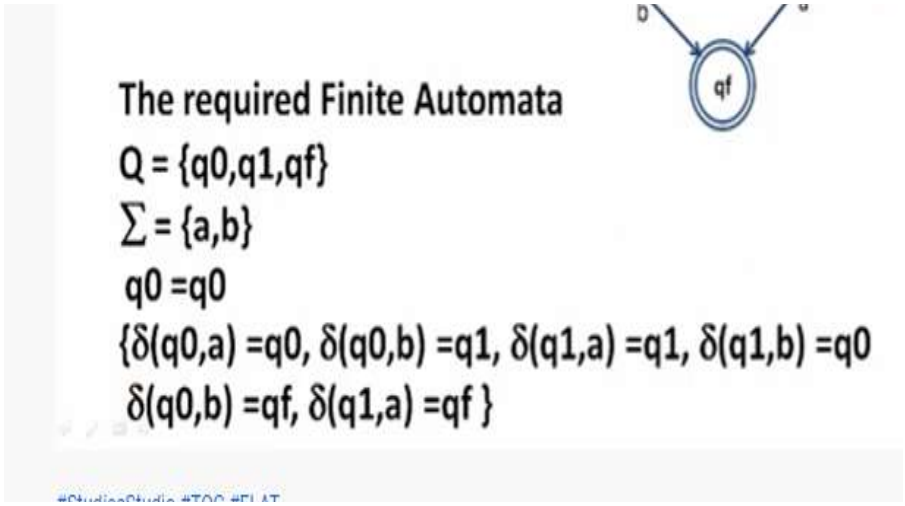
A0 → b**A1** ⇔ **δ(q0,b) =q1**

A1 → a**A1** ⇔ **δ(q1,a) =q1**

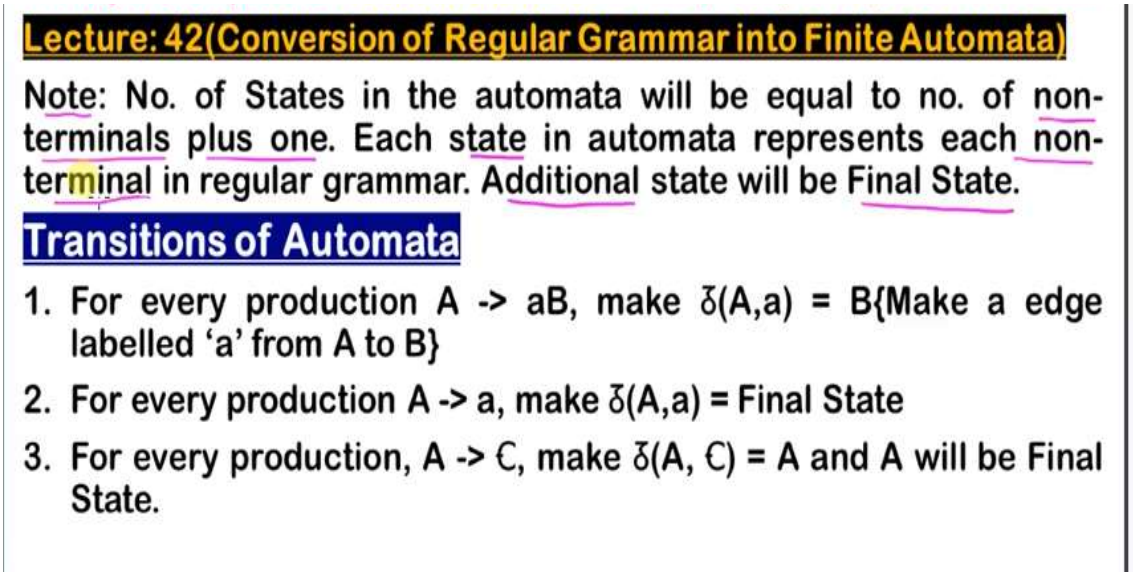
A1 → b**A0** ⇔ **δ(q1,b) =q0**

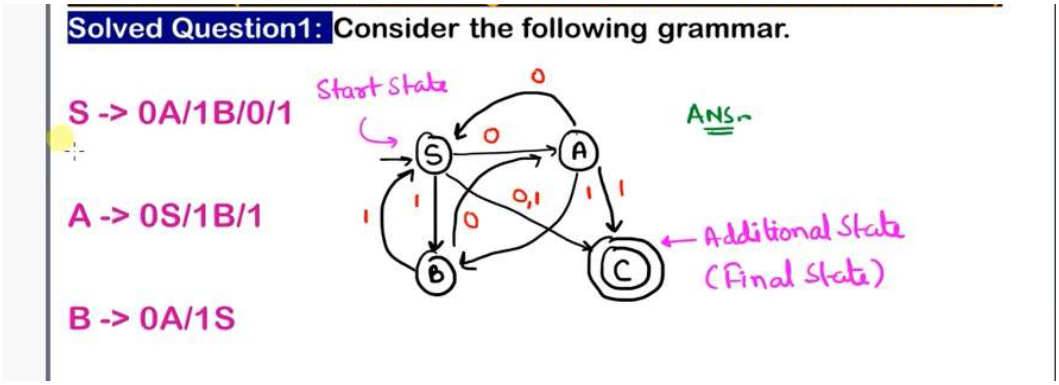
A0 → b ⇔ **δ(q0,b) =qf**

A1 → a ⇔ **δ(q1,a) =qf**



Conversion of Regular Grammar to Finite automata



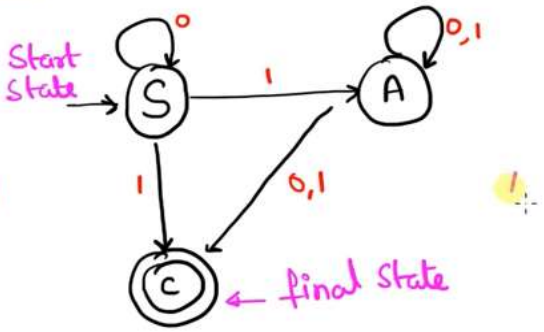


Lecture: 42(Conversion of Regular Grammar into Finite Automata)

Solved Question2: Consider the following grammar.

S -> 0S/1A/1

A -> 0A/1A/0/1



Solved Question3: Consider the following grammar. Trace the transition to accept the string abba.

S -> abA

S -> B

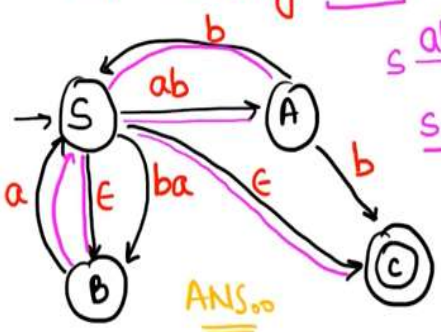
S -> baB

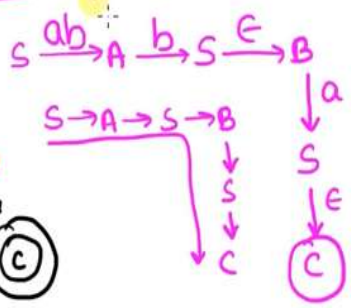
S -> €

A -> bS

B -> aS

A -> b





Conversion of RLG to LLG

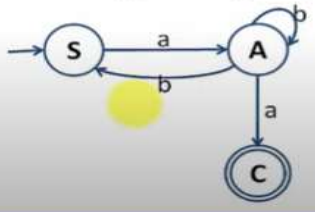
Method:

1: Convert the given Right Linear grammar to FA.
2: Interchange the initial and Final states of FA.
3: Reverse the directions of all the transitions.
4: Construct the regular grammar from this FA.

Conversion of RLG to LLG

Example: 1
Convert the following RLG to LLG
S→aA
A → bA | bS |a

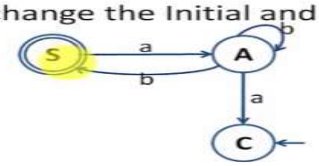
Step 1: Convert the given Right Linear grammar to FA.



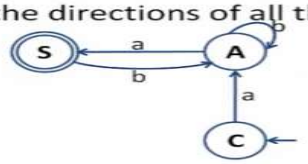
Conversion of RLG to LLG

Example: 1

Step 2: Interchange the Initial and Final states of FA.



Step 3: Reverse the directions of all the transitions.



Step 4: Construct the regular grammar from this FA.

The required LLG is

$C \rightarrow Aa$

$A \rightarrow Ab \mid a$

$S \rightarrow Ab$

Ex: Convert the Following RLG to LLG

S->aS|cA

A->bA|a

Conversion of LLG to RLG

Method:

1: Convert the given Left Linear grammar to FA.
2: Interchange the initial and Final states of FA.
3: Reverse the directions of all the transitions.

Example: 1

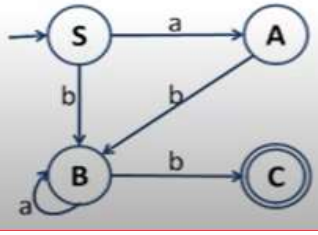
Convert the following LLG to RLG

S→Aa | Bb

A → Bb

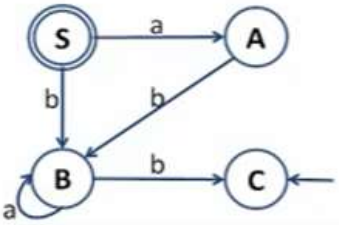
B →Ba | b

Step 1: Convert the given Left Linear grammar to FA.



Example: 1

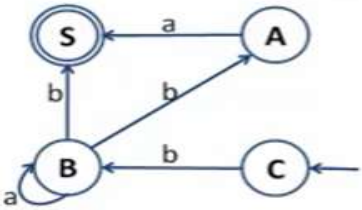
Step 2: Interchange the Initial and Final states of FA.



Step 3: Reverse the directions of all the transitions.

Example: 1

Step 3: Reverse the directions of all the transitions.



Step 4: Construct the regular grammar from this FA.

The required RLG is

C→ bB

B → aB | bA | b

A → a

Example: 2

Convert the following LLG to RLG

S→Ca | Aa | Bb

A → Ab | Ca |Bb | a

B →Bb | b

C →Aa

Step 1: Convert the given Left Linear grammar to FA.

Theorem:(Conversion of Right Linear Grammar to Finite Automata).

Let $G=(V, T, S, P)$ be a right-linear grammar. Then $L(G)$ is a regular language.

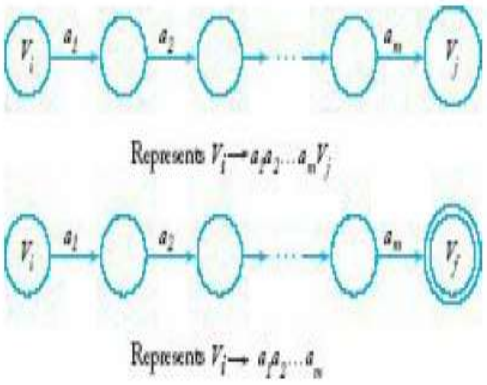
Proof: We assume that $V = \{ V_0, V_1, \ldots \}$, that $S = V_0$, and that we have productions of the form $V_0 \rightarrow v_1 V_i, V_i \rightarrow v_2 V_j, \ldots$ or $V_n \rightarrow v_l, \ldots$. If w is a string in $L(G)$, then because of the form of the productions

$$\begin{aligned} V_0 &\Rightarrow v_1 V_i \\ &\Rightarrow v_1 v_2 V_j \\ &\overset{*}{\Rightarrow} v_1 v_2 \cdots v_k V_n \\ &\Rightarrow v_1 v_2 \cdots v_k v_l = w. \end{aligned} \tag{3.4}$$

The initial state of the automaton will be labeled V_0, and for each variable V_i there will be a nonfinal state labeled V_i. For each production

$$Vi \rightarrow a_1 a_2 \cdots a_m V_j,$$

Figure3.16



the automaton will have transitions to connect Vi and Vj that is,δ will be defined so that

$$\delta^* (V_i, a_1 a_2 \cdots a_m) = V_j.$$

For each production

$$V_i \rightarrow a_1 a_2 \cdots a_m,$$

the corresponding transition of the automaton will be

$$\delta^* (V_i, a_1 a_2 \ldots a_m) = V_f,$$

where V_f is a final state.

The General figure is shown in fig 3.16.

Suppose now that $w \in L(G)$ so that (3.4) is satisfied. In the nfa there is, by construction, a path from V_0 to V_i labeled v_1, a path from V_i to V_j labeled v_2, and so on, so that clearly

$$Vf \in \delta^* (V_0, w),$$

And W is accepted by M.

Conversely, assume that w is accepted by M. Because of the way in which M was constructed, to accept w the automaton has to pass through a sequence of states $V_0, V_i, \ldots$ to V_f, using paths labeled $v_1, v_2, \ldots$. Therefore, w must have the form

$$w = v_1 v_2 \cdots v_k v_l$$

and the derivation

$$V_o \Rightarrow v_1 V_i \Rightarrow v_1 v_2 V_j \overset{*}{\Rightarrow} v_1 v_2 \cdots v_k V_k \Rightarrow v_1 v_2 \cdots v_k v_l$$

is possible. Hence w is in $L(G)$, and the theorem is proved.

Theorem: (Conversion of FA To Right Linear Grammar)

If L is a regular language on the alphabet $\in$, then there exists a right-linear grammar $G = (V, E, S, P)$ such that $L = L < (G)$.

Proof: Let $M = (Q, E, \delta, q_0, F)$ be a dfa that accepts L. We assume that $Q = \{q_0, q_1, \ldots, qn\}$ and $\Sigma = \{a_1, a_2, \ldots, a_m\}$. Construct the right-linear grammar $G = (V, E, S, P)$ with

$$V = \{q_0, q_1, \ldots, q_n\}$$

and $S = q_0$. For each transition

$$\delta(q_i, a_j) = q_k$$

of M, we put in P the production

$$q_i \rightarrow a_j q_k. \qquad (3.5)$$

In addition, if q_k is in F, we add to P the production

$$q_k \rightarrow \lambda. \qquad (3.6)$$

Consider w ∈ L of the form

$$w = a_i a_j \ldots . a_k a_l.$$

For M to accept this string it must make moves via

$$\begin{aligned}
\delta(q_0, a_i) &= q_p, \\
\delta(q_p, a_j) &= q_r, \\
&\vdots \\
\delta(q_s, a_k) &= q_t, \\
\delta(q_t, a_l) &= q_f \in F.
\end{aligned}$$

By construction, the grammar will have one production for each of these δ's. Therefore, we can make the derivation

$$\begin{aligned}
q_0 &\Rightarrow a_i q_p \Rightarrow a_i a_j q_r \overset{*}{\Rightarrow} a_i a_j \cdots a_k q_t \\
&\Rightarrow a_i a_j \cdots a_k a_l q_f \Rightarrow a_i a_j \cdots a_k a_l,
\end{aligned} \tag{3.7}$$

with the grammar G, and $w \in L(G)$.

Conversely, if $w \in L(G)$, then its derivation must have the form (3.7). But this implies that

$$\delta^*(q_0, a_i a_j \ldots a_k a_i) = q_f,$$

Completing the Proof.

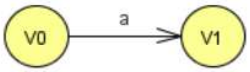
Example

$\delta(q_0, a) = \{q_1\}$	$q_0 \rightarrow aq_1$
$\delta(q_1, a) = \{q_2\}$	$q_1 \rightarrow aq_2$
$\delta(q_2, b) = \{q_2\}$	$q_2 \rightarrow bq_2$
$\delta(q_2, a) = \{q_f\}$	$q_2 \rightarrow aq_f$
$q_f \in F$	$q_f \rightarrow \lambda$

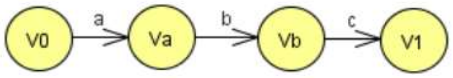
Converting Regular Grammar to DFA

Assume that a regular grammar is given in its right-linear form, this grammar may be easily converted to a DFA. A right-linear grammar, defined by G = (V, T, S, P), may be converted to a DFA, defined by M = (Q, Σ, δ, q_0, F) by:

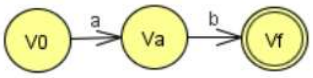
1. Create a state for each variable.
2. Convert each production rule into a transition.
   a. If the production rule is of the form $V_i \rightarrow aV_j$, where $a \in T$, add the transition $\delta(V_i, a) = V_j$ to M. For example, $V_0 \rightarrow aV_1$ becomes:



   b. If the production rule is of the form $V_i \rightarrow wV_j$, where $w \in T^*$, create a series of states which derive w and end in V_j. Add the states in between to Q. For example, $V_0 \rightarrow abcV_1$ becomes:



   c. If the production rule is of the form $V_i \rightarrow w$, where $w \in T^*$, create a series of states which derive w and end in a final state. For example, $V_0 \rightarrow a$ becomes:



The regular grammar for this language:

$$S \rightarrow 0A \mid 1A,$$

$$A \rightarrow 0A \mid 1A \mid +B \mid -B,$$

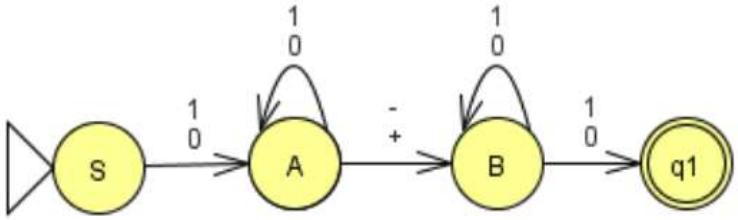
$$B \rightarrow 0B \mid 1B \mid 0 \mid 1.$$

We define the resulting DFA, M = ((Q, Σ, δ, S, F), where δ (right column) is derived from the algorithm above for each of the production rules (left column) of the table below, C ∈ F.

S → 0A	δ(S, 0) = A
S → 1A	δ(S, 1) = A
A → 0A	δ(A, 0) = A
A → 1A	δ(A, 1) = A
A → +B	δ(A, +) = B
A → -B	δ(A, -) = B

B → 0B	δ(B, 0) = B
B → 1B	δ(B, 1) = B
B → 0	δ(B, 0) = C
B → 1	δ(B, 1) = C

The DFA for this language is shown below.



PUMPING LEMMA FOR REGULAR SETS.

The Language L is categorized into two types Finite and infinite.

If any L is finite then it is regular. If the language is infinite then it may or may not be regular.

Let L={ab,abab,ababab……..}

If infinite language has to be regular then they had to be a FA that accepts Infinite language.

If infinite language has to be accepted by FA then there has to be loop inside the Finite automata, and there should be pattern.

Pumping lemma, is a powerful tool for proving certain languages non- regularly. It is also useful for development of algorithms to answer to certain questions concerning finite automata such as whether the language accepted by a given FA is finite or infinite.

If a language is regular, it is accepted by a DFA M =(Q, ∑, , q_0, F) with some particular number of states, say n. Consider an input of n or more symbols $a_1,a_2,….,a_m$, m ≥ n and for i=1, 2 ….m.

If L is a regular language then all strings in it, of length greater than or equal to N, can be divided into three parts. Such that if you either remove or repeat the middle part any number of times, the resulting string will still be in L.

Let (q_0, $a_1a_2….$ a_i) = q_i. It is not possible for each of the n + 1 states q_0, q_1,..., q_n to be distinct, since there are only n different states.

Thus there are two integers j and k, 0 ≤ j <k≤ n, such that $q_j = q_k$

The path labeled a_1 a_2 ……..a_m in the transition diagram of M is illustrated in Fig. 3.1.

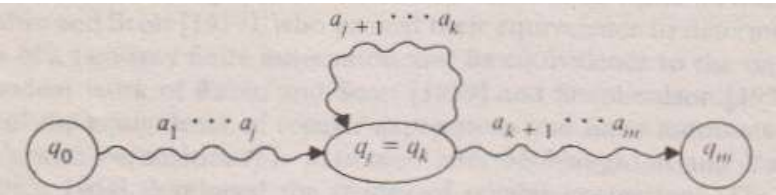


Fig. 3.1 Path in transition diagram of DFA *M*.

Since j < k, the string $a_{j+1} \ldots\ldots a_k$ is of length 1, and since k ≤ n its length is no more than n.

If q_m is in F, that is, $a_1\ a_2, \ldots\ldots a_m$ is in L(M), then $a_1\ a_2 \ldots\ldots a_j a_{k+1}\ a_{k+2} \ldots .. a_m$ is also in L(M), since there is a path from q_0 to q_m that goes through q_j but not around the loop labeled $a_{j+1}, \ldots\ldots\ a_k$. Formally, it is expressed as

$$\begin{aligned}\delta(q_0, a_1 \cdots a_j a_{k+1} \cdots a_m) &= \delta(\delta(q_0, a_1 \cdots a_j), a_{k+1} \cdots a_m) \\ &= \delta(q_j, a_{k+1} \cdots a_m) \\ &= \delta(q_k, a_{k+1} \cdots a_m) \\ &= q_m.\end{aligned}$$

Similarly, we could go around the loop of Fig. 3.1 more than once.

Thus, $a_1, \ldots . a_j\ (a_{j+1} \ldots\ldots\ a_k)^i\ a_{k+1} \ldots\ldots\ a_m$ is in L(M) for any i≥0.

So, in the pumping lemma we have to prove is that given any sufficiently long string accepted by an FA, we can find a substring near the beginning of the string that may be "pumped." i.e., repeated as many times as we like, and the resulting string will be accepted by the FA.

Note: The pumping lemma is true only when any pattern is exist in the L.

For example L={ a^n| n is even}

L= {a^0, a^2, a^4…….}

In this set there is a pattern i,e it is in Arithmetic progression.(0,2,4……)

Consider a machine M=({q0,q1,q2,q3},{0,1}, ,q0,q3)

Theorem :(Pumping Lemma for Regular Sets)

Statement: of Let *M = (Q, ∑, , q_0, F)* be a finite automaton with n states. Let *L* be the regular set accepted by *M*. Let *w* *L* and |w| ≥ n. then there exists *x, y,z* such that *w =xyz, y* and $xy^i z$ *L* for each i ≥ 0.

Proof: Let $w=a_1a_2 \ldots\ldots a_m$, m ≥ n
$(q_0, a_1a_2 \ldots\ldots a_i)=q_i$ for i=1,2,…..m; Q ={ $q_0, q_1, \ldots\ldots q_m$}

That is, Q is the sequence of states with path value $w = a_1a_2.....a_m$. As there are only n distinct states, at least two states in Q must coincide.

Among the various pairs of repeated states, we take the first pair. Let us take them as q_j and q_k ($q_j=q_k$) Then *j* and *k* satisfy the condition $0 \leq jk \leq n$.

The string '*w'* can be decomposed into three substrings $a_1a_2.....a_j$, $a_{j+1},a_{j+2}.....a_k$ *and* $a_{k+1}.a_{k+2}.....a_m$

Let x, y, z denote these strings respectively, as k ≤ n |xy|≤n and w=xyz. The path with value 'w' in the transition diagram of M is shown in Fig 5.27

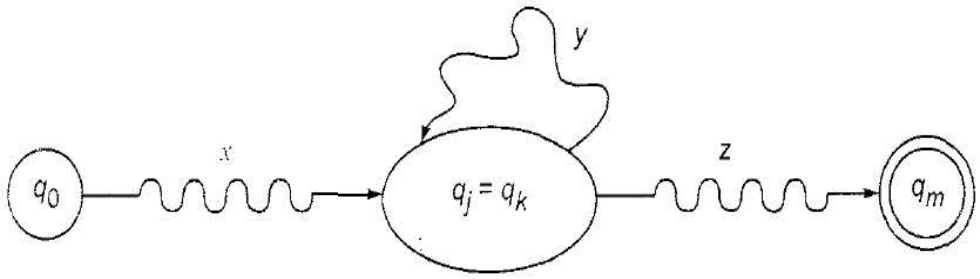


Fig. 5.27 String accepted by *M*.

The automaton '*M'* starts from the initial state q_0. On applying the string *x*, it reaches *qj(=q*k*)*. On applying the string *y,* it comes back to $q_j(= q_k)$. So after application of y^i for each $i \geq 0$, the automaton is in the same state q_j.

On applying z, it reaches q_m, a final state. Hence, $xy^iz \in L$. As every state in Q_1 is obtained by applying an input symbol, $y \neq$.

Note: The decomposition of a string is valid only for strings of length greater than or equal to the number of states. For such a string *w =xyz* we can 'iterate' the substring *y* in *xyz* as many times as we like and get strings of the form xy^iz, which are longer than *xyz* and are in *L*.

APPLICATION OF PUMPING LEMMA

The steps needed for proving that a given set is not regular are as follows

Step 1: Assume that *L* is regular. Let n be the number of states in the corresponding finite automaton.

Step 2 Choose a string w such that $|w| \geq n$. Use pumping lemma to write *w=xyz, wi*th $|xy| \leq n$ and $|y| > 0$.

Step 3 Find a suitable integer i such that xy^iz L. This contradicts our assumption, Hence *L* is not regular.

Context –Free Grammars

Each Language has its own Grammar. Grammar is a set of rules by which sentences in languages are constructed. Context free languages are applied in parser Design. It is also useful for describing block structure in Programming languages.

Context -Free Grammar

A Context Free Grammar (CFG or just grammar) is denoted G = (V. T, P. S), where

5. V is a finite set of variables or non terminal symbols.
6. T is a finite set of terminals. we assume that V and T are disjoint.
7. P is a finite set of rules or productions; each production is of the form A→ where A is a variable and is a string of symbols from (V T)*.
8. Finally, S is a special variable called the start symbol.

Variables are represented by A, B, C……
Terminals are represented by a, b,c…..

Language Generated by CFG are called Context Free Language.

We can present a grammar by simply listing its productions. If $A\rightarrow_1$, $A\rightarrow_2$ ..... $A\rightarrow_k$ are the productions for the variable A of some grammar, then we can express them by the notation,

A→1|2|……..k,

Where the vertical line is read “or.”

Derivation Trees.

The derivation in a CFG can be representing using trees. Such trees representing derivations are called derivation trees. It is also called as parse tree. In a derivation tree, the root is the start variable, all internal nodes are labeled with variables, while all leaves are labeled with terminals. The children of an internal node are labeled from left to right with the right-hand side of the production used.

The vertices of a derivation tree are labeled with terminal or variable symbols of the grammar or

Definition: A derivation G = (V, T, P, S) be a CFG. A tree is a derivation (or parse) tree for G if:

- The root has label S.
- Interior nodes are labeled by variables
- Leaf nodes are labeled by T U { }
- If any node 'X is labeled by A and its children are labeled from left to right by $X_1,X_2\ldots..,X_n$ then $A \rightarrow X_1,X_2\ldots.,X_n$ must be production in P.

String of terminals is obtained by reading the leaves from left to right omitting

For example
Let G = *({S, A}. {a. b}. P, S),* where *P* consists of
S →aAs|a|SS
A→SbA |ba

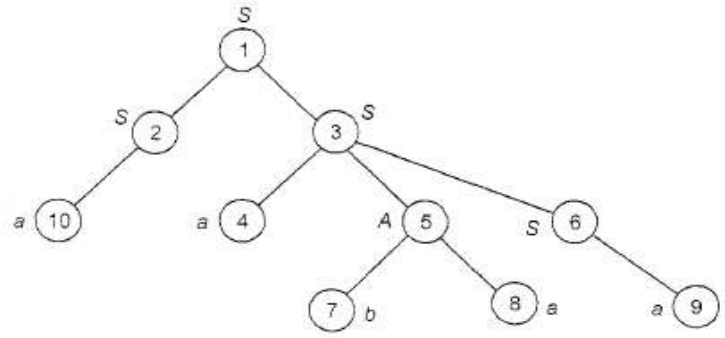


Fig. 6.1 An example of a derivation tree.

Note: Derivation Tree should not end up with variables.

Left most and Right Most derivations:

Left most derivation: A derivation A $\overset{*}{\Rightarrow}$ *w* is called a *leftmost* derivation if we apply a production only to the leftmost variable at every step.
Right Most derivation: *A* derivation *A* $\overset{*}{\Rightarrow}$ w is a *rightmost* derivation if we apply production to the rightmost variable at every step.

Sentential Form

A *sentential form* is any string derivable from the start symbol.

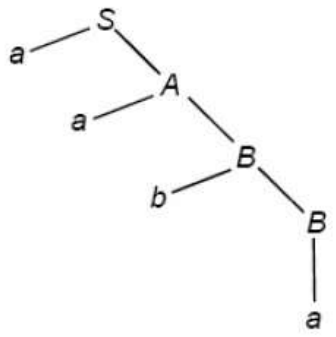
Let $G = (V, T, P, S)$ be a CFG, and $\alpha \in (V \cup T)^*$.
If

$$S \overset{*}{\Rightarrow} \alpha$$

we say that α is a *sentential form*.

If $S \underset{lm}{\Rightarrow} \alpha$ we say that α is a *left-sentential form*, and if $S \underset{rm}{\Rightarrow} \alpha$ we say that α is a *right-sentential form*

For a given CFG with productions $S \to aA, A \to aB, B \to bB, B \to a$. The derivation tree is as shown below.



$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aaba$$

The resultant of the derivation tree is the word $w = aaba$.
This is said to be in "Sentential Form".

Ambiguity in Context-Free Grammars

A terminal string w L(G) is ambiguous if there exist two or more derivation trees for w (or there exist two or more leftmost derivations of w).

Consider, for example, $G = (\{S\}, \{a, b, +, *\}, P, S)$, where P consists of $S \rightarrow S + S \mid S * S \mid a \mid b$. We have two derivation trees for $a + a * b$ given in Fig. 6.10.

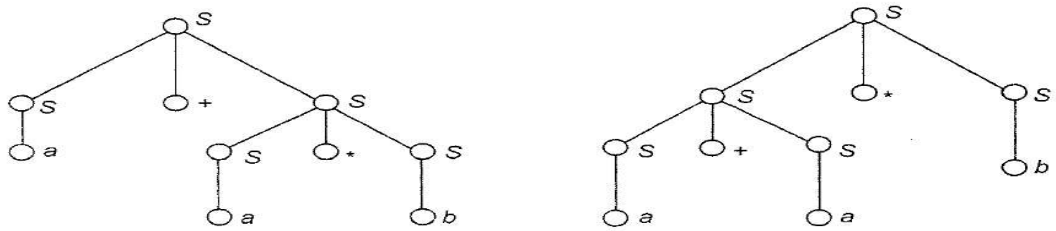


Fig. 6.10 Two derivation trees for $a + a * b$.

The leftmost derivations of $a + a * b$ induced by the two derivation trees are

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow a + S * S \Rightarrow a + a * S \Rightarrow a + a * b$$

Therefore, $a + a * b$ is ambiguous.